

User Manual for wxWindows 2.0: a portable C++ GUI toolkit

Julian Smart et al

February 28th 1999

Contents

Introduction	1
What is wxWindows?	1
Why another cross-platform development tool?	1
Changes from version 1.xx	2
wxWindows requirements.....	3
Availability and location of wxWindows	4
Acknowledgments.....	4
 Multi-platform development with wxWindows.....	6
Include files	6
Libraries	6
Configuration	6
Makefiles.....	7
Windows-specific files.....	7
Conditional compilation.....	8
C++ issues	8
File handling	9
 Programming strategies	11
Strategies for reducing programming errors.....	11
Strategies for portability	11
Strategies for debugging.....	11
 Alphabetical class reference	1
wxAcceleratorEntry	1
wxAcceleratorTable	2
wxActivateEvent	5
wxApp	6
wxArray	15
wxArrayString	25
wxAutomationObject.....	30
wxBusyCursor.....	35
wxButton	36
wxBitmap	39
wxBitmapHandler.....	50
wxBitmapButton.....	54
wxBitmapDataObject	59
wxBrush	61

wxBrushList.....	67
wxCalculateLayoutEvent	68
wxCheckBox	70
wxCheckListBox	72
wxChoice	75
wxClassInfo	79
wxClientDC	81
wxClipboard	82
wxCloseEvent	84
wxColour	86
wxColourData	89
wxColourDatabase	91
wxColourDialog.....	93
wxComboBox.....	94
wxCommand.....	101
wxCommandEvent.....	103
wxCommandProcessor.....	108
wxCondition	110
wxConfigBase	111
wxControl	125
wxCriticalSection	126
wxCriticalSectionLocker.....	127
wxCursor.....	128
wxDatabase	132
wxDataObject	138
wxDataInputStream	140
wxDataOutputStream	141
wxDate	143
wxDC	151
wxDDEClient.....	166
wxDDEConnection.....	167
wxDDEServer	172
wxDebugContext	173
wxDebugStreamBuf.....	178
wxDialog	178
wxDirDialog.....	185
wxDocChildFrame	187
wxDocManager	189
wxDocMDIChildFrame.....	197
wxDocMDIParentFrame	199

wxDocParentFrame	200
wxDocTemplate	202
wxDocument	207
wxDropFilesEvent	214
wxDropSource	216
wxDropTarget	218
wxEraseEvent	220
wxEvent	221
wxEvtHandler	224
wxExpr	230
wxExprDatabase	237
wxFile	241
wxFileDataObject	247
wxFileDialog	248
wxFileDropTarget	252
wxFileHistory	253
wxFileInputStream	256
wxFileOutputStream	257
wxFileStream	258
wxFileType	258
wxFilterInputStream	262
wxFilterOutputStream	262
wxFocusEvent	263
wxFont	264
wxFontData	270
wxFontDialog	273
wxFontList	274
wxFrame	275
wxFTP	287
wxGauge	291
wxGDIObject	295
wxGenericValidator	295
wxGrid	297
wxHashTable	310
wxHelpController	313
wxHTTP	316
wxIdleEvent	317
wxIcon	318
wxImage	325
wxImageHandler	335

wxImageList	339
wxIndividualLayoutConstraint	342
wxInitDialogEvent	345
wxInputStream	346
wxIPv4address	348
wxJoystick	349
wxJoystickEvent	356
wxKeyEvent	359
wxLayoutAlgorithm	362
wxLayoutConstraints	365
wxList	367
wxListBox	373
wxListCtrl	381
wxListEvent	394
wxLocale	396
wxLog	399
wxMask	403
wxMDIChildFrame	404
wxMDIClientWindow	407
wxMDIParentFrame	409
wxMemoryDC	415
wxMemoryInputStream	417
wxMemoryOutputStream	417
wxMenu	418
wxMenuBar	426
wxMenuItem	433
wxMenuEvent	438
wxMessageDialog	439
wxMetafile	440
wxMetafileDC	442
wxMimeTypesManager	443
wxMiniFrame	446
wxModule	448
wxMouseEvent	450
wxMoveEvent	458
wxMultipleChoiceDialog	459
wxMutex	459
wxMutexLocker	462
wxNodeBase	463
wxNotebook	464

wxNotebookEvent	470
wxObject	471
wxObjectRefData	475
wxOutputStream	476
wxPageSetupData	477
wxPageSetupDialog	482
wxPaintDC	483
wxPaintEvent	484
wxPalette	484
wxPanel	488
wxPanelTabView	491
wxPathList.....	492
wxPen	494
wxPenList.....	501
wxPoint	503
wxPostScriptDC.....	504
wxPreviewCanvas	504
wxPreviewControlBar	505
wxPreviewFrame	507
wxPrintData.....	508
wxPrintDialog	512
wxPrinter	513
wxPrinterDC.....	515
wxPrintout	516
wxPrintPreview	519
wxPrivateDataObject	523
wxPrivateDropTarget	525
wxProcess	525
wxProcessEvent	527
wxProtocol	528
wxQueryCol	530
wxQueryField.....	533
wxQueryLayoutInfoEvent.....	535
wxRadioBox.....	537
wxRadioButton.....	543
wxRealPoint.....	546
wxRect	546
wxRecordSet	550
wxRegion	562
wxRegionIterator.....	566

wxSashEvent	569
wxSashLayoutWindow	570
wxSashWindow	573
wxScreenDC	578
wxScrollBar	579
wxScrollEvent	584
wxScrolledWindow	586
wxSingleChoiceDialog	593
wxSize	595
wxSizeEvent	596
wxSlider	597
wxSocketAddress	605
wxSocketBase	607
wxSocketClient	615
wxSocketEvent	617
wxSocketHandler	618
wxSocketServer	620
wxSocketInputStream	622
wxSocketOutputStream	622
wxSpinButton	622
wxSplitterWindow	626
wxStaticBitmap	634
wxStaticBox	637
wxStaticText	638
wxStatusBar	640
wxStreamBase	646
wxStreamBuffer	648
wxString	654
wxStringList	674
wxStringTokenizer	676
wxSysColourChangedEvent	678
wxSystemSettings	678
wxTabbedDialog	682
wxTabbedPanel	683
wxTabControl	684
wxTabView	687
wxTabCtrl	695
wxTabEvent	700
wxTaskBarIcon	701
wxTCPClient	703

wxTCPConnection	705
wxTCPServer.....	709
wxTempFile	710
wxTextCtrl.....	712
wxTextDataObject	723
wxTextEntryDialog.....	725
wxTextDropTarget	726
wxTextValidator	728
wxTextFile.....	730
wxThread	736
wxTime	740
wxTimer	745
wxToolBar	746
wxTreeCtrl.....	761
wxTreeItemData	773
wxTreeEvent.....	774
wxUpdateUIEvent	775
wxURL	779
wxValidator	781
wxVariant	783
wxVariantData	792
wxView.....	793
wxWave	797
wxWindow.....	798
wxWindowDC	843
wxZlibInputStream	844
wxZlibOutputStream	844
Functions	845
File functions.....	845
String functions	850
Dialog functions	852
GDI functions	856
Printer settings	857
Clipboard functions	859
Miscellaneous functions.....	861
Macros	876
wxWindows resource functions	881
Log functions	884
Debugging macros and functions	886

Keycodes	888
Classes by category.....	890
Topic overviews	898
wxApp overview	898
wxString overview	899
Container classes overview	903
Log classes overview	904
Config classes overview	907
Bitmaps and icons overview	908
wxDialog overview	910
Font overview	911
wxSplitterWindow overview	912
wxTreeCtrl overview	913
wxListCtrl overview	914
wxImageList overview.....	915
Common dialogs overview.....	915
Constraints overview	919
Database classes overview	922
Device context overview	926
Debugging overview	927
Window deletion overview	929
Scrolling overview	932
Document/view overview	933
Event handling overview.....	939
Writing a wxWindows application: a rough guide	945
Interprocess communication overview	946
Printing overview	950
The wxWindows resource system	951
Run time class information overview	958
Window styles	959
Tab classes overview	959
wxTabView overview	963
Toolbar overview	963
Validator overview	969
wxExpr overview	971
wxGrid classes overview	974
Drag-and-drop and clipboard overview	975
Multithreading overview	977
File classes and functions overview	977

Internationalization.....	978
Notes on using the reference.....	979
wxPython Notes	1
What is wxPython?	1
Why use wxPython?	1
Other Python GUIs	2
Building wxPython	2
Using wxPython	4
wxWindows classes implemented in wxPython.....	7
Where to go for help	9
Index.....	12

Copyright notice

Copyright © 1998 Julian Smart, Robert Roebling and other members of the wxWindows
team

Portions © 1996 Artificial Intelligence Applications Institute

Please see the wxWindows licence files (preamble.txt, lgpl.txt, gpl.txt, licence.txt, licendoc.txt) for conditions of software and documentation use.

1 Introduction

1.1 What is wxWindows?

wxWindows is a C++ framework providing GUI (Graphical User Interface) and other facilities on more than one platform. Version 2.0 currently supports MS Windows (16-bit, Windows 95 and Windows NT), Unix with GTK+, and Unix with Motif. A Mac port is in an advanced state.

wxWindows was originally developed at the Artificial Intelligence Applications Institute, University of Edinburgh, for internal use. wxWindows has been released into the public domain in the hope that others will also find it useful. Version 2.0 is written and maintained by Julian Smart, Robert Roebling and others.

This manual discusses wxWindows in the context of multi-platform development.

Please note that in the following, "MS Windows" often refers to all platforms related to Microsoft Windows, including 16-bit and 32-bit variants, unless otherwise stated. All trademarks are acknowledged.

1.2 Why another cross-platform development tool?

wxWindows was developed to provide a cheap and flexible way to maximize investment in GUI application development. While a number of commercial class libraries already existed for cross-platform development, none met all of the following criteria:

1. low price;
2. source availability;
3. simplicity of programming;
4. support for a wide range of compilers.

Since wxWindows was started, several other free or almost-free GUI frameworks have emerged. However, none has the range of features, flexibility, documentation and the well-established development team that wxWindows has.

As public domain software and a project open to everyone, wxWindows has benefited from comments, ideas, bug fixes, enhancements and the sheer enthusiasm of users, especially via the Internet. This gives wxWindows a certain advantage over its commercial competitors (and over free libraries without an independent development team), plus a robustness against the transience of one individual or company. This openness and availability of source code is especially important when the future of thousands of lines of application code may depend upon the longevity of the underlying class library.

Version 2.0 goes much further than previous versions in terms of generality and features, allowing applications to be produced that are often indistinguishable from those produced using single-platform toolkits such as Motif and MFC.

The importance of using a platform-independent class library cannot be overstated, since GUI application development is very time-consuming, and sustained popularity of

particular GUIs cannot be guaranteed. Code can very quickly become obsolete if it addresses the wrong platform or audience. wxWindows helps to insulate the programmer from these winds of change. Although wxWindows may not be suitable for every application (such as an OLE-intensive program), it provides access to most of the functionality a GUI program normally requires, plus some extras such as network programming and PostScript output, and can of course be extended as needs dictate. As a bonus, it provides a cleaner programming interface than the native APIs. Programmers may find it worthwhile to use wxWindows even if they are developing on only one platform.

It is impossible to sum up the functionality of wxWindows in a few paragraphs, but here are some of the benefits:

- Low cost (free, in fact!)
- You get the source.
- Available on a variety of popular platforms.
- Works with almost all popular C++ compilers.
- Several example programs.
- Over 900 pages of printable and on-line documentation.
- Includes Tex2RTF, to allow you to produce your own documentation in Windows Help, HTML and Word RTF formats.
- Simple-to-use, object-oriented API.
- Flexible event system.
- Graphics calls include lines, rounded rectangles, splines, polylines, etc.
- Constraint-based layout option.
- Print/preview and document/view architectures.
- Toolbar, notebook, tree control, advanced list control classes.
- PostScript generation under Unix, normal MS Windows printing on the PC.
- MDI (Multiple Document Interface) support.
- Can be used to create DLLs under Windows, dynamic libraries on Unix.
- Common dialogs for file browsing, printing, colour selection, etc.
- Under MS Windows, support for creating metafiles and copying them to the clipboard.
- An API for invoking help from applications.
- Dialog Editor for building dialogs.
- Network support via a family of socket and protocol classes.

1.3 Changes from version 1.xx

These are a few of the major differences between versions 1.xx and 2.0.

Removals:

- XView is no longer supported;
- all controls (panel items) no longer have labels attached to them;
- wxForm has been removed;
- wxCanvasDC, wxPanelDC removed (replaced by wxClientDC, wxWindowDC, wxPaintDC which can be used for any window);
- wxMultiText, wxTextWindow, wxText removed and replaced by wxTextCtrl;

- classes no longer divided into generic and platform-specific parts, for efficiency.

Additions and changes:

- class hierarchy changed, and restrictions about subwindow nesting lifted;
- header files reorganised to conform to normal C++ standards;
- classes less dependent on each another, to reduce executable size;
- wxString used instead of char* wherever possible;
- the number of separate but mandatory utilities reduced;
- the event system has been overhauled, with virtual functions and callbacks being replaced with MFC-like event tables;
- new controls, such as wxTreeCtrl, wxListCtrl, wxSpinButton;
- less inconsistency about what events can be handled, so for example mouse clicks or key presses on controls can now be intercepted;
- the status bar is now a separate class, wxStatusBar, and is implemented in generic wxWindows code;
- some renaming of controls for greater consistency;
- wxBitmap has the notion of bitmap handlers to allow for extension to new formats without ifdefing;
- new dialogs: wxPageSetupDialog, wxFileDialog, wxDirDialog, wxMessageDialog, wxSingleChoiceDialog, wxTextEntryDialog;
- GDI objects are reference-counted and are now passed to most functions by reference, making memory management far easier;
- wxSystemSettings class allows querying for various system-wide properties such as dialog font, colours, user interface element sizes, and so on;
- better platform look and feel conformance;
- toolbar functionality now separated out into a family of classes with the same API;
- device contexts are no longer accessed using wxWindow::GetDC - they are created temporarily with the window as an argument;
- events from sliders and scrollbars can be handled more flexibly;
- the handling of window close events has been changed in line with the new event system;
- the concept of *validator* has been added to allow much easier coding of the relationship between controls and application data;
- the documentation has been revised, with more cross-referencing.

Platform-specific changes:

- The Windows header file (windows.h) is no longer included by wxWindows headers;
- wx.dll supported under Visual C++;
- the full range of Windows 95 window decorations are supported, such as modal frame borders;
- MDI classes brought out of wxFrame into separate classes, and made more flexible.

1.4 wxWindows requirements

To make use of wxWindows, you currently need one or both of the following setups.

(a) PC:

1. A 486 or higher PC running MS Windows.
2. A Windows compiler: most are supported, but please see `install.txt` for details. Supported compilers include Microsoft Visual C++ 4.0 or higher, Borland C++, Cygwin, Metrowerks CodeWarrior.
3. At least 60 MB of disk space.

(b) Unix:

1. Almost any C++ compiler, including GNU C++.
2. Almost any Unix workstation, and one of: GTK+ 1.0, Motif 1.2 or higher, Lesstif.
3. At least 60 MB of disk space.

1.5 Availability and location of wxWindows

wxWindows is currently available from the Artificial Intelligence Applications Institute by anonymous FTP and World Wide Web:

```
ftp://www.remstar.com/pub/wxwin
http://www.wxwindows.org
```

1.6

Acknowledgments

Thanks are due to AIAI for being willing to release the original version of wxWindows into the public domain, and to our patient partners.

We would particularly like to thank the following for their contributions to wxWindows, and the many others who have been involved in the project over the years. Apologies for any unintentional omissions from this list. Yiorgos Adamopoulos, Jamshid Afshar, Alejandro Aguilar-Sierra, AIAI, Patrick Albert, Karsten Ballueder, Michael Bedward, Kai Bendorf, Yura Bidus, Keith Gary Boyce, Chris Breeze, Pete Britton, Ian Brown, C. Buckley, Dmitri Chubraev, Robin Corbet, Cecil Coupe, Andrew Davison, Neil Dudman, Robin Dunn, Hermann Dunkel, Jos van Eijndhoven, Tom Felici, Thomas Fetting, Matthew Flatt, Pasquale Foggia, Josep Fortiana, Todd Fries, Dominic Gallagher, Wolfram Gloger, Norbert Grotz, Stefan Gunter, Bill Hale, Patrick Halke, Stefan Hammes, Guillaume Helle, Harco de Hilster, Cord Hockemeyer, Markus Holzem, Olaf Klein, Leif Jensen, Bart Jourquin, Guilhem Lavaux, Jan Lessner, Nicholas Liebmann, Torsten Liermann, Per Lindqvist, Thomas Runge, Tatu Männistö, Scott Maxwell, Thomas Myers, Oliver Niedung, Hernan Otero, Ian Perrigo, Timothy Peters, Giordano Pezzoli, Harri Pasanen, Thomaso Paoletti, Garrett Potts, Marcel Rasche, Robert Roebing, Dino Scaringella, Jobst Schmalenbach, Arthur Seaton, Paul Shirley, Stein Somers, Petr Smilauer, Neil Smith, Kari Systä, Arthur Tetzlaff-Deas, Jonathan Tonberg, Jyrki Tuomi, Janos Vegh, Andrea Venturoli, Vadim Zeitlin, Xiaokun Zhu, Edward Zimmermann.

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

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

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

2 Multi-platform development with wxWindows

This chapter describes the practical details of using wxWindows. Please see the file `install.txt` for up-to-date installation instructions, and `changes.txt` for differences between versions.

2.1 Include files

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

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

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

#ifdef __BORLANDC__
#pragma hdrstop
#endif

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

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

The file `"wx/wxprec.h"` includes `"wx/wx.h"`. Although this incantation may seem quirky, it is in fact the end result of a lot of experimentation, and several Windows compilers to use precompilation (those tested are Microsoft Visual C++, Borland C++ and Watcom C++).

Borland precompilation is largely automatic. Visual C++ requires specification of `"wx/wxprec.h"` as the file to use for precompilation. Watcom C++ is automatic apart from the specification of the `.pch` file. Watcom C++ is strange in requiring the precompiled header to be used only for object files compiled in the same directory as that in which the precompiled header was created. Therefore, the wxWindows Watcom C++ makefiles go through hoops deleting and recreating a single precompiled header file for each module, thus preventing an accumulation of many multi-megabyte `.pch` files.

2.2 Libraries

Please the wxGTK or wxMotif documentation for use of the Unix version of wxWindows. Under Windows, use the library `wx.lib` for stand-alone Windows applications, or `wxdll.lib` for creating DLLs.

2.3 Configuration

Options are configurable in the file `"wx/XXX/setup.h"` where XXX is the required platform (such as `msw`, `motif`, `gtk`, `mac`). Some settings are a matter of taste, some help

with platform-specific problems, and others can be set to minimize the size of the library. Please see the `setup.h` file and `install.txt` files for details on configuration.

2.4 Makefiles

At the moment there is no attempt to make Unix makefiles and PC makefiles compatible, i.e. one makefile is required for each environment. wxGTK has its own configure system which can also be used with wxMotif, although wxMotif has a simple makefile system of its own.

Sample makefiles for Unix (suffix `.UNX`), MS C++ (suffix `.DOS` and `.NT`), Borland C++ (`.BCC` and `.B32`) and Symantec C++ (`.SC`) are included for the library, demos and utilities.

The controlling makefile for wxWindows is in the platform-specific directory, such as `src/msw` or `src/motif`.

Please see the platform-specific `install.txt` file for further details.

2.5 Windows-specific files

wxWindows application compilation under MS Windows requires at least two extra files, resource and module definition files.

2.5.1 Resource file

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

```
rcinclude "wx/msw/wx.rc"
```

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

```
wxicon icon wx.ico
```

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

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

2.5.2 Module definition file

A module definition file (extension `DEF`) is required for 16-bit applications, and looks like the following:

```
NAME            Hello
DESCRIPTION     'Hello'
```

```
EXETYPE      WINDOWS
STUB         'WINSTUB.EXE'
CODE         PRELOAD MOVEABLE DISCARDABLE
DATA         PRELOAD MOVEABLE MULTIPLE
HEAPSIZE     1024
STACKSIZE    8192
```

The only lines which will usually have to be changed per application are NAME and DESCRIPTION.

2.5.3 Allocating and deleting wxWindows objects

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

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

Don't create a window on the stack, because this will interfere with delayed deletion.

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

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

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

2.6 Conditional compilation

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

2.7 C++ issues

The following documents some miscellaneous C++ issues.

2.7.1 Templates

wxWindows does not use templates since it is a notoriously unportable feature.

2.7.2 RTTI

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

2.7.3 Type of NULL

Some compilers (e.g. the native IRIX cc) define NULL to be 0L so that no conversion to pointers is allowed. Because of that, all these occurrences of NULL in the GTK port use an explicit conversion such as

```
wxWindow *my_window = (wxWindow*) NULL;
```

It is recommended to adhere to this in all code using wxWindows as this make the code (a bit) more portable.

2.7.4 Precompiled headers

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

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

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

2.8 File handling

When building an application which may be used under different environments, one

difficulty is coping with documents which may be moved to different directories on other machines. Saving a file which has pointers to full pathnames is going to be inherently unportable. One approach is to store filenames on their own, with no directory information. The application searches through a number of locally defined directories to find the file. To support this, the class **wxPathList** makes adding directories and searching for files easy, and the global function **wxFileNameFromPath** allows the application to strip off the filename from the path if the filename must be stored. This has undesirable ramifications for people who have documents of the same name in different directories.

As regards the limitations of DOS 8+3 single-case filenames versus unrestricted Unix filenames, the best solution is to use DOS filenames for your application, and also for document filenames *if* the user is likely to be switching platforms regularly. Obviously this latter choice is up to the application user to decide. Some programs (such as YACC and LEX) generate filenames incompatible with DOS; the best solution here is to have your Unix makefile rename the generated files to something more compatible before transferring the source to DOS. Transferring DOS files to Unix is no problem, of course, apart from EOL conversion for which there should be a utility available (such as dos2unix).

See also the File Functions section of the reference manual for descriptions of miscellaneous file handling functions.

3 Programming strategies

This chapter is intended to list strategies that may be useful when writing and debugging wxWindows programs. If you have any good tips, please submit them for inclusion here.

3.1 Strategies for reducing programming errors

3.1.1 Use ASSERT

Although I haven't done this myself within wxWindows, it is good practice to use ASSERT statements liberally, that check for conditions that should or should not hold, and print out appropriate error messages. These can be compiled out of a non-debugging version of wxWindows and your application. Using ASSERT is an example of 'defensive programming': it can alert you to problems later on.

3.1.2 Use wxString in preference to character arrays

Using wxString can be much safer and more convenient than using char *. Again, I haven't practised what I'm preaching, but I'm now trying to use wxString wherever possible. You can reduce the possibility of memory leaks substantially, and it's much more convenient to use the overloaded operators than functions such as strcmp. wxString won't add a significant overhead to your program; the overhead is compensated for by easier manipulation (which means less code).

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

3.2 Strategies for portability

3.2.1 Use relative positioning or constraints

Don't use absolute panel item positioning if you can avoid it. Different GUIs have very differently sized panel items. Consider using the constraint system, although this can be complex to program.

Alternatively, you could use alternative .wrc (wxWindows resource files) on different platforms, with slightly different dimensions in each. Or space your panel items out to avoid problems.

3.2.2 Use wxWindows resource files

Use .wrc (wxWindows resource files) where possible, because they can be easily changed independently of source code. Bitmap resources can be set up to load different kinds of bitmap depending on platform (see the section on resource files).

3.3 Strategies for debugging

3.3.1 Positive thinking

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

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

3.3.2 Simplify the problem

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

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

3.3.3 Use a debugger

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

3.3.4 Use logging functions

There is a variety of logging functions that you can use in your program: see *Logging functions* (p. 884).

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

3.3.5 Use the wxWindows debugging facilities

You can use `wxDebugContext` to check for memory leaks and corrupt memory: in fact in debugging mode, wxWindows will automatically check for memory leaks at the end of the program if wxWindows is suitably configured. Depending on the operating system

and compiler, more or less specific information about the problem will be logged.

You should also use *debug macros* (p. 886) as part of a 'defensive programming' strategy, scattering `wxASSERT`s liberally to test for problems in your code as early as possible. Forward thinking will save a surprising amount of time in the long run.

See the *debugging overview* (p. 927) for further information.

3.3.6 Check Windows debug messages

Under Windows, it's worth running your program with DBWIN running or some other program that shows Windows-generated debug messages. It's possible it'll show invalid handles being used. You may have fun seeing what commercial programs cause these normally hidden errors! Microsoft recommend using the debugging version of Windows, which shows up even more problems. However, I doubt it's worth the hassle for most applications. `wxWindows` is designed to minimize the possibility of such errors, but they can still happen occasionally, slipping through unnoticed because they are not severe enough to cause a crash.

3.3.7 Genetic mutation

If we had sophisticated genetic algorithm tools that could be applied to programming, we could use them. Until then, a common -- if rather irrational -- technique is to just make arbitrary changes to the code until something different happens. You may have an intuition why a change will make a difference; otherwise, just try altering the order of code, comment lines out, anything to get over an impasse. Obviously, this is usually a last resort.

4 Alphabetical class reference

4.1 wxAcceleratorEntry

An object used by an application wishing to create an *accelerator table* (p. 2).

Derived from

None

Include files

<wx/accel.h>

See also

wxAcceleratorTable (p. 2), *wxWindow::SetAcceleratorTable* (p. 832)

4.1.1 wxAcceleratorEntry::wxAcceleratorEntry

wxAcceleratorEntry()

Default constructor.

wxAcceleratorEntry(int flags, int keyCode, int cmd)

Constructor.

Parameters

flags

One of wxACCEL_SHIFT, wxACCEL_CTRL and wxACCEL_NORMAL. Indicates which modifier key is held down.

keyCode

The keycode to be detected. See *Keycodes* (p. 888) for a full list of keycodes.

cmd

The menu or control command identifier.

4.1.2 wxAcceleratorEntry::GetCommand

int GetCommand() const

Returns the command identifier for the accelerator table entry.

4.1.3 wxAcceleratorEntry::GetFlags

int GetFlags() const

Returns the flags for the accelerator table entry.

4.1.4 wxAcceleratorEntry::GetKeyCode

int GetKeyCode() const

Returns the keycode for the accelerator table entry.

4.1.5 wxAcceleratorEntry::Set

void Set(int flags, int keyCode, int cmd)

Sets the accelerator entry parameters.

Parameters

flags

One of wxACCEL_SHIFT, wxACCEL_CTRL and wxACCEL_NORMAL. Indicates which modifier key is held down.

keyCode

The keycode to be detected. See *Keycodes* (p. 888) for a full list of keycodes.

cmd

The menu or control command identifier.

4.2 wxAcceleratorTable

An accelerator table allows the application to specify a table of keyboard shortcuts for menus or other commands. On Windows, menu or button commands are supported; on GTK, only menu commands are supported.

The object **wxNullAcceleratorTable** is defined to be a table with no data, and is the initial accelerator table for a window.

Derived from

wxObject (p. 471)

Include files

<wx/accel.h>

Example

```
wxAcceleratorEntry entries[4];
entries[0].Set(wxACCEL_CTRL, (int) 'N', ID_NEW_WINDOW);
entries[1].Set(wxACCEL_CTRL, (int) 'X', wxID_EXIT);
entries[2].Set(wxACCEL_SHIFT, (int) 'A', ID_ABOUT);
entries[3].Set(wxACCEL_NONE, W XK_DELETE, wxID_CUT);
wxAcceleratorTable accel(4, entries);
frame->SetAcceleratorTable(accel);
```

Remarks

An accelerator takes precedence over normal processing and can be a convenient way to program some event handling. For example, you can use an accelerator table to enable a dialog with a multi-line text control to accept CTRL-Enter as meaning 'OK' (but not in GTK at present).

See also

wxAcceleratorEntry (p. 1), *wxWindow::SetAcceleratorTable* (p. 832)

4.2.1 wxAcceleratorTable::wxAcceleratorTable

wxAcceleratorTable()

Default constructor.

wxAcceleratorTable(const wxAcceleratorTable& *bitmap*)

Copy constructor.

wxAcceleratorTable(int *n*, wxAcceleratorEntry *entries*[])

Creates from an array of *wxAcceleratorEntry* (p. 1) objects.

wxAcceleratorTable(const wxString& *resource*)

Loads the accelerator table from a Windows resource (Windows only).

Parameters

n

Number of accelerator entries.

entries

The array of entries.

resource

Name of a Windows accelerator.

4.2.2 wxAcceleratorTable::~~wxAcceleratorTable

~wxAcceleratorTable()

Destroys the wxAcceleratorTable object.

4.2.3 wxAcceleratorTable::Ok

bool Ok() const

Returns TRUE if the accelerator table is valid.

4.2.4 wxAcceleratorTable::operator =

wxAcceleratorTable& operator =(const wxAcceleratorTable& *accel*)

Assignment operator. This operator does not copy any data, but instead passes a pointer to the data in *accel* and increments a reference counter. It is a fast operation.

Parameters

accel

Accelerator table to assign.

Return value

Returns 'this' object.

4.2.5 wxAcceleratorTable::operator ==

bool operator ==(const wxAcceleratorTable& *accel*)

Equality operator. This operator tests whether the internal data pointers are equal (a fast test).

Parameters

accel

Accelerator table to compare with 'this'

Return value

Returns TRUE if the accelerator tables were effectively equal, FALSE otherwise.

4.2.6 wxAcceleratorTable::operator !=

bool operator !=(const wxAcceleratorTable& accel)

Inequality operator. This operator tests whether the internal data pointers are unequal (a fast test).

Parameters

accel

Accelerator table to compare with 'this'

Return value

Returns TRUE if the accelerator tables were unequal, FALSE otherwise.

4.3 wxActivateEvent

An activate event is sent when a window or application is being activated or deactivated.

Derived from

wxEvent (p. 221)

wxObject (p. 471)

Include files

<wx/event.h>

Event table macros

To process an activate event, use these event handler macros to direct input to a member function that takes a wxActivateEvent argument.

EVT_ACTIVATE(func)

Process a wxEVT_ACTIVATE event.

EVT_ACTIVATE_APP(func)

Process a wxEVT_ACTIVATE_APP event.

Remarks

A top-level window (a dialog or frame) receives an activate event when is being activated or deactivated. This is indicated visually by the title bar changing colour, and a subwindow gaining the keyboard focus.

An application is activated or deactivated when one of its frames becomes activated, or a frame becomes inactivate resulting in all application frames being inactive. (Windows only)

See also

wxWindow::OnActivate (p. 817), *wxApp::OnActivate* (p. 10), *Event handling overview* (p. 939)

4.3.1 wxActivateEvent::wxActivateEvent

wxActivateEvent(WXTYPE *eventType* = 0, **bool** *active* = TRUE, **int** *id* = 0)

Constructor.

4.3.2 wxActivateEvent::m_active

bool *m_active*

TRUE if the window or application was activated.

4.3.3 wxActivateEvent::GetActive

bool *GetActive*() **const**

Returns TRUE if the application or window is being activated, FALSE otherwise.

4.4 wxApp

The **wxApp** class represents the application itself. It is used to:

- set and get application-wide properties;
- implement the windowing system message or event loop;
- initiate application processing via *wxApp::OnInit* (p. 12);
- allow default processing of events not handled by other objects in the application.

You should use the macro `IMPLEMENT_APP(appClass)` in your application implementation file to tell wxWindows how to create an instance of your application class.

Use `DECLARE_APP(appClass)` in a header file if you want the `wxGetApp` function (which returns a reference to your application object) to be visible to other files.

Derived from

wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/app.h>

See also

wxApp overview (p. 898)

4.4.1 wxApp::wxApp

void wxApp()

Constructor. Called implicitly with a definition of a wxApp object.

The argument is a language identifier; this is an experimental feature and will be expanded and documented in future versions.

4.4.2 wxApp::~~wxApp

void ~wxApp()

Destructor. Will be called implicitly on program exit if the wxApp object is created on the stack.

4.4.3 wxApp::argc

int argc

Number of command line arguments (after environment-specific processing).

4.4.4 wxApp::argv

char ** argv

Command line arguments (after environment-specific processing).

4.4.5 wxApp::CreateLogTarget

virtual wxLog* CreateLogTarget()

Creates a wxLog class for the application to use for logging errors. The default implementation returns a new wxLogGui class.

See also

wxLog (p. 399)

4.4.6 wxApp::Dispatch

void Dispatch()

Dispatches the next event in the windowing system event queue.

This can be used for programming event loops, e.g.

```
while (app.Pending())  
    Dispatch();
```

See also

wxApp::Pending (p. 13)

4.4.7 wxApp::GetAppName

wxString GetAppName() const

Returns the application name.

Remarks

wxWindows sets this to a reasonable default before calling *wxApp::OnInit* (p. 12), but the application can reset it at will.

4.4.8 wxApp::GetAuto3D

bool GetAuto3D() const

Returns TRUE if 3D control mode is on, FALSE otherwise.

See also

wxApp::SetAuto3D (p. 14)

4.4.9 wxApp::GetClassName

wxString GetClassName() const

Gets the class name of the application. The class name may be used in a platform specific manner to refer to the application.

See also

wxApp::SetClassName (p. 14)

4.4.10 wxApp::GetExitOnDelete

bool GetExitOnDelete() const

Returns TRUE if the application will exit when the top-level window is deleted, FALSE otherwise.

See also

wxApp::SetExitOnDelete (p. 14)

4.4.11 wxApp::GetTopWindow

wxWindow * GetTopWindow() const

Returns a pointer to the top window.

Remarks

If the top window hasn't been set using *wxApp::SetTopWindow* (p. 15), this function will find the first top-level window (frame or dialog) and return that.

See also

wxApp::SetTopWindow (p. 15)

4.4.12 wxApp::ExitMainLoop

void ExitMainLoop()

Call this to explicitly exit the main message (event) loop. You should normally exit the main loop (and the application) by deleting the top window.

4.4.13 wxApp::Initialized

bool Initialized()

Returns TRUE if the application has been initialized (i.e. if *wxApp::OnInit* (p. 12) has returned successfully). This can be useful for error message routines to determine which method of output is best for the current state of the program (some windowing systems may not like dialogs to pop up before the main loop has been entered).

4.4.14 wxApp::MainLoop

int MainLoop()

Called by wxWindows on creation of the application. Override this if you wish to provide your own (environment-dependent) main loop.

Return value

Returns 0 under X, and the wParam of the WM_QUIT message under Windows.

4.4.15 wxApp::OnActivate

void OnActivate(wxActivateEvent& event)

Provide this member function to know whether the application is being activated or deactivated (Windows only).

See also

wxWindow::OnActivate (p. 817), *wxActivateEvent* (p. 5)

4.4.16 wxApp::OnExit

int OnExit()

Provide this member function for any processing which needs to be done as the application is about to exit.

4.4.17 wxApp::OnCharHook

void OnCharHook(wxKeyEvent& event)

This event handler function is called (under Windows only) to allow the window to intercept keyboard events before they are processed by child windows.

Parameters

event
The keypress event.

Remarks

Use the wxEVT_CHAR_HOOK macro in your event table.

If you use this member, you can selectively consume keypress events by calling

wxEvtHandler::Skip (p. 224) for characters the application is not interested in.

See also

wxKeyEvent (p. 359), *wxWindow::OnChar* (p. 817), *wxWindow::OnCharHook* (p. 818), *wxDialog::OnCharHook* (p. 182)

4.4.18 wxApp::OnIdle

void OnIdle(*wxIdleEvent*& *event*)

Override this member function for any processing which needs to be done when the application is idle. You should call *wxApp::OnIdle* from your own function, since this forwards OnIdle events to windows and also performs garbage collection for windows whose destruction has been delayed.

wxWindows' strategy for OnIdle processing is as follows. After pending user interface events for an application have all been processed, *wxWindows* sends an OnIdle event to the application object. *wxApp::OnIdle* itself sends an OnIdle event to each application window, allowing windows to do idle processing such as updating their appearance. If either *wxApp::OnIdle* or a window OnIdle function requested more time, by calling *wxIdleEvent::RequestMore* (p. 318), *wxWindows* will send another OnIdle event to the application object. This will occur in a loop until either a user event is found to be pending, or OnIdle requests no more time. Then all pending user events are processed until the system goes idle again, when OnIdle is called, and so on.

See also

wxWindow::OnIdle (p. 823), *wxIdleEvent* (p. 317), *wxWindow::SendIdleEvents* (p. 13)

4.4.19 wxApp::OnEndSession

void OnEndSession(*wxCloseEvent*& *event*)

This is an event handler function called when the operating system or GUI session is about to close down. The application has a chance to silently save information, and can optionally close itself.

Use the `EVT_END_SESSION` event table macro to handle query end session events.

The default handler calls *wxWindow::Close* (p. 802) with a `TRUE` argument (forcing the application to close itself silently).

Remarks

Under X, *OnEndSession* is called in response to the 'die' event.

Under Windows, *OnEndSession* is called in response to the `WM_ENDSESSION` message.

See also

wxWindow::Close (p. 802), *wxWindow::OnCloseWindow* (p. 819), *wxCloseEvent* (p. 84), *wxApp::OnQueryEndSession* (p. 12)

4.4.20 wxApp::OnInit

bool OnInit()

This must be provided by the application, and will usually create the application's main window, optionally calling *wxApp::SetTopWindow* (p. 15).

Return TRUE to continue processing, FALSE to exit the application.

4.4.21 wxApp::OnQueryEndSession

void OnQueryEndSession(wxCloseEvent& event)

This is an event handler function called when the operating system or GUI session is about to close down. Typically, an application will try to save unsaved documents at this point.

If *wxCloseEvent::CanVeto* (p. 85) returns TRUE, the application is allowed to veto the shutdown by calling *wxCloseEvent::Veto* (p. 86). The application might veto the shutdown after prompting for documents to be saved, and the user has cancelled the save.

Use the `EVT_QUERY_END_SESSION` event table macro to handle query end session events.

You should check whether the application is forcing the deletion of the window using *wxCloseEvent::GetForce* (p. 85). If this is TRUE, destroy the window using *wxWindow::Destroy* (p. 805). If not, it is up to you whether you respond by destroying the window.

The default handler calls *wxWindow::Close* (p. 802) on the top-level window, and vetoes the shutdown if *Close* returns FALSE. This will be sufficient for many applications.

Remarks

Under X, *OnQueryEndSession* is called in response to the 'save session' event.

Under Windows, *OnQueryEndSession* is called in response to the `WM_QUERYENDSESSION` message.

See also

wxWindow::Close (p. 802), *wxWindow::OnCloseWindow* (p. 819), *wxCloseEvent* (p. 84),

wxApp::OnEndSession (p. 11)

4.4.22 wxApp::ProcessMessage

bool ProcessMessage(MSG *msg)

Windows-only function for processing a message. This function is called from the main message loop, checking for windows that may wish to process it. The function returns TRUE if the message was processed, FALSE otherwise. If you use wxWindows with another class library with its own message loop, you should make sure that this function is called to allow wxWindows to receive messages. For example, to allow co-existence with the Microsoft Foundation Classes, override the PreTranslateMessage function:

```
// Provide wxWindows message loop compatibility
BOOL CTheApp::PreTranslateMessage(MSG *msg)
{
    if (wxTheApp && wxTheApp->ProcessMessage(msg))
        return TRUE;
    else
        return CWinApp::PreTranslateMessage(msg);
}
```

4.4.23 wxApp::Pending

bool Pending()

Returns TRUE if unprocessed events are in the window system event queue (MS Windows and Motif).

[See also](#)

wxApp::Dispatch (p. 8)

4.4.24 wxApp::SendIdleEvents

bool SendIdleEvents()

Sends idle events to all top-level windows.

bool SendIdleEvents(wxWindow* win)

Sends idle events to a window and its children.

[Remarks](#)

These functions poll the top-level windows, and their children, for idle event processing. If TRUE is returned, more OnIdle processing is requested by one or more window.

[See also](#)

wxApp::OnIdle (p. 11), *wxWindow::OnIdle* (p. 823), *wxIdleEvent* (p. 317)

4.4.25 **wxApp::SetAppName**

void SetAppName(const wxString& name)

Sets the name of the application. The name may be used in dialogs (for example by the document/view framework). A default name is set by wxWindows.

[See also](#)

wxApp::GetAppName (p. 8)

4.4.26 **wxApp::SetAuto3D**

void SetAuto3D(const bool auto3D)

Switches automatic 3D controls on or off.

Parameters

auto3D

If TRUE, all controls will be created with 3D appearances unless overridden for a control or dialog. The default is TRUE

Remarks

This has an effect on Windows only.

[See also](#)

wxApp::GetAuto3D (p. 8)

4.4.27 **wxApp::SetClassName**

void SetClassName(const wxString& name)

Sets the class name of the application. This may be used in a platform specific manner to refer to the application.

[See also](#)

wxApp::GetClassName (p. 8)

4.4.28 **wxApp::SetExitOnDelete**

void SetExitOnDelete(bool *flag*)

Allows the programmer to specify whether the application will exit when the top-level frame is deleted.

Parameters

flag

If TRUE (the default), the application will exit when the top-level frame is deleted. If FALSE, the application will continue to run.

Remarks

Currently, setting this to FALSE only has an effect under Windows.

4.4.29 wxApp::SetTopWindow**void SetTopWindow(wxWindow* *window*)**

Sets the 'top' window. You can call this from within *wxApp::OnInit* (p. 12) to let *wxWindows* know which is the main window. You don't have to set the top window; it's only a convenience so that (for example) certain dialogs without parents can use a specific window as the top window. If no top window is specified by the application, *wxWindows* just uses the first frame or dialog in its top-level window list, when it needs to use the top window.

Parameters

window

The new top window.

See also

wxApp::GetTopWindow (p. 9), *wxApp::OnInit* (p. 12)

4.5 wxArray

This section describes the so called *dynamic arrays*. This is a C array-like data structure i.e. the member access time is constant (and not linear according to the number of container elements as for linked lists). However, these arrays are dynamic in the sense that they will automatically allocate more memory if there is not enough of it for adding a new element. They also perform range checking on the index values but in debug mode only, so please be sure to compile your application in debug mode to use it (see *debugging overview* (p. 927) for details). So, unlike the arrays in some other languages, attempt to access an element beyond the arrays bound doesn't automatically expand the array but provokes an assertion failure instead in debug build and does nothing (except possibly crashing your program) in the release build.

The array classes were designed to be reasonably efficient, both in terms of run-time

speed and memory consumption and the executable size. The speed of array item access is, of course, constant (independent of the number of elements) making them much more efficient than linked lists (*wxList* (p. 367)). Adding items to the arrays is also implemented in more or less constant time - but the price is preallocating the memory in advance. In the *memory management* (p. 18) section you may find some useful hints about optimizing *wxArray* memory usage. As for executable size, all *wxArray* functions are inline, so they do not take *any space at all*.

wxWindows has three different kinds of array. All of them derive from *wxBaseArray* class which works with untyped data and can not be used directly. The standard macros *WX_DEFINE_ARRAY()*, *WX_DEFINE_SORTED_ARRAY()* and *WX_DEFINE_OBJARRAY()* are used to define a new class deriving from it. The classes declared will be called in this documentation *wxArray*, *wxSortedArray* and *wxObjArray* but you should keep in mind that no classes with such names actually exist, each time you use one of *WX_DEFINE_XXXARRAY* macro you define a class with a new name. In fact, these names are "template" names and each usage of one of the macros mentioned above creates a template specialization for the given element type.

wxArray is suitable for storing integer types and pointers which it does not treat as objects in any way, i.e. the element pointed to by the pointer is not deleted when the element is removed from the array. It should be noted that all of *wxArray*'s functions are inline, so it costs strictly nothing to define as many array types as you want (either in terms of the executable size or the speed) as long as at least one of them is defined and this is always the case because *wxArrays* are used by *wxWindows* internally. This class has one serious limitation: it can only be used for storing integral types (*bool*, *char*, *short*, *int*, *long* and their unsigned variants) or pointers (of any kind). An attempt to use with objects of *sizeof()* greater than *sizeof(long)* will provoke a runtime assertion failure, however declaring a *wxArray* of floats will not (on the machines where *sizeof(float)* \leq *sizeof(long)*), yet it will **not** work, please use *wxObjArray* for storing floats and doubles (NB: a more efficient *wxArrayDouble* class is scheduled for the next release of *wxWindows*).

wxSortedArray is a *wxArray* variant which should be used when searching in the array is a frequently used operation. It requires you to define an additional function for comparing two elements of the array element type and always stores its items in the sorted order (according to this function). Thus, it's *Index()* (p. 23) function execution time is $O(\log(N))$ instead of $O(N)$ for the usual arrays but the *Add()* (p. 22) method is slower: it is $O(\log(N))$ instead of constant time (neglecting time spent in memory allocation routine). However, in a usual situation elements are added to an array much less often than searched inside it, so *wxSortedArray* may lead to huge performance improvements compared to *wxArray*. Finally, it should be noticed that, as *wxArray*, *wxSortedArray* can be only used for storing integral types or pointers.

wxObjArray class treats its elements like "objects". It may delete them when they are removed from the array (invoking the correct destructor) and copies them using the objects copy constructor. In order to implement this behaviour the definition of the *wxObjArray* arrays is split in two parts: first, you should declare the new *wxObjArray* class using *WX_DECLARE_OBJARRAY()* macro and then you must include the file defining the implementation of template type: *<wx/arrimpl.cpp>* and define the array class with *WX_DEFINE_OBJARRAY()* macro from a point where the full (as opposed to 'forward') declaration of the array elements class is in scope. As it probably sounds very

complicated here is an example:

```
#include <wx/dynarray.h>

// we must forward declare the array because it's used inside the class
// declaration
class MyDirectory;
class MyFile;

// this defines two new types: ArrayOfDirectories and ArrayOfFiles which
// can be
// now used as shown below
WX_DECLARE_OBJARRAY(MyDirectory, ArrayOfDirectories);
WX_DECLARE_OBJARRAY(MyFile, ArrayOfFiles);

class MyDirectory
{
...
    ArrayOfDirectories m_subdirectories; // all subdirectories
    ArrayOfFiles       m_files;       // all files in this directory
};

...

// now that we have MyDirectory declaration in scope we may finish the
// definition of ArrayOfDirectories
#include <wx/arrimpl.cpp> // this is a magic incantation which must be
done!
WX_DEFINE_OBJARRAY(ArrayOfDirectories);

// that's all!
```

It is not as elegant as writing

```
typedef std::vector<MyDirectory> ArrayOfDirectories;
```

but is not that complicated and allows the code to be compiled with any, however dumb, C++ compiler in the world.

Things are much simpler for `wxArray` and `wxSortedArray` however: it is enough just to write

```
WX_DEFINE_ARRAY(MyDirectory *, ArrayOfDirectories);
WX_DEFINE_SORTED_ARRAY(MyFile *, ArrayOfFiles);
```

See also:

Container classes overview (p. 903), *wxList* (p. 367)

Required headers:

`<wx/dynarray.h>` for `wxArray` and `wxSortedArray` and additionally `<wx/arrimpl.cpp>` for `wxObjArray`.

4.5.1 Macros for template array definition

To use an array you must first define the array class. This is done with the help of the macros in this section. The class of array elements must be (at least) forward declared for `WX_DEFINE_ARRAY`, `WX_DEFINE_SORTED_ARRAY` and `WX_DECLARE_OBJARRAY` macros and must be fully declared before you use `WX_DEFINE_OBJARRAY` macro.

WX_DEFINE_ARRAY (p. 19)

WX_DEFINE_SORTED_ARRAY (p. 19)

WX_DECLARE_OBJARRAY (p. 20)

WX_DEFINE_OBJARRAY (p. 20)

4.5.2 Constructors and destructors

Array classes are 100% C++ objects and as such they have the appropriate copy constructors and assignment operators. Copying `wxArray` just copies the elements but copying `wxObjArray` copies the arrays items. However, for memory-efficiency sake, neither of these classes has virtual destructor. It is not very important for `wxArray` which has trivial destructor anyhow, but it does mean that you should avoid deleting `wxObjArray` through a `wxBaseArray` pointer (as you would never use `wxBaseArray` anyhow it shouldn't be a problem) and that you should not derive your own classes from the array classes.

wxArray default constructor (p. 21)

wxArray copy constructors and assignment operators (p. 21)

~wxArray (p. 22)

4.5.3 Memory management

Automatic array memory management is quite trivial: the array starts by preallocating some minimal amount of memory (defined by `WX_ARRAY_DEFAULT_INITIAL_SIZE`) and when further new items exhaust already allocated memory it reallocates it adding 50% of the currently allocated amount, but no more than some maximal number which is defined by `ARRAY_MAXSIZE_INCREMENT` constant. Of course, this may lead to some memory being wasted (`ARRAY_MAXSIZE_INCREMENT` in the worst case, i.e. 4Kb in the current implementation), so the *Shrink()* (p. 25) function is provided to unallocate the extra memory. The *Alloc()* (p. 22) function can also be quite useful if you know in advance how many items you are going to put in the array and will prevent the array code from reallocating the memory more times than needed.

Alloc (p. 22)

Shrink (p. 25)

4.5.4 Number of elements and simple item access

Functions in this section return the total number of array elements and allow to retrieve them - possibly using just the C array indexing [] operator which does exactly the same as *Item()* (p. 24) method.

Count (p. 23)
GetCount (p. 23)
IsEmpty (p. 24)
Item (p. 24)
Last (p. 24)

4.5.5 Adding items

Add (p. 22)
Insert (p. 24)

4.5.6 Removing items

WX_CLEAR_ARRAY (p. 21)
Empty (p. 23)
Clear (p. 23)
Remove (p. 25)

4.5.7 Searching and sorting

Index (p. 23)
Sort (p. 25)

4.5.8 WX_DEFINE_ARRAY

WX_DEFINE_ARRAY(*T*, *name*)

This macro defines a new array class named *name* and containing the elements of type *T*. Example:

```
WX_DEFINE_ARRAY(int, wxArrayInt);  
  
class MyClass;  
WX_DEFINE_ARRAY(MyClass *, wxArrayOfMyClass);
```

Note that wxWindows predefines the following standard array classes: *wxArrayInt*, *wxArrayLong* and *wxArrayPtrVoid*.

4.5.9 WX_DEFINE_SORTED_ARRAY

WX_DEFINE_SORTED_ARRAY(*T*, *name*)

This macro defines a new sorted array class named *name* and containing the elements of type *T*. Example:

```
WX_DEFINE_SORTED_ARRAY(int, wxArrayInt);

class MyClass;
WX_DEFINE_SORTED_ARRAY(MyClass *, wxArrayOfMyClass);

You will have to initialize the objects of this class by passing a comparison function to
the array object constructor like this:
int CompareInts(int n1, int n2)
{
    return n1 - n2;
}

wxArrayInt sorted(CompareInts);

int CompareMyClassObjects(MyClass *item1, MyClass *item2)
{
    // sort the items by their address...
    return Stricmp(item1->GetAddress(), item2->GetAddress());
}

wxArrayOfMyClass another(CompareMyClassObjects);
```

4.5.10 WX_DECLARE_OBJARRAY**WX_DECLARE_OBJARRAY(*T*, *name*)**

This macro declares a new object array class named *name* and containing the elements of type *T*. Example:

```
class MyClass;
WX_DECLARE_OBJARRAY(MyClass, wxArrayOfMyClass); // note: not "MyClass *"!
```

You must use *WX_DEFINE_OBJARRAY()* (p. 20) macro to define the array class - otherwise you would get link errors.

4.5.11 WX_DEFINE_OBJARRAY**WX_DEFINE_OBJARRAY(*name*)**

This macro defines the methods of the array class *name* not defined by the *WX_DECLARE_OBJARRAY()* (p. 20) macro. You must include the file `<wx/arrimpl.cpp>` before using this macro and you must have the full declaration of the class of array elements in scope! If you forget to do the first, the error will be caught by the compiler, but, unfortunately, many compilers will not give any warnings if you forget to do the second - but the objects of the class will not be copied correctly and their real destructor will not be called.

Example of usage:

```
// first declare the class!
class MyClass
{
public:
    MyClass(const MyClass&);

    ...

    virtual ~MyClass();
};

#include <wx/arrimpl.cpp>
WX_DEFINE_OBJARRAY(wxArrayOfMyClass);
```

4.5.12 WX_CLEAR_ARRAY

void WX_CLEAR_ARRAY(wxArray& array)

This macro may be used to delete all elements of the array before emptying it. It can not be used with wxObjArrays - but they will delete their elements anyhow when you call Empty().

4.5.13 Default constructors

wxArray()

wxObjArray()

Default constructor initializes an empty array object.

wxSortedArray(int (*)(T first, T second)compareFunction)

There is no default constructor for wxSortedArray classes - you must initialize it with a function to use for item comparison. It is a function which is passed two arguments of type *T* where *T* is the array element type and which should return a negative, zero or positive value according to whether the first element passed to it is less than, equal to or greater than the second one.

4.5.14 wxArray copy constructor and assignment operator

wxArray(const wxArray& array)

wxSortedArray(const wxSortedArray& array)

wxObjArray(const wxObjArray& array)

wxArray& operator=(const wxArray& array)

wxSortedArray& operator=(const wxSortedArray& array)

wxObjArray& operator=(const wxObjArray& array)

The copy constructors and assignment operators perform a shallow array copy (i.e. they don't copy the objects pointed to even if the source array contains the items of pointer type) for wxArray and wxSortedArray and a deep copy (i.e. the array element are copied too) for wxObjArray.

4.5.15 wxArray::~~wxArray

~wxArray()

~wxSortedArray()

~wxObjArray()

The wxObjArray destructor deletes all the items owned by the array. This is not done by wxArray and wxSortedArray versions - you may use *WX_CLEAR_ARRAY* (p. 21) macro for this.

4.5.16 wxArray::Add

void Add(T item)

void Add(T *item)

void Add(T &item)

Appends a new element to the array (where *T* is the type of the array elements.)

The first version is used with wxArray and wxSortedArray. The second and the third are used with wxObjArray. There is an important difference between them: if you give a pointer to the array, it will take ownership of it, i.e. will delete it when the item is deleted from the array. If you give a reference to the array, however, the array will make a copy of the item and will not take ownership of the original item. Once again, it only makes sense for wxObjArrays because the other array types never take ownership of their elements.

4.5.17 wxArray::Alloc

void Alloc(size_t count)

Preallocates memory for a given number of array elements. It is worth calling when the number of items which are going to be added to the array is known in advance because it will save unneeded memory reallocation. If the array already has enough memory for the given number of items, nothing happens.

4.5.18 wxArray::Clear

void Clear()

This function does the same as *Empty()* (p. 23) and additionally frees the memory allocated to the array.

4.5.19 wxArray::Count

size_t Count() const

Same as *GetCount()* (p. 23). This function is deprecated - it exists only for compatibility.

4.5.20 wxObjArray::Detach

T * Detach(size_t index)

Removes the element from the array, but, unlike,

Remove() (p. 25) doesn't delete it. The function returns the pointer to the removed element.

4.5.21 wxArray::Empty

void Empty()

Empties the array. For wxObjArray classes, this destroys all of the array elements. For wxArray and wxSortedArray this does nothing except marking the array of being empty - this function does not free the allocated memory, use *Clear()* (p. 23) for this.

4.5.22 wxArray::GetCount

size_t GetCount() const

Return the number of items in the array.

4.5.23 wxArray::Index

int Index(T& item, bool searchFromEnd = FALSE)

int Index(T& item)

The first version of the function is for wxArray and wxObjArray, the second is for

`wxSortedArray` only.

Searches the element in the array, starting from either beginning or the end depending on the value of *searchFromEnd* parameter. `wxNOT_FOUND` is returned if the element is not found, otherwise the index of the element is returned.

Linear search is used for the `wxArray` and `wxObjArray` classes but binary search in the sorted array is used for `wxSortedArray` (this is why *searchFromEnd* parameter doesn't make sense for it).

4.5.24 `wxArray::Insert`

`void Insert(T item, size_t n)`

`void Insert(T *item, size_t n)`

`void Insert(T &item, size_t n)`

Insert a new item into the array before the item *n* - thus, *Insert(something, 0u)* will insert an item in such way that it will become the first array element.

Please see *Add()* (p. 22) for explanation of the differences between the overloaded versions of this function.

4.5.25 `wxArray::IsEmpty`

`bool IsEmpty() const`

Returns `TRUE` if the array is empty, `FALSE` otherwise.

4.5.26 `wxArray::Item`

`T& Item(size_t index) const`

Returns the item at the given position in the array. If *index* is out of bounds, an assert failure is raised in the debug builds but nothing special is done in the release build.

The returned value is of type "reference to the array element type" for all of the array classes.

4.5.27 `wxArray::Last`

`T& Last() const`

Returns the last element in the array, i.e. is the same as `Item(GetCount() - 1)`. An assert failure is raised in the debug mode if the array is empty.

The returned value is of type "reference to the array element type" for all of the array classes.

4.5.28 wxArray::Remove

Remove(size_t *index*)

Remove(T *item*)

Removes the element from the array either by index or by value. When an element is removed from wxObjArray it is deleted by the array - use *Detach()* (p. 23) if you don't want this to happen. On the other hand, when an object is removed from a wxArray nothing happens - you should delete it manually if required:

```
T *item = array[n];
delete item;
array.Remove(n)
```

See also *WX_CLEAR_ARRAY* (p. 21) macro which deletes all elements of a wxArray (supposed to contain pointers).

4.5.29 wxArray::Shrink

void Shrink()

Frees all memory unused by the array. If the program knows that no new items will be added to the array it may call *Shrink()* to reduce its memory usage. However, if a new item is added to the array, some extra memory will be allocated again.

4.5.30 wxArray::Sort

void Sort(CMPFUNC<T> *compareFunction*)

The notation CMPFUNC<T> should be read as if we had the following declaration:

```
template int CMPFUNC(T *first, T *second);
```

where *T* is the type of the array elements. I.e. it is a function returning *int* which is passed two arguments of type *T* *.

Sorts the array using the specified compare function: this function should return a negative, zero or positive value according to whether the first element passed to it is less than, equal to or greater than the second one.

wxSortedArray doesn't have this function because it is always sorted.

4.6 wxArrayString

`wxArrayString` is an efficient container for storing `wxString` (p. 654) objects. It has the same features as all `wxArray` (p. 15) classes, i.e. it dynamically expands when new items are added to it (so it is as easy to use as a linked list), but the access time to the elements is constant, instead of being linear in number of elements as in the case of linked lists. It is also very size efficient and doesn't take more space than a C array `wxString[]` type. `wxArrayString` uses its knowledge of internals of `wxString` class to achieve this.

This class is used in the same way as other dynamic *arrays* (p. 15), except that no `WX_DEFINE_ARRAY` declaration is needed for it. When a string is added or inserted in the array, a copy of the string is created, so the original string may be safely deleted (e.g. if it was a `char *` pointer the memory it was using can be freed immediately after this). In general, there is no need to worry about string memory deallocation when using this class - it will always free the memory it uses itself.

The references returned by *Item* (p. 29), *Last* (p. 29) or *operator[]* (p. 27) are not constant, so the array elements may be modified in place like this

```
array.Last().MakeUpper();
```

Finally, none of the methods of this class is virtual including its destructor, so this class should not be derived from.

Derived from

Although this is not true strictly speaking, this class may be considered as a specialization of `wxArray` (p. 15) class for the `wxString` member data: it is not implemented like this, but it does have all of the `wxArray` functions.

Include files

```
<wx/string.h>
```

See also

`wxArray` (p. 15), `wxString` (p. 654), *wxString overview* (p. 899)

4.6.1 `wxArrayString::wxArrayString`

`wxArrayString()`

`wxArrayString(const wxArrayString& array)`

Default and copy constructors.

4.6.2 `wxArrayString::~~wxArrayString`

~wxArrayString()

Destructor frees memory occupied by the array strings. For the performance reasons it is not virtual, so this class should not be derived from.

4.6.3 wxArrayString::operator=**wxArrayString & operator =(const wxArrayString& array)**

Assignment operator.

4.6.4 wxArrayString::operator[]**wxString& operatorp[](size_t nIndex)**

Return the array element at position *nIndex*. An assert failure will result from an attempt to access an element beyond the end of array in debug mode, but no check is done in release mode.

This is the operator version of *Item* (p. 29) method.

4.6.5 wxArrayString::Add**void Add(const wxString& str)**

Appends a new item to the array.

See also: *Insert* (p. 28)

4.6.6 wxArrayString::Alloc**void Alloc(size_t nCount)**

Preallocates enough memory to store *nCount* items. This function may be used to improve array class performance before adding a known number of items consecutively.

See also: *Dynamic array memory management* (p. 18)

4.6.7 wxArrayString::Clear**void Clear()**

Clears the array contents and frees memory.

See also: *Empty* (p. 28)

4.6.8 wxArrayString::Count

size_t Count() const

Returns the number of items in the array. This function is deprecated and is for backwards compatibility only, please use *GetCount* (p. 28) instead.

4.6.9 wxArrayString::Empty

void Empty()

Empties the array: after a call to this function *GetCount* (p. 28) will return 0. However, this function does not free the memory used by the array and so should be used when the array is going to be reused for storing other strings. Otherwise, you should use *Clear* (p. 27) to empty the array and free memory.

4.6.10 wxArrayString::GetCount

size_t GetCount() const

Returns the number of items in the array.

4.6.11 wxArrayString::Index

int Index(const char * sz, bool bCase = TRUE, bool bFromEnd = FALSE)

Search the element in the array, starting from the beginning if *bFromEnd* is FALSE or from end otherwise. If *bCase*, comparison is case sensitive (default), otherwise the case is ignored.

Returns index of the first item matched or *wxNOT_FOUND* if there is no match.

4.6.12 wxArrayString::Insert

void Insert(const wxString& str, size_t nIndex)

Insert a new element in the array before the position *nIndex*. Thus, for example, to insert the string in the beginning of the array you would write

```
Insert("foo", 0);
```

If *nIndex* is equal to *GetCount()* + 1 this function behaves as *Add* (p. 27).

4.6.13 wxArrayString::IsEmpty

IsEmpty()

Returns TRUE if the array is empty, FALSE otherwise. This function returns the same result as *GetCount() == 0* but is probably easier to read.

4.6.14 wxArrayString::Item

wxString& Item(size_t nIndex) const

Return the array element at position *nIndex*. An assert failure will result from an attempt to access an element beyond the end of array in debug mode, but no check is done in release mode.

See also *operator[]* (p. 27) for the operator version.

4.6.15 wxArrayString::Last

Last()

Returns the last element of the array. Attempt to access the last element of an empty array will result in assert failure in debug build, however no checks are done in release mode.

4.6.16 wxArrayString::Remove (by value)

void Remove(const char * sz)

Removes the first item matching this value. An assert failure is provoked by an attempt to remove an element which does not exist in debug build.

See also: *Index* (p. 28), *Remove* (p. 29)

4.6.17 wxArrayString::Remove (by index)

void Remove(size_t nIndex)

Removes the item at given position.

See also: *Remove* (p. 29)

4.6.18 wxArrayString::Shrink

void Shrink()

Releases the extra memory allocated by the array. This function is useful to minimize the array memory consumption.

See also: *Alloc* (p. 27), *Dynamic array memory management* (p. 18)

4.6.19 wxArrayString::Sort (alphabetically)**void Sort(bool reverseOrder = FALSE)**

Sorts the array in alphabetical order or in reverse alphabetical order if *reverseOrder* is TRUE.

See also: *Sort* (p. 30)

4.6.20 wxArrayString::Sort (user defined)**void Sort(CompareFunction compareFunction)**

Sorts the array using the specified *compareFunction* for item comparison. *CompareFunction* is defined as a function taking two *const wxString&* parameters and returning *int* value less than, equal to or greater than 0 if the first string is less than, equal to or greater than the second one.

Example

The following example sorts strings by their length.

```
static int CompareStringLen(const wxString& first, const wxString&
second)
{
    return first.length() - second.length();
}

...

wxArrayString array;

array.Add("one");
array.Add("two");
array.Add("three");
array.Add("four");

array.Sort(CompareStringLen);
```

See also: *Sort* (p. 30)

4.7 wxAutomationObject

The **wxAutomationObject** class represents an OLE automation object containing a

single data member, an IDispatch pointer. It contains a number of functions that make it easy to perform automation operations, and set and get properties. The class makes heavy use of the *wxVariant* (p. 783) class.

The usage of these classes is quite close to OLE automation usage in Visual Basic. The API is high-level, and the application can specify multiple properties in a single string. The following example gets the current Excel instance, and if it exists, makes the active cell bold.

```
wxAutomationObject excelObject;  
if (excelObject.GetInstance("Excel.Application"))  
    excelObject.PutProperty("ActiveCell.Font.Bold", TRUE);
```

Note that this class works under Windows only, and currently only for Visual C++.

Derived from

wxObject (p. 471)

Include files

<wx/msw/ole/automtn.h>

See also

wxVariant (p. 783)

4.7.1 wxAutomationObject::wxAutomationObject

wxAutomationObject(WXIDISPATCH* dispatchPtr = NULL)

Constructor, taking an optional IDispatch pointer which will be released when the object is deleted.

4.7.2 wxAutomationObject::~~wxAutomationObject

~wxAutomationObject()

Destructor. If the internal IDispatch pointer is non-null, it will be released.

4.7.3 wxAutomationObject::CallMethod

wxVariant CallMethod(const wxString& method, int noArgs, wxVariant args[]) const

wxVariant CallMethod(const wxString& method, ...) const

Calls an automation method for this object. The first form takes a method name, number of arguments, and an array of variants. The second form takes a method name and zero to six constant references to variants. Since the variant class has constructors for the basic data types, and C++ provides temporary objects automatically, both of the following lines are syntactically valid:

```
wxVariant res = obj.CallMethod("Sum", wxVariant(1.2), wxVariant(3.4));  
wxVariant res = obj.CallMethod("Sum", 1.2, 3.4);
```

Note that *method* can contain dot-separated property names, to save the application needing to call `GetProperty` several times using several temporary objects. For example:

```
object.CallMethod("ActiveCell.Font.ShowDialog", "My caption");
```

4.7.4 wxAutomationObject::CreateInstance**bool CreateInstance(const wxString& classId) const**

Creates a new object based on the class id, returning TRUE if the object was successfully created, or FALSE if not.

4.7.5 wxAutomationObject::GetDispatchPtr**IDispatch* GetDispatchPtr() const**

Gets the IDispatch pointer.

4.7.6 wxAutomationObject::GetInstance**bool GetInstance(const wxString& classId) const**

Retrieves the current object associated with a class id, and attaches the IDispatch pointer to this object. Returns TRUE if a pointer was successfully retrieved, FALSE otherwise.

Note that this cannot cope with two instances of a given OLE object being active simultaneously, such as two copies of Excel running. Which object is referenced cannot currently be specified.

4.7.7 wxAutomationObject::GetObject

bool GetObject(wxAutomationObject&obj const wxString& property, int noArgs = 0, wxVariant args[] = NULL) const

Retrieves a property from this object, assumed to be a dispatch pointer, and initialises *obj* with it. To avoid having to deal with IDispatch pointers directly, use this function in preference to *wxAutomationObject::GetProperty* (p. 33) when retrieving objects from other objects.

Note that an IDispatch pointer is stored as a void* pointer in wxVariant objects.

[See also](#)

wxAutomationObject::GetProperty (p. 33)

4.7.8 wxAutomationObject::GetProperty

wxVariant GetProperty(const wxString& property, int noArgs, wxVariant args[]) const

wxVariant GetProperty(const wxString& property, ...) const

Gets a property value from this object. The first form takes a property name, number of arguments, and an array of variants. The second form takes a property name and zero to six constant references to variants. Since the variant class has constructors for the basic data types, and C++ provides temporary objects automatically, both of the following lines are syntactically valid:

```
wxVariant res = obj.GetProperty("Range", wxVariant("A1"));  
wxVariant res = obj.GetProperty("Range", "A1");
```

Note that *property* can contain dot-separated property names, to save the application needing to call *GetProperty* several times using several temporary objects.

4.7.9 wxAutomationObject::Invoke

bool Invoke(const wxString& member, int action, wxVariant& retValue, int noArgs, wxVariant args[], const wxVariant* ptrArgs[] = 0) const

This function is a low-level implementation that allows access to the IDispatch *Invoke* function. It is not meant to be called directly by the application, but is used by other convenience functions.

[Parameters](#)

member

The member function or property name.

action

Bitlist: may contain DISPATCH_PROPERTYPUT, DISPATCH_PROPERTYPUTREF, DISPATCH_METHOD.

retValue

Return value (ignored if there is no return value)

.

noArgs

Number of arguments in *args* or *ptrArgs*.

args

If non-null, contains an array of variants.

ptrArgs

If non-null, contains an array of constant pointers to variants.

Return value

TRUE if the operation was successful, FALSE otherwise.

Remarks

Two types of argument array are provided, so that when possible pointers are used for efficiency.

4.7.10 wxAutomationObject::PutProperty

bool PutProperty(const wxString& *property*, int *noArgs*, wxVariant *args*[]) const

bool PutProperty(const wxString& *property*, ...)

Puts a property value into this object. The first form takes a property name, number of arguments, and an array of variants. The second form takes a property name and zero to six constant references to variants. Since the variant class has constructors for the basic data types, and C++ provides temporary objects automatically, both of the following lines are syntactically valid:

```
obj.PutProperty("Value", wxVariant(23));  
obj.PutProperty("Value", 23);
```

Note that *property* can contain dot-separated property names, to save the application needing to call GetProperty several times using several temporary objects.

4.7.11 wxAutomationObject::SetDispatchPtr

void SetDispatchPtr(WXIDISPATCH* dispatchPtr)

Sets the IDispatch pointer. This function does not check if there is already an IDispatch pointer.

You may need to cast from IDispatch* to WXIDISPATCH* when calling this function.

4.8 wxBusyCursor

This class makes it easy to tell your user that the program is temporarily busy. Just create a wxBusyCursor object on the stack, and within the current scope, the hourglass will be shown.

For example:

```
wxBusyCursor wait;

for (int i = 0; i < 100000; i++)
    DoACalculation();
```

It works by calling *wxBeginBusyCursor* (p. 862) in the constructor, and *wxEndBusyCursor* (p. 865) in the destructor.

Derived from

None

Include files

<wx/utils.h>

See also

wxBeginBusyCursor (p. 862), *wxEndBusyCursor* (p. 865)

4.8.1 wxBusyCursor::wxBusyCursor

wxBusyCursor(wxCursor* cursor = wxHOURLASS_CURSOR)

Constructs a busy cursor object, calling *wxBeginBusyCursor* (p. 862).

4.8.2 wxBusyCursor::~~wxBusyCursor

~wxBusyCursor()

Destroys the busy cursor object, calling *wxEndBusyCursor* (p. 865).

4.9 wxButton

A button is a control that contains a text string, and is one of the commonest elements of a GUI. It may be placed on a *dialog box* (p. 178) or *panel* (p. 488), or indeed almost any other window.

Derived from

wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/button.h>

Window styles

There are no special styles for *wxButton*.

See also *window styles overview* (p. 959).

Event handling

EVT_BUTTON(id, func)	Process a wxEVT_COMMAND_BUTTON_CLICKED event, when the button is clicked.
-----------------------------	----------------------------------------------------------------------------------------

See also

wxBitmapButton (p. 54)

4.9.1 wxButton::wxButton

wxButton()

Default constructor.

wxButton(wxWindow* parent, wxWindowID id, const wxString& label, const wxPoint& pos, const wxSize& size = wxDefaultSize, long style = 0, const wxValidator& validator, const wxString& name = "button")

Constructor, creating and showing a button.

Parameters

parent

Parent window. Must not be NULL.

id

Button identifier. A value of -1 indicates a default value.

label

Text to be displayed on the button.

pos

Button position.

size

Button size. If the default size (-1, -1) is specified then the button is sized appropriately for the text.

style

Window style. See *wxButton* (p. 36).

validator

Window validator.

name

Window name.

See also

wxButton::Create (p. 37), *wxValidator* (p. 781)

4.9.2 *wxButton::~~wxButton*

~wxButton()

Destructor, destroying the button.

4.9.3 *wxButton::Create*

bool Create(wxWindow* parent, wxWindowID id, const wxString& label, const wxPoint& pos, const wxSize& size = wxDefaultSize, long style = 0, const wxValidator& validator, const wxString& name = "button")

Button creation function for two-step creation. For more details, see *wxButton::wxButton* (p. 36).

4.9.4 wxButton::GetLabel

wxString GetLabel() const

Returns the string label for the button.

Return value

The button's label.

See also

wxButton::SetLabel (p. 38)

4.9.5 wxButton::SetDefault

void SetDefault()

This sets the button to be the default item for the panel or dialog box.

Remarks

Under Windows, only dialog box buttons respond to this function. As normal under Windows and Motif, pressing return causes the default button to be depressed when the return key is pressed. See also *wxWindow::SetFocus* (p. 835) which sets the keyboard focus for windows and text panel items, and *wxWindow::GetDefaultItem* (p. 808).

Note that under Motif, calling this function immediately after creation of a button and before the creation of other buttons will cause misalignment of the row of buttons, since default buttons are larger. To get around this, call *SetDefault* after you have created a row of buttons: wxWindows will then set the size of all buttons currently on the panel to the same size.

4.9.6 wxButton::SetLabel

void SetLabel(const wxString& label)

Sets the string label for the button.

Parameters

label

The label to set.

See also

wxButton::GetLabel (p. 38)

4.10 wxBitmap

This class encapsulates the concept of a platform-dependent bitmap, either monochrome or colour.

Derived from

wxGDIObject (p. 295)

wxObject (p. 471)

Include files

<wx/bitmap.h>

Predefined objects

Objects:

wxNullBitmap

See also

wxBitmap overview (p. 908), *supported bitmap file formats* (p. 908), *wxDC::Blit* (p. 152), *wxIcon* (p. 318), *wxCursor* (p. 128), *wxBitmap* (p. 39), *wxMemoryDC* (p. 415)

4.10.1 wxBitmap::wxBitmap

wxBitmap()

Default constructor.

wxBitmap(const wxBitmap& *bitmap*)

Copy constructor.

wxBitmap(void* *data*, int *type*, int *width*, int *height*, int *depth* = -1)

Creates a bitmap from the given data, which can be of arbitrary type.

**wxBitmap(const char *bits*[], int *width*, int *height*
int *depth* = 1)**

Creates a bitmap from an array of bits.

wxBitmap(int *width*, int *height*, int *depth* = -1)

Creates a new bitmap.

wxBitmap(const char bits)**

Creates a bitmap from XPM data.

wxBitmap(const wxString& name, long type)

Loads a bitmap from a file or resource.

Parameters

bits

Specifies an array of pixel values.

width

Specifies the width of the bitmap.

height

Specifies the height of the bitmap.

depth

Specifies the depth of the bitmap. If this is omitted, the display depth of the screen is used.

name

This can refer to a resource name under MS Windows, or a filename under MS Windows and X. Its meaning is determined by the *type* parameter.

type

May be one of the following:

wxBITMAP_TYPE_BMP	Load a Windows bitmap file.
wxBITMAP_TYPE_BMP_RESOURCE	Load a Windows bitmap from the resource database.
wxBITMAP_TYPE_GIF	Load a GIF bitmap file.
wxBITMAP_TYPE_XBM	Load an X bitmap file.
wxBITMAP_TYPE_XPM	Load an XPM bitmap file.
wxBITMAP_TYPE_RESOURCE	Load a Windows resource name.

The validity of these flags depends on the platform and wxWindows configuration. If all possible wxWindows settings are used, the Windows platform supports BMP file, BMP resource, XPM data, and XPM. Under wxGTK, the available formats are BMP file, XPM data, XPM file, and PNG file. Under wxMotif, the available formats are XBM data, XBM file, XPM data, XPM file.

Remarks

The first form constructs a bitmap object with no data; an assignment or another member function such as `Create` or `LoadFile` must be called subsequently.

The second and third forms provide copy constructors. Note that these do not copy the bitmap data, but instead a pointer to the data, keeping a reference count. They are

therefore very efficient operations.

The fourth form constructs a bitmap from data whose type and value depends on the value of the *type* argument.

The fifth form constructs a (usually monochrome) bitmap from an array of pixel values, under both X and Windows.

The sixth form constructs a new bitmap.

The seventh form constructs a bitmap from pixmap (XPM) data, if wxWindows has been configured to incorporate this feature.

To use this constructor, you must first include an XPM file. For example, assuming that the file `mybitmap.xpm` contains an XPM array of character pointers called `mybitmap`:

```
#include "mybitmap.xpm"

...

wxBitmap *bitmap = new wxBitmap(mybitmap);
```

The eighth form constructs a bitmap from a file or resource. *name* can refer to a resource name under MS Windows, or a filename under MS Windows and X.

Under Windows, *type* defaults to `wxBITMAP_TYPE_BMP_RESOURCE`. Under X, *type* defaults to `wxBITMAP_TYPE_XPM`.

See also

`wxBitmap::LoadFile` (p. 45)

wxPython note:

Constructors supported by wxPython are:

wxBitmap(name, flag)	Loads a bitmap from a file
wxNoRefBitmap(name, flag)	This one won't own the reference, so Python won't call the destructor, this is good for toolbars and such where the parent will manage the bitmap.
wxEmptyBitmap(width, height, depth=-1)	Creates an empty bitmap with the given specifications

4.10.2 wxBitmap::~~wxBitmap

~wxBitmap()

Destroys the `wxBitmap` object and possibly the underlying bitmap data. Because reference counting is used, the bitmap may not actually be destroyed at this point - only

when the reference count is zero will the data be deleted.

If the application omits to delete the bitmap explicitly, the bitmap will be destroyed automatically by `wxWindows` when the application exits.

Do not delete a bitmap that is selected into a memory device context.

4.10.3 `wxBitmap::AddHandler`

static void AddHandler(`wxBitmapHandler*` handler)

Adds a handler to the end of the static list of format handlers.

handler

A new bitmap format handler object. There is usually only one instance of a given handler class in an application session.

[See also](#)

`wxBitmapHandler` (p. 50)

4.10.4 `wxBitmap::CleanUpHandlers`

static void CleanUpHandlers()

Deletes all bitmap handlers.

This function is called by `wxWindows` on exit.

4.10.5 `wxBitmap::Create`

virtual bool Create(`int width`, `int height`, `int depth = -1`)

Creates a fresh bitmap. If the final argument is omitted, the display depth of the screen is used.

virtual bool Create(`void* data`, `int type`, `int width`, `int height`, `int depth = -1`)

Creates a bitmap from the given data, which can be of arbitrary type.

[Parameters](#)

width

The width of the bitmap in pixels.

height

The height of the bitmap in pixels.

depth

The depth of the bitmap in pixels. If this is -1, the screen depth is used.

data

Data whose type depends on the value of *type*.

type

A bitmap type identifier - see *wxBitmap::wxBitmap* (p. 39) for a list of possible values.

Return value

TRUE if the call succeeded, FALSE otherwise.

Remarks

The first form works on all platforms. The portability of the second form depends on the type of data.

See also

wxBitmap::wxBitmap (p. 39)

4.10.6 wxBitmap::FindHandler

static wxBitmapHandler* FindHandler(const wxString& name)

Finds the handler with the given name.

static wxBitmapHandler* FindHandler(const wxString& extension, long bitmapType)

Finds the handler associated with the given extension and type.

static wxBitmapHandler* FindHandler(long bitmapType)

Finds the handler associated with the given bitmap type.

name

The handler name.

extension

The file extension, such as "bmp".

bitmapType

The bitmap type, such as wxBITMAP_TYPE_BMP.

Return value

A pointer to the handler if found, NULL otherwise.

See also

wxBitmapHandler (p. 50)

4.10.7 wxBitmap::GetDepth

int GetDepth() const

Gets the colour depth of the bitmap. A value of 1 indicates a monochrome bitmap.

4.10.8 wxBitmap::GetHandlers

static wxList& GetHandlers()

Returns the static list of bitmap format handlers.

See also

wxBitmapHandler (p. 50)

4.10.9 wxBitmap::GetHeight

int GetHeight() const

Gets the height of the bitmap in pixels.

4.10.10 wxBitmap::GetPalette

wxPalette* GetPalette() const

Gets the associated palette (if any) which may have been loaded from a file or set for the bitmap.

See also

wxPalette (p. 484)

4.10.11 wxBitmap::GetMask

wxMask* GetMask() const

Gets the associated mask (if any) which may have been loaded from a file or set for the bitmap.

See also

wxBitmap::SetMask (p. 48), *wxMask* (p. 403)

4.10.12 wxBitmap::GetWidth

int GetWidth() const

Gets the width of the bitmap in pixels.

See also

wxBitmap::GetHeight (p. 44)

4.10.13 wxBitmap::InitStandardHandlers

static void InitStandardHandlers()

Adds the standard bitmap format handlers, which, depending on wxWindows configuration, can be handlers for Windows bitmap, Windows bitmap resource, and XPM.

This function is called by wxWindows on startup.

See also

wxBitmapHandler (p. 50)

4.10.14 wxBitmap::InsertHandler

static void InsertHandler(wxBitmapHandler* handler)

Adds a handler at the start of the static list of format handlers.

handler

A new bitmap format handler object. There is usually only one instance of a given handler class in an application session.

See also

wxBitmapHandler (p. 50)

4.10.15 wxBitmap::LoadFile

bool LoadFile(const wxString& name, long type)

Loads a bitmap from a file or resource.

Parameters

name

Either a filename or a Windows resource name. The meaning of *name* is determined by the *type* parameter.

type

One of the following values:

wxBITMAP_TYPE_BMP	Load a Windows bitmap file.
wxBITMAP_TYPE_BMP_RESOURCE	Load a Windows bitmap from the resource database.
wxBITMAP_TYPE_GIF	Load a GIF bitmap file.
wxBITMAP_TYPE_XBM	Load an X bitmap file.
wxBITMAP_TYPE_XPM	Load an XPM bitmap file.

The validity of these flags depends on the platform and wxWindows configuration.

Return value

TRUE if the operation succeeded, FALSE otherwise.

Remarks

A palette may be associated with the bitmap if one exists (especially for colour Windows bitmaps), and if the code supports it. You can check if one has been created by using the *GetPalette* (p. 44) member.

See also

wxBitmap::SaveFile (p. 47)

4.10.16 **wxBitmap::Ok**

bool Ok() const

Returns TRUE if bitmap data is present.

4.10.17 **wxBitmap::RemoveHandler**

static bool RemoveHandler(const wxString& name)

Finds the handler with the given name, and removes it. The handler is not deleted.

name

The handler name.

Return value

TRUE if the handler was found and removed, FALSE otherwise.

See also

wxBitmapHandler (p. 50)

4.10.18 wxBitmap::SaveFile

bool SaveFile(const wxString& *name*, int *type*, wxPalette* *palette* = NULL)

Saves a bitmap in the named file.

Parameters

name

A filename. The meaning of *name* is determined by the *type* parameter.

type

One of the following values:

wxBITMAP_TYPE_BMP	Save a Windows bitmap file.
wxBITMAP_TYPE_GIF	Save a GIF bitmap file.
wxBITMAP_TYPE_XBM	Save an X bitmap file.
wxBITMAP_TYPE_XPM	Save an XPM bitmap file.

The validity of these flags depends on the platform and wxWindows configuration.

palette

An optional palette used for saving the bitmap.

Return value

TRUE if the operation succeeded, FALSE otherwise.

Remarks

Depending on how wxWindows has been configured, not all formats may be available.

See also

wxBitmap::LoadFile (p. 45)

4.10.19 wxBitmap::SetDepth

void SetDepth(int *depth*)

Sets the depth member (does not affect the bitmap data).

Parameters

depth
Bitmap depth.

4.10.20 **wxBitmap::SetHeight**

void SetHeight(int *height*)

Sets the height member (does not affect the bitmap data).

Parameters

height
Bitmap height in pixels.

4.10.21 **wxBitmap::SetMask**

void SetMask(wxMask* *mask*)

Sets the mask for this bitmap.

Remarks

The bitmap object owns the mask once this has been called.

See also

wxBitmap::GetMask (p. 44), *wxMask* (p. 403)

4.10.22 **wxBitmap::SetOk**

void SetOk(int *isOk*)

Sets the validity member (does not affect the bitmap data).

Parameters

isOk
Validity flag.

4.10.23 **wxBitmap::SetPalette**

void SetPalette(wxPalette* *palette*)

Sets the associated palette: it will be deleted in the wxBitmap destructor, so if you do not

wish it to be deleted automatically, reset the palette to NULL before the bitmap is deleted.

Parameters

palette

The palette to set.

Remarks

The bitmap object owns the palette once this has been called.

See also

wxPalette (p. 484)

4.10.24 **wxBitmap::SetWidth**

void SetWidth(int *width*)

Sets the width member (does not affect the bitmap data).

Parameters

width

Bitmap width in pixels.

4.10.25 **wxBitmap::operator =**

wxBitmap& operator =(const wxBitmap& *bitmap*)

Assignment operator. This operator does not copy any data, but instead passes a pointer to the data in *bitmap* and increments a reference counter. It is a fast operation.

Parameters

bitmap

Bitmap to assign.

Return value

Returns 'this' object.

4.10.26 **wxBitmap::operator ==**

bool operator ==(const wxBitmap& *bitmap*)

Equality operator. This operator tests whether the internal data pointers are equal (a fast test).

Parameters

bitmap
Bitmap to compare with 'this'

Return value

Returns TRUE if the bitmaps were effectively equal, FALSE otherwise.

4.10.27 wxBitmap::operator !=

bool operator !=(const wxBitmap& *bitmap*)

Inequality operator. This operator tests whether the internal data pointers are unequal (a fast test).

Parameters

bitmap
Bitmap to compare with 'this'

Return value

Returns TRUE if the bitmaps were unequal, FALSE otherwise.

4.11 wxBitmapHandler

Overview (p. 908)

This is the base class for implementing bitmap file loading/saving, and bitmap creation from data. It is used within wxBitmap and is not normally seen by the application.

If you wish to extend the capabilities of wxBitmap, derive a class from wxBitmapHandler and add the handler using *wxBitmap::AddHandler* (p. 42) in your application initialisation.

Derived from

wxObject (p. 471)

Include files

<wx/bitmap.h>

See also

wxBitmap (p. 39), *wxIcon* (p. 318), *wxCursor* (p. 128)

4.11.1 wxBitmapHandler::wxBitmapHandler

wxBitmapHandler()

Default constructor. In your own default constructor, initialise the members `m_name`, `m_extension` and `m_type`.

4.11.2 wxBitmapHandler::~~wxBitmapHandler

~wxBitmapHandler()

Destroys the `wxBitmapHandler` object.

4.11.3 wxBitmapHandler::Create

virtual bool Create(*wxBitmap* bitmap*, *void* data*, *int type*, *int width*, *int height*, *int depth = -1*)

Creates a bitmap from the given data, which can be of arbitrary type. The `wxBitmap` object *bitmap* is manipulated by this function.

Parameters

bitmap

The `wxBitmap` object.

width

The width of the bitmap in pixels.

height

The height of the bitmap in pixels.

depth

The depth of the bitmap in pixels. If this is -1, the screen depth is used.

data

Data whose type depends on the value of *type*.

type

A bitmap type identifier - see `wxBitmapHandler::wxBitmapHandler` (p. 39) for a list of possible values.

Return value

TRUE if the call succeeded, FALSE otherwise (the default).

4.11.4 wxBitmapHandler::GetName

wxString GetName() const

Gets the name of this handler.

4.11.5 wxBitmapHandler::GetExtension

wxString GetExtension() const

Gets the file extension associated with this handler.

4.11.6 wxBitmapHandler::GetType

long GetType() const

Gets the bitmap type associated with this handler.

4.11.7 wxBitmapHandler::LoadFile

bool LoadFile(wxBitmap* *bitmap*, const wxString& *name*, long *type*)

Loads a bitmap from a file or resource, putting the resulting data into *bitmap*.

Parameters

bitmap

The bitmap object which is to be affected by this operation.

name

Either a filename or a Windows resource name. The meaning of *name* is determined by the *type* parameter.

type

See *wxBitmap::wxBitmap* (p. 39) for values this can take.

Return value

TRUE if the operation succeeded, FALSE otherwise.

See also

wxBitmap::LoadFile (p. 45)

wxBitmap::SaveFile (p. 47)

wxBitmapHandler::SaveFile (p. 53)

4.11.8 wxBitmapHandler::SaveFile

bool SaveFile(wxBitmap* *bitmap*, const wxString& *name*, int *type*, wxPalette* *palette* = NULL)

Saves a bitmap in the named file.

Parameters

bitmap

The bitmap object which is to be affected by this operation.

name

A filename. The meaning of *name* is determined by the *type* parameter.

type

See *wxBitmap::wxBitmap* (p. 39) for values this can take.

palette

An optional palette used for saving the bitmap.

Return value

TRUE if the operation succeeded, FALSE otherwise.

See also

wxBitmap::LoadFile (p. 45)

wxBitmap::SaveFile (p. 47)

wxBitmapHandler::LoadFile (p. 52)

4.11.9 wxBitmapHandler::SetName

void SetName(const wxString& *name*)

Sets the handler name.

Parameters

name

Handler name.

4.11.10 wxBitmapHandler::SetExtension

void SetExtension(const wxString& *extension*)

Sets the handler extension.

Parameters

extension

Handler extension.

4.11.11 wxBitmapHandler::SetType

void SetType(long type)

Sets the handler type.

Parameters

name

Handler type.

4.12 wxBitmapButton

A bitmap button is a control that contains a bitmap. It may be placed on a *dialog box* (p. 178) or *panel* (p. 488), or indeed almost any other window.

Derived from

wxButton (p. 36)

wxControl (p. 125)

wxWindow (p. 798)

wxEvtHandler (p. 224)

wxObject (p. 471)

Include files

<wx/bmpbuttn.h>

Remarks

A bitmap button can be supplied with a single bitmap, and wxWindows will draw all button states using this bitmap. If the application needs more control, additional bitmaps for the selected state, unpressed focussed state, and greyed-out state may be supplied.

Window styles

wxBU_AUTODRAW

If this is specified, the button will be drawn automatically using the label bitmap only, providing a 3D-look border. If this style is not specified, the button will be drawn without borders and using all provided bitmaps.

See also *window styles overview* (p. 959).

Event handling

EVT_BUTTON(id, func)

Process a `wxEVT_COMMAND_BUTTON_CLICKED` event, when the button is clicked.

See also

wxButton (p. 36)

4.12.1 `wxBitmapButton::wxBitmapButton`

`wxBitmapButton()`

Default constructor.

`wxBitmapButton(wxWindow* parent, wxWindowID id, const wxBitmap& bitmap, const wxPoint& pos, const wxSize& size = wxDefaultSize, long style = wxBU_AUTODRAW, const wxValidator& validator, const wxString& name = "button")`

Constructor, creating and showing a button.

Parameters

parent

Parent window. Must not be NULL.

id

Button identifier. A value of -1 indicates a default value.

bitmap

Bitmap to be displayed.

pos

Button position.

size

Button size. If the default size (-1, -1) is specified then the button is sized appropriately for the bitmap.

style

Window style. See *wxBitmapButton* (p. 54).

validator

Window validator.

name

Window name.

Remarks

The *bitmap* parameter is normally the only bitmap you need to provide, and `wxWindows` will draw the button correctly in its different states. If you want more control, call any of the functions `wxBitmapButton::SetBitmapSelected` (p. 59), `wxBitmapButton::SetBitmapFocus` (p. 58), `wxBitmapButton::SetBitmapDisabled` (p. 57).

Note that the bitmap passed is smaller than the actual button created.

See also

`wxBitmapButton::Create` (p. 56), `wxValidator` (p. 781)

4.12.2 `wxBitmapButton::~~wxBitmapButton`

`~wxBitmapButton()`

Destructor, destroying the button.

4.12.3 `wxBitmapButton::Create`

`bool Create(wxWindow* parent, wxWindowID id, const wxBitmap& bitmap, const wxPoint& pos, const wxSize& size = wxDefaultSize, long style = 0, const wxValidator& validator, const wxString& name = "button")`

Button creation function for two-step creation. For more details, see `wxBitmapButton::wxBitmapButton` (p. 55).

4.12.4 `wxBitmapButton::GetBitmapDisabled`

`wxBitmap& GetBitmapLabel() const`

Returns the bitmap for the disabled state.

Return value

A reference to the disabled state bitmap.

See also

`wxBitmapButton::SetBitmapDisabled` (p. 57)

4.12.5 `wxBitmapButton::GetBitmapFocus`

wxBitmap& GetBitmapFocus() const

Returns the bitmap for the focussed state.

Return value

A reference to the focussed state bitmap.

See also

wxBitmapButton::SetBitmapFocus (p. 58)

4.12.6 wxBitmapButton::GetBitmapLabel**wxBitmap& GetBitmapLabel() const**

Returns the label bitmap (the one passed to the constructor).

Return value

A reference to the button's label bitmap.

See also

wxBitmapButton::SetBitmapLabel (p. 58)

4.12.7 wxBitmapButton::GetBitmapSelected**wxBitmap& GetBitmapSelected() const**

Returns the bitmap for the selected state.

Return value

A reference to the selected state bitmap.

See also

wxBitmapButton::SetBitmapSelected (p. 59)

4.12.8 wxBitmapButton::SetBitmapDisabled**void SetBitmapDisabled(const wxBitmap& *bitmap*)**

Sets the bitmap for the disabled button appearance.

Parameters

bitmap
The bitmap to set.

See also

wxBitmapButton::GetBitmapDisabled (p. 56), *wxBitmapButton::SetBitmapLabel* (p. 58), *wxBitmapButton::SetBitmapSelected* (p. 59), *wxBitmapButton::SetBitmapFocus* (p. 58)

4.12.9 wxBitmapButton::SetBitmapFocus

void SetBitmapFocus(const wxBitmap& *bitmap*)

Sets the bitmap for the button appearance when it has the keyboard focus.

Parameters

bitmap
The bitmap to set.

See also

wxBitmapButton::GetBitmapFocus (p. 56), *wxBitmapButton::SetBitmapLabel* (p. 58), *wxBitmapButton::SetBitmapSelected* (p. 59), *wxBitmapButton::SetBitmapDisabled* (p. 57)

4.12.10 wxBitmapButton::SetBitmapLabel

void SetBitmapLabel(const wxBitmap& *bitmap*)

Sets the bitmap label for the button.

Parameters

bitmap
The bitmap label to set.

Remarks

This is the bitmap used for the unselected state, and for all other states if no other bitmaps are provided.

See also

wxBitmapButton::GetBitmapLabel (p. 57)

4.12.11 wxBitmapButton::SetBitmapSelected

void SetBitmapSelected(const wxBitmap& *bitmap*)

Sets the bitmap for the selected (depressed) button appearance.

Parameters

bitmap

The bitmap to set.

See also

wxBitmapButton::GetBitmapSelected (p. 57), *wxBitmapButton::SetBitmapLabel* (p. 58), *wxBitmapButton::SetBitmapFocus* (p. 58), *wxBitmapButton::SetBitmapDisabled* (p. 57)

4.13 wxBitmapDataObject

wxBitmapDataObject is a specialization of *wxDataObject* for bitmap data. It can be used without change to paste data into the *wxClipboard* (p. 82) or a *wxDropSource* (p. 216). A user may wish to derive a new class from this class for providing a bitmap on-demand in order to minimize memory consumption when offering data in several formats, such as a bitmap and GIF.

In order to offer bitmap data on-demand *GetSize* (p. 60) and *WriteData* (p. 61) will have to be overridden.

Derived from

wxDataObject (p. 138)

Include files

<wx/dataobj.h>

See also

wxDataObject (p. 138)

4.13.1 wxBitmapDataObject::wxBitmapDataObject

wxBitmapDataObject()

Default constructor. Call *SetBitmap* (p. 60) later or override *WriteData* (p. 61) and *GetSize* (p. 60) for providing data on-demand.

wxBitmapDataObject(const wxBitmap& *bitmap*)

Constructor, passing a bitmap.

4.13.2 wxBitmapDataObject::GetSize

virtual size_t GetSize() const

Returns the data size. By default, returns the size of the bitmap data set in the constructor or using *SetBitmap* (p. 60). This can be overridden to provide size data on-demand. Note that you'd have to call the inherited *GetSize* method as this is the only way to get to know the transfer size of the bitmap in a platform dependent way - a bitmap has different size under GTK and Windows. In practice, this would look like this:

```
size_t MyBitmapDataObject::GetSize()
{
    // Get bitmap from global container. This container
    // should be able to "produce" data in all formats
    // offered by the application but store it only in
    // one format to reduce memory consumption.

    wxBitmap my_bitmap = my_global_container->GetBitmap();

    // temporarily set bitmap

    SetBitmap( my_bitmap );

    size_t ret = wxBitmapDataObject::GetSize();

    // unset bitmap again

    SetBitmap( wxNullBitmap );

    return ret;
}
```

TODO: Offer a nicer way to do this. Maybe by providing a platform dependent function in this class like

```
size_t GetBitmapSize( const wxBitmap &bitmap )
```

4.13.3 wxBitmapDataObject::GetBitmap

virtual wxBitmap GetBitmap() const

Returns the bitmap associated with the data object. You may wish to override this method when offering data on-demand, but this is not required by wxWindows' internals. Use this method to get data in bitmap form from the *wxClipboard* (p. 82).

4.13.4 wxBitmapDataObject::SetBitmap

virtual void SetBitmap(const wxBitmap& bitmap)

Sets the bitmap associated with the data object. This method is called internally when retrieving data from the *wxClipboard* (p. 82) and may be used to paste data to the clipboard directly (instead of on-demand).

4.13.5 *wxBitmapDataObject::WriteData*

virtual void WriteData(voiddest*) const**

Write the data owned by this class to *dest*. By default, this calls *WriteBitmap* (p. 61) with the bitmap set in the constructor or using *SetBitmap* (p. 60). This can be overridden to provide bitmap data on-demand; in this case *WriteBitmap* (p. 61) must be called from within the overriding *WriteData()* method.

4.13.6 *wxBitmapDataObject::WriteBitmap*

void WriteBitmap(const *wxBitmap*& *bitmap* voiddest*) const**

Writes the bitmap *bitmap* to *dest*. This method must be called from *WriteData* (p. 61).

4.14 *wxBrush*

A brush is a drawing tool for filling in areas. It is used for painting the background of rectangles, ellipses, etc. It has a colour and a style.

Derived from

wxGDIObject (p. 295)

wxObject (p. 471)

Include files

<wx/brush.h>

Predefined objects

Objects:

wxNullBrush

Pointers:

wxBLUE_BRUSH
wxGREEN_BRUSH
wxWHITE_BRUSH
wxBLACK_BRUSH
wxGREY_BRUSH
wxMEDIUM_GREY_BRUSH

wxLIGHT_GREY_BRUSH
wxTRANSPARENT_BRUSH
wxCYAN_BRUSH
wxRED_BRUSH

Remarks

On a monochrome display, wxWindows shows all brushes as white unless the colour is really black.

Do not initialize objects on the stack before the program commences, since other required structures may not have been set up yet. Instead, define global pointers to objects and create them in *wxApp::OnInit* (p. 12) or when required.

An application may wish to create brushes with different characteristics dynamically, and there is the consequent danger that a large number of duplicate brushes will be created. Therefore an application may wish to get a pointer to a brush by using the global list of brushes **wxTheBrushList**, and calling the member function **FindOrCreateBrush**.

wxBrush uses a reference counting system, so assignments between brushes are very cheap. You can therefore use actual wxBrush objects instead of pointers without efficiency problems. Once one wxBrush object changes its data it will create its own brush data internally so that other brushes, which previously shared the data using the reference counting, are not affected.

See also

wxBrushList (p. 67), *wxDC* (p. 151), *wxDC::SetBrush* (p. 163)

4.14.1 wxBrush::wxBrush

wxBrush()

Default constructor. The brush will be uninitialised, and *wxBrush::Ok* (p. 65) will return FALSE.

wxBrush(const wxColour& colour, int style)

Constructs a brush from a colour object and style.

wxBrush(const wxString& colourName, int style)

Constructs a brush from a colour name and style.

wxBrush(const wxBitmap& stippleBitmap)

Constructs a stippled brush using a bitmap.

wxBrush(const wxBrush& brush)

Copy constructor. This uses reference counting so is a cheap operation.

Parameters

colour

Colour object.

colourName

Colour name. The name will be looked up in the colour database.

style

One of:

wxTRANSPARENT

Transparent (no fill).

wxSOLID

Solid.

wxBDIAGONAL_HATCH

Backward diagonal hatch.

wxCROSSDIAG_HATCH

Cross-diagonal hatch.

wxFDIAGONAL_HATCH

Forward diagonal hatch.

wxCROSS_HATCH

Cross hatch.

wxHORIZONTAL_HATCH

Horizontal hatch.

wxVERTICAL_HATCH

Vertical hatch.

brush

Pointer or reference to a brush to copy.

stippleBitmap

A bitmap to use for stippling.

Remarks

If a stipple brush is created, the brush style will be set to wxSTIPPLE.

See also

wxBrushList (p. 67), *wxColour* (p. 86), *wxColourDatabase* (p. 91)

4.14.2 wxBrush::~wxBrush

void ~wxBrush()

Destructor.

Remarks

The destructor may not delete the underlying brush object of the native windowing system, since wxBrush uses a reference counting system for efficiency.

Although all remaining brushes are deleted when the application exits, the application should try to clean up all brushes itself. This is because `wxWindows` cannot know if a pointer to the brush object is stored in an application data structure, and there is a risk of double deletion.

4.14.3 `wxBrush::GetColour`

`wxColour& GetColour() const`

Returns a reference to the brush colour.

[See also](#)

`wxBrush::SetColour` (p. 65)

4.14.4 `wxBrush::GetStipple`

`wxBitmap * GetStipple() const`

Gets a pointer to the stipple bitmap. If the brush does not have a `wxSTIPPLE` style, this bitmap may be non-NULL but uninitialised (*`wxBitmap::Ok` (p. 46)* returns `FALSE`).

[See also](#)

`wxBrush::SetStipple` (p. 65)

4.14.5 `wxBrush::GetStyle`

`int GetStyle() const`

Returns the brush style, one of:

<code>wxTRANSPARENT</code>	Transparent (no fill).
<code>wxSOLID</code>	Solid.
<code>wxBDIAGONAL_HATCH</code>	Backward diagonal hatch.
<code>wxCROSSDIAG_HATCH</code>	Cross-diagonal hatch.
<code>wxFDIAGONAL_HATCH</code>	Forward diagonal hatch.
<code>wxCROSS_HATCH</code>	Cross hatch.
<code>wxHORIZONTAL_HATCH</code>	Horizontal hatch.
<code>wxVERTICAL_HATCH</code>	Vertical hatch.
<code>wxSTIPPLE</code>	Stippled using a bitmap.

[See also](#)

`wxBrush::SetStyle` (p. 66), `wxBrush::SetColour` (p. 65), `wxBrush::SetStipple` (p. 65)

4.14.6 wxBrush::Ok

bool Ok() const

Returns TRUE if the brush is initialised. It will return FALSE if the default constructor has been used (for example, the brush is a member of a class, or NULL has been assigned to it).

4.14.7 wxBrush::SetColour

void SetColour(wxColour& colour)

Sets the brush colour using a reference to a colour object.

void SetColour(const wxString& colourName)

Sets the brush colour using a colour name from the colour database.

void SetColour(const unsigned char red, const unsigned char green, const unsigned char blue)

Sets the brush colour using red, green and blue values.

[See also](#)

wxBrush::GetColour (p. 64)

4.14.8 wxBrush::SetStipple

void SetStipple(const wxBitmap& bitmap)

Sets the stipple bitmap.

Parameters

bitmap

The bitmap to use for stippling.

Remarks

The style will be set to wxSTIPPLE.

Note that there is a big difference between stippling in X and Windows. On X, the stipple is a mask between the wxBitmap and current colour. On Windows, the current colour is ignored, and the bitmap colour is used. However, for pre-defined modes like wxCROSS_HATCH, the behaviour is the same for both platforms.

See also

wxBitmap (p. 39)

4.14.9 wxBrush::SetStyle

void SetStyle(int style)

Sets the brush style.

style

One of:

wxTRANSPARENT	Transparent (no fill).
wxSOLID	Solid.
wxBDIAGONAL_HATCH	Backward diagonal hatch.
wxCROSSDIAG_HATCH	Cross-diagonal hatch.
wxFDIAGONAL_HATCH	Forward diagonal hatch.
wxCROSS_HATCH	Cross hatch.
wxHORIZONTAL_HATCH	Horizontal hatch.
wxVERTICAL_HATCH	Vertical hatch.
wxSTIPPLE	Stippled using a bitmap.

See also

wxBrush::GetStyle (p. 64)

4.14.10 wxBrush::operator =

wxBrush& operator =(const wxBrush& brush)

Assignment operator, using reference counting. Returns a reference to 'this'.

4.14.11 wxBrush::operator ==

bool operator ==(const wxBrush& brush)

Equality operator. Two brushes are equal if they contain pointers to the same underlying brush data. It does not compare each attribute, so two independently-created brushes using the same parameters will fail the test.

4.14.12 wxBrush::operator !=

bool operator !=(const wxBrush& brush)

Inequality operator. Two brushes are not equal if they contain pointers to different underlying brush data. It does not compare each attribute.

4.15 wxBrushList

A brush list is a list containing all brushes which have been created.

Derived from

wxList (p. 367)
wxObject (p. 471)

Include files

<wx/gdicmn.h>

Remarks

There is only one instance of this class: **wxTheBrushList**. Use this object to search for a previously created brush of the desired type and create it if not already found. In some windowing systems, the brush may be a scarce resource, so it can pay to reuse old resources if possible. When an application finishes, all brushes will be deleted and their resources freed, eliminating the possibility of 'memory leaks'. However, it is best not to rely on this automatic cleanup because it can lead to double deletion in some circumstances.

There are two mechanisms in recent versions of wxWindows which make the brush list less useful than it once was. Under Windows, scarce resources are cleaned up internally if they are not being used. Also, a referencing counting mechanism applied to all GDI objects means that some sharing of underlying resources is possible. You don't have to keep track of pointers, working out when it is safe to delete a brush, because the referencing counting does it for you. For example, you can set a brush in a device context, and then immediately delete the brush you passed, because the brush is 'copied'.

So you may find it easier to ignore the brush list, and instead create and copy brushes as you see fit. If your Windows resource meter suggests your application is using too many resources, you can resort to using GDI lists to share objects explicitly.

The only compelling use for the brush list is for wxWindows to keep track of brushes in order to clean them up on exit. It is also kept for backward compatibility with earlier versions of wxWindows.

See also

wxBrush (p. 61)

4.15.1 wxBrushList::wxBrushList

void wxBrushList()

Constructor. The application should not construct its own brush list: use the object pointer **wxTheBrushList**.

4.15.2 wxBrushList::AddBrush**void AddBrush(wxBrush *brush)**

Used internally by wxWindows to add a brush to the list.

4.15.3 wxBrushList::FindOrCreateBrush**wxBrush * FindOrCreateBrush(const wxColour& colour, int style)**

Finds a brush with the specified attributes and returns it, else creates a new brush, adds it to the brush list, and returns it.

wxBrush * FindOrCreateBrush(const wxString& colourName, int style)

Finds a brush with the specified attributes and returns it, else creates a new brush, adds it to the brush list, and returns it.

Finds a brush of the given specification, or creates one and adds it to the list.

Parameters

colour

Colour object.

colourName

Colour name, which should be in the colour database.

style

Brush style. See *wxBrush::SetStyle* (p. 66) for a list of styles.

4.15.4 wxBrushList::RemoveBrush**void RemoveBrush(wxBrush *brush)**

Used by wxWindows to remove a brush from the list.

4.16 wxCalculateLayoutEvent

This event is sent by *wxLayoutAlgorithm* (p. 362) to calculate the amount of the remaining client area that the window should occupy.

Derived from

wxEvt (p. 221)
wxObject (p. 471)

Include files

<wx/laywin.h>

Event table macros

EVT_CALCULATE_LAYOUT(func) Process a `wxEVT_CALCULATE_LAYOUT` event, which asks the window to take a 'bite' out of a rectangle provided by the algorithm.

See also

wxQueryLayoutInfoEvent (p. 535), *wxSashLayoutWindow* (p. 570), *wxLayoutAlgorithm* (p. 362).

4.16.1 wxCalculateLayoutEvent::wxCalculateLayoutEvent

wxCalculateLayoutEvent(wxWindowID id = 0)

Constructor.

4.16.2 wxCalculateLayoutEvent::GetFlags

int GetFlags() const

Returns the flags associated with this event. Not currently used.

4.16.3 wxCalculateLayoutEvent::GetRect

wxRect GetRect() const

Before the event handler is entered, returns the remaining parent client area that the window could occupy. When the event handler returns, this should contain the remaining parent client rectangle, after the event handler has subtracted the area that its window occupies.

4.16.4 wxCalculateLayoutEvent::SetFlags

void SetFlags(int flags)

Sets the flags associated with this event. Not currently used.

4.16.5 wxCalculateLayoutEvent::SetRect

void SetRect(const wxRect& rect)

Call this to specify the new remaining parent client area, after the space occupied by the window has been subtracted.

4.17 wxCheckBox

A checkbox is a labelled box which is either on (checkmark is visible) or off (no checkmark).

Derived from

wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/checkbox.h>

Window styles

There are no special styles for `wxCheckBox`.

See also *window styles overview* (p. 959).

Event handling

EVT_CHECKBOX(id, func)

Process a
`wxEVT_COMMAND_CHECKBOX_CLICKED`
event, when the checkbox is clicked.

See also

wxRadioButton (p. 543), *wxCommandEvent* (p. 103)

4.17.1 wxCheckBox::wxCheckBox

wxCheckBox()

Default constructor.

wxCheckBox(wxWindow* *parent*, wxWindowID *id*, const wxString& *label*, const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size* = wxDefaultSize, long *style* = 0, const wxValidator& *val*, const wxString& *name* = "checkBox")

Constructor, creating and showing a checkbox.

Parameters

parent

Parent window. Must not be NULL.

id

Checkbox identifier. A value of -1 indicates a default value.

label

Text to be displayed next to the checkbox.

pos

Checkbox position. If the position (-1, -1) is specified then a default position is chosen.

size

Checkbox size. If the default size (-1, -1) is specified then a default size is chosen.

style

Window style. See *wxCheckBox* (p. 70).

validator

Window validator.

name

Window name.

See also

wxCheckBox::Create (p. 71), *wxValidator* (p. 781)

4.17.2 wxCheckBox::~~wxCheckBox**~wxCheckBox()**

Destructor, destroying the checkbox.

4.17.3 wxCheckBox::Create

bool Create(wxWindow* parent, wxWindowID id, const wxString& label, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0, const wxValidator& val, const wxString& name = "checkBox")

Creates the checkbox for two-step construction. See `wxCheckBox::wxCheckBox` (p. 70) for details.

4.17.4 wxCheckBox::GetValue

bool GetValue() const

Gets the state of the checkbox.

Return value

Returns TRUE if it is checked, FALSE otherwise.

4.17.5 wxCheckBox::SetValue

void SetValue(const bool state)

Sets the checkbox to the given state.

Parameters

state

If TRUE, the check is on, otherwise it is off.

4.18 wxCheckListBox

A checklistbox is like a listbox, but allows items to be checked or unchecked.

This class is currently implemented under Windows and GTK. When using this class under Windows `wxWindows` must be compiled with `USE_OWNER_DRAWN` set to 1.

Only the new functions for this class are documented; see also `wxListBox` (p. 373).

Derived from

`wxListBox` (p. 373)

`wxControl` (p. 125)

`wxWindow` (p. 798)

`wxEvtHandler` (p. 224)

`wxObject` (p. 471)

Include files

<wx/checklst.h>

Window styles

See *wxListBox* (p. 373).

Event handling

EVT_CHECKLISTBOX(id, func)	Process a wxEVT_COMMAND_CHECKLISTBOX_TOGG LE event, when an item in the check list box is checked or unchecked.
-----------------------------------	--------------------------------------------------------------------------------------------------------------------------

See also

wxListBox (p. 373), *wxChoice* (p. 75), *wxComboBox* (p. 94), *wxListCtrl* (p. 381),
wxCommandEvent (p. 103)

4.18.1 wxCheckListBox::wxCheckListBox

wxCheckListBox()

Default constructor.

wxCheckListBox(wxWindow* parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, int n, const wxString choices[] = NULL, long style = 0, const wxValidator& validator = wxDefaultValidator, const wxString& name = "listBox")

Constructor, creating and showing a list box.

Parameters

parent

Parent window. Must not be NULL.

id

Window identifier. A value of -1 indicates a default value.

pos

Window position.

size

Window size. If the default size (-1, -1) is specified then the window is sized appropriately.

n

Number of strings with which to initialise the control.

choices

An array of strings with which to initialise the control.

style

Window style. See *wxCheckListBox* (p. 72).

validator

Window validator.

name

Window name.

wxPython note:

The *wxCheckListBox* constructor in wxPython reduces the *nand* and *choices* arguments are to a single argument, which is a list of strings.

4.18.2 wxCheckListBox::~~wxCheckListBox

void ~wxCheckListBox()

Destructor, destroying the list box.

4.18.3 wxCheckListBox::Check

void Check(int *item*, bool *check* = TRUE)

Checks the given item.

Parameters

item

Index of item to check.

check

TRUE if the item is to be checked, FALSE otherwise.

4.18.4 wxCheckListBox::IsChecked

bool IsChecked(int *item*) const

Returns TRUE if the given item is checked, FALSE otherwise.

Parameters

item

Index of item whose check status is to be returned.

4.19 wxChoice

A choice item is used to select one of a list of strings. Unlike a listbox, only the selection is visible until the user pulls down the menu of choices.

Derived from

wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/choice.h>

Window styles

There are no special styles for wxChoice.

See also *window styles overview* (p. 959).

Event handling

EVT_CHOICE(id, func)	Process a wxEVT_COMMAND_CHOICE_SELECTED event, when an item on the list is selected.
-----------------------------	--------------------------------------------------------------------------------------------

See also

wxListBox (p. 373), *wxComboBox* (p. 94), *wxCommandEvent* (p. 103)

4.19.1 wxChoice::wxChoice

wxChoice()

Default constructor.

wxChoice(wxWindow *parent, wxWindowID id, const wxPoint& pos, const wxSize& size, int n, const wxString choices[], long style = 0, const wxValidator& validator = wxDefaultValidator, const wxString& name = "choice")

Constructor, creating and showing a choice.

Parameters

parent

Parent window. Must not be NULL.

id

Window identifier. A value of -1 indicates a default value.

pos

Window position.

size

Window size. If the default size (-1, -1) is specified then the choice is sized appropriately.

n

Number of strings with which to initialise the choice control.

choices

An array of strings with which to initialise the choice control.

style

Window style. See *wxChoice* (p. 75).

validator

Window validator.

name

Window name.

See also

wxChoice::Create (p. 77), *wxValidator* (p. 781)

wxPython note:

The *wxChoice* constructor in wxPython reduces the *nand choices* arguments are to a single argument, which is a list of strings.

4.19.2 wxChoice::~~wxChoice

~wxChoice()

Destructor, destroying the choice item.

4.19.3 wxChoice::Append

void Append(const wxString& item)

Adds the item to the end of the choice control.

Parameters

item
String to add.

4.19.4 wxChoice::Clear

void Clear()

Clears the strings from the choice item.

4.19.5 wxChoice::Create

bool Create(wxWindow *parent, wxWindowID id, const wxPoint& pos, const wxSize& size, int n, const wxString choices[], long style = 0, const wxString& name = "choice")

Creates the choice for two-step construction. See *wxChoice::wxChoice* (p. 75).

4.19.6 wxChoice::FindString

int FindString(const wxString& string) const

Finds a choice matching the given string.

Parameters

string
String to find.

Return value

Returns the position if found, or -1 if not found.

4.19.7 wxChoice::GetColumns

int GetColumns() const

Gets the number of columns in this choice item.

Remarks

This is implemented for Motif only.

4.19.8 wxChoice::GetSelection

int GetSelection() const

Gets the id (position) of the selected string, or -1 if there is no selection.

4.19.9 wxChoice::GetString**wxString GetString(int *n*) const**

Returns the string at the given position.

Parameters

n
The zero-based position.

Return value

The string at the given position, or the empty string if *n* is invalid.

4.19.10 wxChoice::GetStringSelection**wxString GetStringSelection() const**

Gets the selected string, or the empty string if no string is selected.

4.19.11 wxChoice::Number**int Number() const**

Returns the number of strings in the choice control.

4.19.12 wxChoice::SetColumns**void SetColumns(int *n* = 1)**

Sets the number of columns in this choice item.

Parameters

n
Number of columns.

Remarks

This is implemented for Motif only.

4.19.13 wxChoice::SetSelection

void SetSelection(int *n*)

Sets the choice by passing the desired string position.

Parameters

n
The string position to select, starting from zero.

See also

wxChoice::SetStringSelection (p. 79)

4.19.14 wxChoice::SetStringSelection

void SetStringSelection(const wxString& *string*)

Sets the choice by passing the desired string.

Parameters

string
The string to select.

See also

wxChoice::SetSelection (p. 79)

4.20 wxClassInfo

This class stores meta-information about classes. Instances of this class are not generally defined directly by an application, but indirectly through use of macros such as **DECLARE_DYNAMIC_CLASS** and **IMPLEMENT_DYNAMIC_CLASS**.

Derived from

No parent class.

Include files

<wx/object.h>

See also

Overview (p. 958), *wxObject* (p. 471)

4.20.1 wxClassInfo::wxClassInfo

wxClassInfo(char* className, char* baseClass1, char* baseClass2, int size, wxObjectConstructorFn fn)

Constructs a wxClassInfo object. The supplied macros implicitly construct objects of this class, so there is no need to create such objects explicitly in an application.

4.20.2 wxClassInfo::CreateObject

wxObject* CreateObject()

Creates an object of the appropriate kind. Returns NULL if the class has not been declared dynamically createable (typically, it's an abstract class).

4.20.3 wxClassInfo::FindClass

static wxClassInfo * FindClass(char* name)

Finds the wxClassInfo object for a class of the given string name.

4.20.4 wxClassInfo::GetBaseClassName1

char* GetBaseClassName1() const

Returns the name of the first base class (NULL if none).

4.20.5 wxClassInfo::GetBaseClassName2

char* GetBaseClassName2() const

Returns the name of the second base class (NULL if none).

4.20.6 wxClassInfo::GetClassName

char * GetClassName() const

Returns the string form of the class name.

4.20.7 wxClassInfo::GetSize

int GetSize() const

Returns the size of the class.

4.20.8 wxClassInfo::InitializeClasses

static void InitializeClasses()

Initializes pointers in the wxClassInfo objects for fast execution of IsKindOf. Called in base wxWindows library initialization.

4.20.9 wxClassInfo::IsKindOf

bool IsKindOf(wxClassInfo* info)

Returns TRUE if this class is a kind of (inherits from) the given class.

4.21 wxClientDC

A wxClientDC must be constructed if an application wishes to paint on the client area of a window from outside an **OnPaint** event. This should normally be constructed as a temporary stack object; don't store a wxClientDC object.

To draw on a window from within **OnPaint**, construct a *wxPaintDC* (p. 483) object.

To draw on the whole window including decorations, construct a *wxWindowDC* (p. 843) object (Windows only).

Derived from

wxDC (p. 151)

Include files

<wx/dcclient.h>

See also

wxDC (p. 151), *wxMemoryDC* (p. 415), *wxPaintDC* (p. 483), *wxWindowDC* (p. 843), *wxScreenDC* (p. 578)

4.21.1 wxClientDC::wxClientDC

wxClientDC(wxWindow* window)

Constructor. Pass a pointer to the window on which you wish to paint.

4.22 wxClipboard

A class for manipulating the clipboard. Note that this is not compatible with the clipboard class from wxWindows 1.xx, which has the same name but a different implementation.

To use the clipboard, you call member functions of the global **wxTheClipboard** object.

Call *wxClipboard::Open* (p. 84) to get ownership of the clipboard. If this operation returns TRUE, you now own the clipboard. Call *wxClipboard::AddData* (p. 83) to put data on the clipboard (one or more times), or *wxClipboard::GetData* (p. 83) to retrieve data from the clipboard. Call *wxClipboard::Close* (p. 83) to close the clipboard and relinquish ownership. You should keep the clipboard open only momentarily.

For example:

```
// Write some text to the clipboard
if (wxTheClipboard->Open())
{
    // This data objects are held by the clipboard,
    // so do not delete them in the app.
    wxTheClipboard->AddData( new wxTextDataObject("Some text") );
    wxTheClipboard->Close();
}

// Read some text
if (wxTheClipboard->Open())
{
    wxTextDataObject data;
    if (wxTheClipboard->IsSupported(data))
    {
        wxTheClipboard->GetData(data);
        wxMessageBox(data.GetText());
    }
    wxTheClipboard->Close();
}
```

Derived from

wxObject (p. 471)

Include files

<wx/clipbrd.h>

See also

Drag and drop overview (p. 975), *wxDataObject* (p. 138)

4.22.1 wxClipboard::wxClipboard

wxClipboard()

Constructor.

4.22.2 wxClipboard::~~wxClipboard

~wxClipboard()

Destructor.

4.22.3 wxClipboard::AddData

bool AddData(wxDataObject* data)

Call this function to add a data object to the clipboard. This function can be called several times to put different formats on the clipboard.

4.22.4 wxClipboard::Clear

void Clear()

Clears the global clipboard object and the system's clipboard if possible.

4.22.5 wxClipboard::Close

bool Close()

Call this function to close the clipboard, having opened it with *wxClipboard::Open* (p. 84).

4.22.6 wxClipboard::GetData

bool GetData(wxDataObject& data)

Call this function to fill *data* with data on the clipboard, if available in the required format. Returns TRUE on success.

4.22.7 wxClipboard::IsSupported

bool IsSupported(wxDataObject& data)

Returns TRUE if the format of the given data object is available on the clipboard.

4.22.8 wxClipboard::Open

bool Open()

Call this function to open the clipboard before calling *wxClipboard::SetData* (p. 84) and *wxClipboard::GetData* (p. 83).

Call *wxClipboard::Close* (p. 83) when you have finished with the clipboard. You should keep the clipboard open for only a very short time.

Returns TRUE on success. This should be tested (as in the sample shown above).

4.22.9 wxClipboard::SetData

bool SetData(wxDataObject* data)

Call this function to set the data object to the clipboard. This function will clear all previous contents in the clipboard, so calling it several times does not make any sense.

4.23 wxCloseEvent

This event class contains information about window and session close events.

Derived from

wxEvent (p. 221)

Include files

<wx/event.h>

Event table macros

To process a close event, use these event handler macros to direct input to member functions that take a *wxCloseEvent* argument.

EVT_CLOSE(func)	Process a close event, supplying the member function. This event applies to <i>wxFrame</i> and <i>wxDialog</i> classes.
EVT_QUERY_END_SESSION(func)	Process a query end session event, supplying the member function. This event applies to <i>wxApp</i> only.
EVT_END_SESSION(func)	Process an end session event, supplying the member function. This event applies to <i>wxApp</i> only.

See also

wxWindow::OnCloseWindow (p. 819), *wxWindow::Close* (p. 802),
wxApp::OnQueryEndSession (p. 12), *wxApp::OnEndSession* (p. 11), *Window deletion overview* (p. 929)

4.23.1 wxCloseEvent::wxCloseEvent

wxCloseEvent(WXTYPE *commandEventType* = 0, int *id* = 0)

Constructor.

4.23.2 wxCloseEvent::CanVeto

bool CanVeto()

Returns TRUE if you can veto a system shutdown or a window close event. Vetoing a window close event is not possible if the calling code wishes to force the application to exit, and so this function must be called to check this.

4.23.3 wxCloseEvent::GetLoggingOff

bool GetLoggingOff() const

Returns TRUE if the user is logging off.

4.23.4 wxCloseEvent::GetSessionEnding

bool GetSessionEnding() const

Returns TRUE if the session is ending.

4.23.5 wxCloseEvent::GetForce

bool GetForce() const

Returns TRUE if the application wishes to force the window to close. This will shortly be obsolete, replaced by CanVeto.

4.23.6 wxCloseEvent::SetCanVeto

void SetCanVeto(bool *canVeto*)

Sets the 'can veto' flag.

4.23.7 `wxCloseEvent::SetForce`

void SetForce(bool *force*) const

Sets the 'force' flag.

4.23.8 `wxCloseEvent::SetLoggingOff`

void SetLoggingOff(bool *loggingOff*) const

Sets the 'logging off' flag.

4.23.9 `wxCloseEvent::Veto`

void Veto(bool *veto* = *TRUE*)

Call this from your event handler to veto a system shutdown or to signal to the calling application that a window close did not happen.

You can only veto a shutdown if `wxCloseEvent::CanVeto` (p. 85) returns *TRUE*.

4.24 `wxColour`

A colour is an object representing a combination of Red, Green, and Blue (RGB) intensity values, and is used to determine drawing colours. See the entry for `wxColourDatabase` (p. 91) for how a pointer to a predefined, named colour may be returned instead of creating a new colour.

Valid RGB values are in the range 0 to 255.

Derived from

`wxObject` (p. 471)

Include files

<wx/colour.h>

Predefined objects

Objects:

`wxNullColour`

Pointers:

wxBLACK
wxWHITE
wxRED
wxBLUE
wxGREEN
wxCYAN
wxLIGHT_GREY

See also

wxColourDatabase (p. 91), *wxPen* (p. 494), *wxBrush* (p. 61), *wxColourDialog* (p. 93)

4.24.1 wxColour::wxColour

wxColour()

Default constructor.

wxColour(const unsigned char *red*, const unsigned char *green*, const unsigned char *blue*)

Constructs a colour from red, green and blue values.

wxColour(const wxString& *colourName*)

Constructs a colour object using a colour name listed in **wxTheColourDatabase**.

wxColour(const wxColour& *colour*)

Copy constructor.

Parameters

red

The red value.

green

The green value.

blue

The blue value.

colourName

The colour name.

colour

The colour to copy.

See also

wxColourDatabase (p. 91)

wxPython note:

Constructors supported by wxPython are:

```
wxColour(red=0, green=0, blue=0)  
wxNamedColour(name)
```

4.24.2 wxColour::Blue

unsigned char Blue() const

Returns the blue intensity.

4.24.3 wxColour::GetPixel

long GetPixel() const

Returns a pixel value which is platform-dependent. On Windows, a COLORREF is returned. On X, an allocated pixel value is returned.

-1 is returned if the pixel is invalid (on X, unallocated).

4.24.4 wxColour::Green

unsigned char Green() const

Returns the green intensity.

4.24.5 wxColour::Ok

bool Ok() const

Returns TRUE if the colour object is valid (the colour has been initialised with RGB values).

4.24.6 wxColour::Red

unsigned char Red() const

Returns the red intensity.

4.24.7 wxColour::Set

void Set(const unsigned char *red*, const unsigned char *green*, const unsigned char *blue*)

Sets the RGB intensity values.

4.24.8 wxColour::operator =

wxColour& operator =(const wxColour& *colour*)

Assignment operator, taking another colour object.

wxColour& operator =(const wxString& *colourName*)

Assignment operator, using a colour name to be found in the colour database.

[See also](#)

wxColourDatabase (p. 91)

4.24.9 wxColour::operator ==

bool operator ==(const wxColour& *colour*)

Tests the equality of two colours by comparing individual red, green blue colours.

4.24.10 wxColour::operator !=

bool operator !=(const wxColour& *colour*)

Tests the inequality of two colours by comparing individual red, green blue colours.

4.25 wxColourData

This class holds a variety of information related to colour dialogs.

[Derived from](#)

wxObject (p. 471)

[Include files](#)

<wx/cmndata.h>

See also

wxColour (p. 86), *wxColourDialog* (p. 93), *wxColourDialog overview* (p. 915)

4.25.1 wxColourData::wxColourData**wxColourData()**

Constructor. Initializes the custom colours to white, the *data colour* setting to black, and the *choose full* setting to TRUE.

4.25.2 wxColourData::~~wxColourData**~wxColourData()**

Destructor.

4.25.3 wxColourData::GetChooseFull**bool GetChooseFull() const**

Under Windows, determines whether the Windows colour dialog will display the full dialog with custom colour selection controls. Has no meaning under other platforms.

The default value is TRUE.

4.25.4 wxColourData::GetColour**wxColour& GetColour() const**

Gets the current colour associated with the colour dialog.

The default colour is black.

4.25.5 wxColourData::GetCustomColour**wxColour& GetCustomColour(int i) const**

Gets the *i*th custom colour associated with the colour dialog. *i* should be an integer between 0 and 15.

The default custom colours are all white.

4.25.6 wxColourData::SetChooseFull

void SetChooseFull(const bool *flag*)

Under Windows, tells the Windows colour dialog to display the full dialog with custom colour selection controls. Under other platforms, has no effect.

The default value is TRUE.

4.25.7 wxColourData::SetColour

void SetColour(const wxColour& *colour*)

Sets the default colour for the colour dialog.

The default colour is black.

4.25.8 wxColourData::SetCustomColour

void SetColour(int *i*, const wxColour& *colour*)

Sets the *i*th custom colour for the colour dialog. *i* should be an integer between 0 and 15.

The default custom colours are all white.

4.25.9 wxColourData::operator =

void operator =(const wxColourData& *data*)

Assignment operator for the colour data.

4.26 wxColourDatabase

wxWindows maintains a database of standard RGB colours for a predefined set of named colours (such as "BLACK", "LIGHT GREY"). The application may add to this set if desired by using *Append*. There is only one instance of this class:
wxTheColourDatabase.

Derived from

wxList (p. 367)

wxObject (p. 471)

Include files

<wx/gdicmn.h>

Remarks

The colours in the standard database are as follows:

AQUAMARINE, BLACK, BLUE, BLUE VIOLET, BROWN, CADET BLUE, CORAL, CORNFLOWER BLUE, CYAN, DARK GREY, DARK GREEN, DARK OLIVE GREEN, DARK ORCHID, DARK SLATE BLUE, DARK SLATE GREY, DARK TURQUOISE, DIM GREY, FIREBRICK, FOREST GREEN, GOLD, GOLDENROD, GREY, GREEN, GREEN YELLOW, INDIAN RED, KHAKI, LIGHT BLUE, LIGHT GREY, LIGHT STEEL BLUE, LIME GREEN, MAGENTA, MAROON, MEDIUM AQUAMARINE, MEDIUM BLUE, MEDIUM FOREST GREEN, MEDIUM GOLDENROD, MEDIUM ORCHID, MEDIUM SEA GREEN, MEDIUM SLATE BLUE, MEDIUM SPRING GREEN, MEDIUM TURQUOISE, MEDIUM VIOLET RED, MIDNIGHT BLUE, NAVY, ORANGE, ORANGE RED, ORCHID, PALE GREEN, PINK, PLUM, PURPLE, RED, SALMON, SEA GREEN, SIENNA, SKY BLUE, SLATE BLUE, SPRING GREEN, STEEL BLUE, TAN, THISTLE, TURQUOISE, VIOLET, VIOLET RED, WHEAT, WHITE, YELLOW, YELLOW GREEN.

See also

wxColour (p. 86)

4.26.1 wxColourDatabase::wxColourDatabase

wxColourDatabase()

Constructs the colour database.

4.26.2 wxColourDatabase::FindColour

wxColour* FindColour(const wxString& colourName)

Finds a colour given the name. Returns NULL if not found.

4.26.3 wxColourDatabase::FindName

wxString FindName(const wxColour& colour) const

Finds a colour name given the colour. Returns NULL if not found.

4.26.4 wxColourDatabase::Initialize

void Initialize()

Initializes the database with a number of stock colours. Called by `wxWindows` on start-up.

4.27 `wxColourDialog`

This class represents the colour chooser dialog.

Derived from

`wxDialog` (p. 178)
`wxWindow` (p. 798)
`wxEvtHandler` (p. 224)
`wxObject` (p. 471)

Include files

<wx/colordlg.h>

See also

wxColourDialog Overview (p. 915), *wxColour* (p. 86), *wxColourData* (p. 89)

4.27.1 `wxColourDialog::wxColourDialog`

`wxColourDialog(wxWindow* parent, wxColourData* data = NULL)`

Constructor. Pass a parent window, and optionally a pointer to a block of colour data, which will be copied to the colour dialog's colour data.

See also

wxColourData (p. 89)

4.27.2 `wxColourDialog::~wxColourDialog`

`~wxColourDialog()`

Destructor.

4.27.3 `wxColourDialog::GetColourData`

`wxColourData& GetColourData()`

Returns the *colour data* (p. 89) associated with the colour dialog.

4.27.4 wxColourDialog::ShowModal

int ShowModal()

Shows the dialog, returning wxID_OK if the user pressed OK, and wxOK_CANCEL otherwise.

4.28 wxComboBox

A combobox is like a combination of an edit control and a listbox. It can be displayed as static list with editable or read-only text field; or a drop-down list with text field; or a drop-down list without a text field.

A combobox permits a single selection only. Combobox items are numbered from zero.

Derived from

wxChoice (p. 75)
wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/combo.h>

Window styles

wxCB_SIMPLE	Creates a combobox with a permanently displayed list.
wxCB_DROPDOWN	Creates a combobox with a drop-down list.
wxCB_READONLY	Creates a combo box consisting of a drop-down list and static text item displaying the current selection.
wxCB_SORT	Sorts the entries in the list alphabetically.

See also *window styles overview* (p. 959).

Event handling

EVT_COMBOBOX(id, func)	Process a wxEVT_COMMAND_COMBOBOX_SELECTED event, when an item on the list is selected.
EVT_TEXT(id, func)	Process a wxEVT_COMMAND_TEXT_UPDATED event, when the combobox text changes.

See also

wxListBox (p. 373), *wxTextCtrl* (p. 712), *wxChoice* (p. 75), *wxCommandEvent* (p. 103)

4.28.1 wxComboBox::wxComboBox**wxComboBox()**

Default constructor.

```
wxComboBox(wxWindow* parent, wxWindowID id, const wxString& value = "",  
const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, int n,  
const wxString choices[], long style = 0, const wxValidator& validator =  
wxDefaultValidator, const wxString& name = "comboBox")
```

Constructor, creating and showing a combobox.

Parameters

parent

Parent window. Must not be NULL.

id

Window identifier. A value of -1 indicates a default value.

pos

Window position.

size

Window size. If the default size (-1, -1) is specified then the window is sized appropriately.

n

Number of strings with which to initialise the control.

choices

An array of strings with which to initialise the control.

style

Window style. See *wxComboBox* (p. 94).

validator

Window validator.

name

Window name.

See also

wxComboBox::Create (p. 96), *wxValidator* (p. 781)

wxPython note:

The `wxComboBox` constructor in `wxPython` reduces the `nand choices` arguments are to a single argument, which is a list of strings.

4.28.2 `wxComboBox::~~wxComboBox`

`~wxComboBox()`

Destructor, destroying the combobox.

4.28.3 `wxComboBox::Append`

`void Append(const wxString& item)`

Adds the item to the end of the combobox.

`void Append(const wxString& item, char* clientData)`

Adds the item to the end of the combobox, associating the given data with the item.

Parameters

item

The string to add.

clientData

Client data to associate with the item.

4.28.4 `wxComboBox::Clear`

`void Clear()`

Clears all strings from the combobox.

4.28.5 `wxComboBox::Create`

`bool Create(wxWindow* parent, wxWindowID id, const wxString& value = "", const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, int n, const wxString choices[], long style = 0, const wxValidator& validator = wxDefaultValidator, const wxString& name = "comboBox")`

Creates the combobox for two-step construction. Derived classes should call or replace this function. See `wxComboBox::wxComboBox` (p. 95) for further details.

4.28.6 wxComboBox::Copy

void Copy()

Copies the selected text to the clipboard.

4.28.7 wxComboBox::Cut

void Cut()

Copies the selected text to the clipboard and removes the selection.

4.28.8 wxComboBox::Delete

void Delete(int *n*)

Deletes an item from the combobox.

Parameters

n
The item to delete, starting from zero.

4.28.9 wxComboBox::FindString

int FindString(const wxString& *string*)

Finds a choice matching the given string.

Parameters

string
The item to find.

Return value

The position if found, or -1 if not found.

4.28.10 wxComboBox::GetClientData

char* GetClientData(int *n*) const

Returns a pointer to the client data associated with the given item (if any).

Parameters

n
An item, starting from zero.

Return value

A pointer to the client data, or NULL if the item was not found.

4.28.11 wxComboBox::GetInsertionPoint

long GetInsertionPoint() const

Returns the insertion point for the combobox's text field.

4.28.12 wxComboBox::GetLastPosition

long GetLastPosition() const

Returns the last position in the combobox text field.

4.28.13 wxComboBox::GetSelection

int GetSelection() const

Gets the position of the selected string, or -1 if there is no selection.

4.28.14 wxComboBox::GetString

wxString GetString(int *n*) const

Returns the string at position *n*.

Parameters

n
The item position, starting from zero.

Return value

The string if the item is found, otherwise the empty string.

4.28.15 wxComboBox::GetStringSelection

wxString GetStringSelection() const

Gets the selected string.

4.28.16 wxComboBox::GetValue

wxString GetValue() const

Returns the current value in the combobox text field.

4.28.17 wxComboBox::Number

int Number() const

Returns the number of items in the combobox list.

4.28.18 wxComboBox::Paste

void Paste()

Pastes text from the clipboard to the text field.

4.28.19 wxComboBox::Replace

void Replace(long from, long to, const wxString& text)

Replaces the text between two positions with the given text, in the combobox text field.

Parameters

from
The first position.

to
The second position.

text
The text to insert.

4.28.20 wxComboBox::Remove

void Remove(long from, long to)

Removes the text between the two positions in the combobox text field.

Parameters

from

The first position.

to

The last position.

4.28.21 wxComboBox::SetClientData

void SetClientData(int *n*, char* *data*)

Associates the given client data pointer with the given item.

Parameters

n

The zero-based item.

data

The client data.

4.28.22 wxComboBox::SetInsertionPoint

void SetInsertionPoint(long *pos*)

Sets the insertion point in the combobox text field.

Parameters

pos

The new insertion point.

4.28.23 wxComboBox::SetInsertionPointEnd

void SetInsertionPointEnd()

Sets the insertion point at the end of the combobox text field.

4.28.24 wxComboBox::SetSelection

void SetSelection(int *n*)

Selects the given item in the combobox list.

void SetSelection(long *from*, long *to*)

Selects the text between the two positions, in the combobox text field.

Parameters

n
The zero-based item to select.

from
The first position.

to
The second position.

4.28.25 wxComboBox::SetValue

void SetValue(const wxString& text)

Sets the text for the combobox text field.

Parameters

text
The text to set.

4.29 wxCommand

`wxCommand` is a base class for modelling an application command, which is an action usually performed by selecting a menu item, pressing a toolbar button or any other means provided by the application to change the data or view.

Derived from

wxObject (p. 471)

Include files

<wx/docview.h>

See also

Overview (p. 937)

4.29.1 wxCommand::wxCommand

wxCommand(bool canUndo = FALSE, const wxString& name = NULL)

Constructor. `wxCommand` is an abstract class, so you will need to derive a new class and call this constructor from your own constructor.

canUndo tells the command processor whether this command is undo-able. You can achieve the same functionality by overriding the `CanUndo` member function (if for example the criteria for undoability is context-dependant).

name must be supplied for the command processor to display the command name in the application's edit menu.

4.29.2 wxCommand::~~wxCommand

~wxCommand()

Destructor.

4.29.3 wxCommand::CanUndo

bool CanUndo()

Returns TRUE if the command can be undone, FALSE otherwise.

4.29.4 wxCommand::Do

bool Do()

Override this member function to execute the appropriate action when called. Return TRUE to indicate that the action has taken place, FALSE otherwise. Returning FALSE will indicate to the command processor that the action is not undoable and should not be added to the command history.

4.29.5 wxCommand::GetName

wxString GetName()

Returns the command name.

4.29.6 wxCommand::Undo

bool Undo()

Override this member function to un-execute a previous Do. Return TRUE to indicate that the action has taken place, FALSE otherwise. Returning FALSE will indicate to the command processor that the action is not redoable and no change should be made to the command history.

How you implement this command is totally application dependent, but typical strategies

include:

- Perform an inverse operation on the last modified piece of data in the document. When redone, a copy of data stored in command is pasted back or some operation reapplied. This relies on the fact that you know the ordering of Undos; the user can never Undo at an arbitrary position in the command history.
- Restore the entire document state (perhaps using document transactioning). Potentially very inefficient, but possibly easier to code if the user interface and data are complex, and an 'inverse execute' operation is hard to write.

The docview sample uses the first method, to remove or restore segments in the drawing.

4.30 wxCommandEvent

This event class contains information about command events, which originate from a variety of simple controls. More complex controls, such as *wxTreeCtrl* (p. 761), have separate command event classes.

Derived from

wxEvent (p. 221)

Include files

<wx/event.h>

Event table macros

To process a menu command event, use these event handler macros to direct input to member functions that take a *wxCommandEvent* argument.

EVT_COMMAND(id, event, func)	Process a command, supplying the window identifier, command event identifier, and member function.
EVT_COMMAND_RANGE(id1, id2, event, func)	Process a command for a range of window identifiers, supplying the minimum and maximum window identifiers, command event identifier, and member function.
EVT_BUTTON(id, func)	Process a <code>wxEVT_COMMAND_BUTTON_CLICKED</code> command, which is generated by a <code>wxButton</code> control.
EVT_CHECKBOX(id, func)	Process a <code>wxEVT_COMMAND_CHECKBOX_CLICKED</code> command, which is generated by a <code>wxCheckBox</code> control.
EVT_CHOICE(id, func)	Process a <code>wxEVT_COMMAND_CHOICE_SELECTED</code> command, which is generated by a <code>wxChoice</code> control.

EVT_LISTBOX(id, func)	Process a wxEVT_COMMAND_LISTBOX_SELECTED command, which is generated by a wxListBox control.
EVT_LISTBOX_DCLICK(id, func)	Process a wxEVT_COMMAND_LISTBOX_DOUBLECLICKED command, which is generated by a wxListBox control.
EVT_TEXT(id, func)	Process a wxEVT_COMMAND_TEXT_UPDATED command, which is generated by a wxTextCtrl control.
EVT_TEXT_ENTER(id, func)	Process a wxEVT_COMMAND_TEXT_ENTER command, which is generated by a wxTextCtrl control.
EVT_MENU(id, func)	Process a wxEVT_COMMAND_MENU_SELECTED command, which is generated by a menu item.
EVT_MENU_RANGE(id1, id2, func)	Process a wxEVT_COMMAND_MENU_RANGE command, which is generated by a range of menu items.
EVT_SLIDER(id, func)	Process a wxEVT_COMMAND_SLIDER_UPDATED command, which is generated by a wxSlider control.
EVT_RADIOBOX(id, func)	Process a wxEVT_COMMAND_RADIOBOX_SELECTED command, which is generated by a wxRadioBox control.
EVT_RADIOBUTTON(id, func)	Process a wxEVT_COMMAND_RADIOBUTTON_SELECTED command, which is generated by a wxRadioButton control.
EVT_SCROLLBAR(id, func)	Process a wxEVT_COMMAND_SCROLLBAR_UPDATED command, which is generated by a wxScrollBar control. This is provided for compatibility only; more specific scrollbar event macros should be used instead (see <i>wxScrollEvent</i> (p. 584)).
EVT_COMBOBOX(id, func)	Process a wxEVT_COMMAND_COMBOBOX_SELECTED command, which is generated by a wxComboBox control.
EVT_TOOL(id, func)	Process a wxEVT_COMMAND_TOOL_CLICKED event (a synonym for wxEVT_COMMAND_MENU_SELECTED). Pass the id of the tool.
EVT_TOOL_RANGE(id1, id2, func)	Process a wxEVT_COMMAND_TOOL_CLICKED event

	for a range id identifiers. Pass the ids of the tools.
EVT_TOOL_RCLICKED(id, func)	Process a wxEVT_COMMAND_TOOL_RCLICKED event. Pass the id of the tool.
EVT_TOOL_RCLICKED_RANGE(id1, id2, func)	Process a wxEVT_COMMAND_TOOL_RCLICKED event for a range of ids. Pass the ids of the tools.
EVT_TOOL_ENTER(id, func)	Process a wxEVT_COMMAND_TOOL_ENTER event. Pass the id of the toolbar itself. The value of wxCommandEvent::GetSelection is the tool id, or -1 if the mouse cursor has moved off a tool.
EVT_COMMAND_LEFT_CLICK(id, func)	Process a wxEVT_COMMAND_LEFT_CLICK command, which is generated by a control (Windows 95 and NT only).
EVT_COMMAND_LEFT_DCLICK(id, func)	Process a wxEVT_COMMAND_LEFT_DCLICK command, which is generated by a control (Windows 95 and NT only).
EVT_COMMAND_RIGHT_CLICK(id, func)	Process a wxEVT_COMMAND_RIGHT_CLICK command, which is generated by a control (Windows 95 and NT only).
EVT_COMMAND_SET_FOCUS(id, func)	Process a wxEVT_COMMAND_SET_FOCUS command, which is generated by a control (Windows 95 and NT only).
EVT_COMMAND_KILL_FOCUS(id, func)	Process a wxEVT_COMMAND_KILL_FOCUS command, which is generated by a control (Windows 95 and NT only).
EVT_COMMAND_ENTER(id, func)	Process a wxEVT_COMMAND_ENTER command, which is generated by a control.

4.30.1 wxCommandEvent::m_clientData

char* m_clientData

Contains a pointer to client data for listboxes and choices, if the event was a selection.

4.30.2 wxCommandEvent::m_commandInt

int m_commandInt

Contains an integer identifier corresponding to a listbox, choice or radiobox selection (only if the event was a selection, not a deselection), or a boolean value representing the value of a checkbox.

4.30.3 wxCommandEvent::m_commandString

char* m_commandString

Contains a string corresponding to a listbox or choice selection.

4.30.4 wxCommandEvent::m_extraLong

long m_extraLong

Extra information. If the event comes from a listbox selection, it is a boolean determining whether the event was a selection (TRUE) or a deselection (FALSE). A listbox deselection only occurs for multiple-selection boxes, and in this case the index and string values are indeterminate and the listbox must be examined by the application.

4.30.5 wxCommandEvent::wxCommandEvent

wxCommandEvent(WXTYPE *commandEventType* = 0, int *id* = 0)

Constructor.

4.30.6 wxCommandEvent::Checked

bool Checked()

Returns TRUE or FALSE for a checkbox selection event.

4.30.7 wxCommandEvent::GetClientData

char* GetClientData()

Returns client data pointer for a listbox or choice selection event (not valid for a deselection).

4.30.8 wxCommandEvent::GetExtraLong

long GetExtraLong()

Returns the **m_extraLong** member.

4.30.9 wxCommandEvent::GetInt**int GetInt()**

Returns the **m_commandInt** member.

4.30.10 wxCommandEvent::GetSelection**int GetSelection()**

Returns item index for a listbox or choice selection event (not valid for a deselection).

4.30.11 wxCommandEvent::GetString**char* GetString()**

Returns item string for a listbox or choice selection event (not valid for a deselection).

4.30.12 wxCommandEvent::IsSelection**bool IsSelection()**

For a listbox or choice event, returns TRUE if it is a selection, FALSE if it is a deselection.

4.30.13 wxCommandEvent::SetClientData**void SetClientData(char* *clientData*)**

Sets the client data for this event.

4.30.14 wxCommandEvent::SetExtraLong**void SetExtraLong(int *extraLong*)**

Sets the **m_extraLong** member.

4.30.15 wxCommandEvent::SetInt**void SetInt(int *intCommand*)**

Sets the **m_commandInt** member.

4.30.16 wxCommandEvent::SetString

void SetString(char* *string*)

Sets the `m_commandString` member.

4.31 wxCommandProcessor

`wxCommandProcessor` is a class that maintains a history of `wxCommands`, with undo/redo functionality built-in. Derive a new class from this if you want different behaviour.

Derived from

wxObject (p. 471)

Include files

<wx/docview.h>

See also

wxCommandProcessor overview (p. 938), *wxCommand* (p. 101)

4.31.1 wxCommandProcessor::wxCommandProcessor

wxCommandProcessor(int *maxCommands* = 100)

Constructor.

maxCommands defaults to a rather arbitrary 100, but can be set from 1 to any integer. If your `wxCommand` classes store a lot of data, you may wish to limit the number of commands stored to a smaller number.

4.31.2 wxCommandProcessor::~~wxCommandProcessor

~wxCommandProcessor()

Destructor.

4.31.3 wxCommandProcessor::CanUndo

virtual bool CanUndo()

Returns TRUE if the currently-active command can be undone, FALSE otherwise.

4.31.4 wxCommandProcessor::ClearCommands

virtual void ClearCommands()

Deletes all the commands in the list and sets the current command pointer to NULL.

4.31.5 wxCommandProcessor::Do

virtual bool Do()

Executes (redoes) the current command (the command that has just been undone if any).

4.31.6 wxCommandProcessor::GetCommands

wxList& GetCommands() const

Returns the list of commands.

4.31.7 wxCommandProcessor::GetMaxCommands

int GetMaxCommands() const

Returns the maximum number of commands that the command processor stores.

4.31.8 wxCommandProcessor::GetEditMenu

wxMenu* GetEditMenu() const

Returns the edit menu associated with the command processor.

4.31.9 wxCommandProcessor::Initialize

virtual void Initialize()

Initializes the command processor, setting the current command to the last in the list (if any), and updating the edit menu (if one has been specified).

4.31.10 wxCommandProcessor::SetEditMenu

void SetEditMenu(wxMenu* menu)

Tells the command processor to update the Undo and Redo items on this menu as appropriate. Set this to NULL if the menu is about to be destroyed and command operations may still be performed, or the command processor may try to access an invalid pointer.

4.31.11 **wxCommandProcessor::Submit**

virtual bool Submit(wxCommand *command, bool storeIt = TRUE)

Submits a new command to the command processor. The command processor calls `wxCommand::Do` to execute the command; if it succeeds, the command is stored in the history list, and the associated edit menu (if any) updated appropriately. If it fails, the command is deleted immediately. Once `Submit` has been called, the passed command should not be deleted directly by the application.

storeIt indicates whether the successful command should be stored in the history list.

4.31.12 **wxCommandProcessor::Undo**

virtual bool Undo()

Undoes the command just executed.

4.32 **wxCondition**

TODO

[Derived from](#)

None.

[Include files](#)

<wx/thread.h>

[See also](#)

wxThread (p. 736), *wxMutex* (p. 459)

4.32.1 **wxCondition::wxCondition**

wxCondition()

Default constructor.

4.32.2 wxCondition::~~wxCondition

~wxCondition()

Destroys the wxCondition object.

4.32.3 wxCondition::Broadcast

void Broadcast()

Broadcasts to all waiting objects.

4.32.4 wxCondition::Signal

void Signal()

Signals the object.

4.32.5 wxCondition::Wait

void Wait(wxMutex& *mutex*)

Waits indefinitely.

bool Wait(wxMutex& *mutex*, unsigned long *sec*, unsigned long *nsec*)

Waits until a signal is raised or the timeout has elapsed.

Parameters

mutex
wxMutex object.

sec
Timeout in seconds

nsec
Timeout nanoseconds component (added to *sec*).

Return value

The second form returns if the signal was raised, or FALSE if there was a timeout.

4.33 wxConfigBase

`wxConfigBase` class defines the basic interface of all config classes. It can not be used by itself (it's an abstract base class) and you'll always use one of its derivations: `wxIniConfig`, `wxFileConfig`, `wxRegConfig` or any other.

However, usually you don't even need to know the precise nature of the class you're working with but you would just use the `wxConfigBase` methods. This allows you to write the same code regardless of whether you're working with the registry under Win32 or text-based config files under Unix (or even Windows 3.1 .INI files if you're really unlucky). To make writing the portable code even easier, `wxWindows` provides a typedef `wxConfig` which is mapped onto the native `wxConfigBase` implementation on the given platform: i.e. `wxRegConfig` under Win32, `wxIniConfig` under Win16 and `wxFileConfig` otherwise.

See *config overview* (p. 907) for the descriptions of all features of this class.

Derived from

No base class

Include files

<wx/config.h> (to let `wxWindows` choose a `wxConfig` class for your platform)
<wx/confbase.h> (base config class)
<wx/fileconf.h> (`wxFileConfig` class)
<wx/msw/regconf.h> (`wxRegConfig` class)
<wx/msw/iniconf.h> (`wxIniConfig` class)

Example

Here is how you would typically use this class:

```
// using wxConfig instead of writing wxFileConfig or wxRegConfig
enhances
// portability of the code
wxConfig *config = new wxConfig("MyAppName");

wxString str;
if ( config->Read("LastPrompt", &str) ) {
    // last prompt was found in the config file/registry and its value
    is now
    // in str
    ...
}
else {
    // no last prompt...
}

// another example: using default values and the full path instead of
just
// key name: if the key is not found , the value 17 is returned
long value = config->Read("/LastRun/CalculatedValues/MaxValue", -1);
...
...
...
// at the end of the program we would save everything back
```



```
config->Write("LastPrompt", str);  
config->Write("/LastRun/CalculatedValues/MaxValue", value);  
  
// the changes will be written back automatically  
delete config;
```

This basic example, of course, doesn't show all `wxConfig` features, such as enumerating, testing for existence and deleting the entries and groups of entries in the config file, its abilities to automatically store the default values or expand the environment variables on the fly. However, the main idea is that using this class is easy and that it should normally do what you expect it to.

NB: in the documentation of this class, the words "config file" also mean "registry hive" for `wxRegConfig` and, generally speaking, might mean any physical storage where a `wxConfigBase`-derived class stores its data.

4.33.1 Static functions

These functions deal with the "default" config object. Although its usage is not at all mandatory it may be convenient to use a global config object instead of creating and deleting the local config objects each time you need one (especially because creating a `wxFileConfig` object might be a time consuming operation). In this case, you may create this global config object in the very start of the program and *Set()* it as the default. Then, from anywhere in your program, you may access it using the *Get()* function. Of course, you should delete it on the program termination (otherwise, not only a memory leak will result, but even more importantly the changes won't be written back!).

As it happens, you may even further simplify the procedure described above: you may forget about calling *Set()*. When *Get()* is called and there is no current object, it will create one using *Create()* function. To disable this behaviour *DontCreateOnDemand()* is provided.

Set (p. 124)

Get (p. 119)

Create (p. 118)

DontCreateOnDemand (p. 119)

4.33.2 Constructor and destructor

wxConfigBase (p. 117)

~wxConfigBase (p. 118)

4.33.3 Path management

As explained in *config overview* (p. 907), the config classes support a file system-like hierarchy of keys (files) and groups (directories). As in the file system case, to specify a

key in the config class you must use a path to it. Config classes also support the notion of the current group, which makes it possible to use the relative paths. To clarify all this, here is an example (it's only for the sake of demonstration, it doesn't do anything sensible!):

```
wxConfig *config = new wxConfig("FooBarApp");

// right now the current path is '/'
conf->Write("RootEntry", 1);

// go to some other place: if the group(s) don't exist, they will be
// created
conf->SetPath("/Group/Subgroup");

// create an entry in subgroup
conf->Write("SubgroupEntry", 3);

// '..' is understood
conf->Write("../GroupEntry", 2);
conf->SetPath("../");

wxASSERT( conf->Read("Subgroup/SubgroupEntry", 0l) == 3 );

// use absolute path: it's allowed, too
wxASSERT( conf->Read("/RootEntry", 0l) == 1 );
```

Warning: it's probably a good idea to always restore the path to its old value on function exit:

```
void foo(wxConfigBase *config)
{
    wxString strOldPath = config->GetPath();

    config->SetPath("/Foo/Data");
    ...

    config->SetPath(strOldPath);
}
```

because otherwise the assert in the following example will surely fail (we suppose here that *foo()* function is the same as above except that it doesn't save and restore the path):

```
void bar(wxConfigBase *config)
{
    config->Write("Test", 17);

    foo(config);

    // we're reading "/Foo/Data/Test" here! -1 will probably be
    // returned...
    wxASSERT( config->Read("Test", -1) == 17 );
}
```

Finally, the path separator in `wxConfigBase` and derived classes is always '/', regardless of the platform (i.e. it's **not** '\\' under Windows).

SetPath (p. 124)

GetPath (p. 121)

4.33.4 Enumeration

The functions in this section allow to enumerate all entries and groups in the config file. All functions here return FALSE when there are no more items.

You must pass the same index to *GetNext* and *GetFirst* (don't modify it). Please note that it's **not** the index of the current item (you will have some great surprises with *wxRegConfig* if you assume this) and you shouldn't even look at it: it's just a "cookie" which stores the state of the enumeration. It can't be stored inside the class because it would prevent you from running several enumerations simultaneously, that's why you must pass it explicitly.

Having said all this, enumerating the config entries/groups is very simple:

```
wxArrayString aNames;

// enumeration variables
wxString str;
long dummy;

// first enum all entries
bool bCont = config->GetFirstEntry(str, dummy);
while ( bCont ) {
    aNames.Add(str);

    bCont = GetConfig()->GetNextEntry(str, dummy);
}

... we have all entry names in aNames...

// now all groups...
bCont = GetConfig()->GetFirstGroup(str, dummy);
while ( bCont ) {
    aNames.Add(str);

    bCont = GetConfig()->GetNextGroup(str, dummy);
}

... we have all group (and entry) names in aNames...
```

There are also functions to get the number of entries/subgroups without actually enumerating them, but you will probably never need them.

GetFirstGroup (p. 120)

GetNextGroup (p. 120)

GetFirstEntry (p. 120)

GetNextEntry (p. 120)

GetNumberOfEntries (p. 121)

GetNumberOfGroups (p. 121)

4.33.5 Tests of existence

HasGroup (p. 121)

HasEntry (p. 121)

Exists (p. 119)

4.33.6 Miscellaneous accessors

SetAppName (p. 124)

GetAppName (p. 120)

SetVendorName (p. 124)

GetVendorName (p. 121)

4.33.7 Key access

These function are the core of *wxConfigBase* class: they allow you to read and write config file data. All *Read* function take a default value which will be returned if the specified key is not found in the config file.

Currently, only two types of data are supported: string and long (but it might change in the near future). To work with other types: for *int* or *bool* you can work with function taking/returning *long* and just use the casts. Better yet, just use *long* for all variables which you're going to save in the config file: chances are that `sizeof(bool) == sizeof(int) == sizeof(long)` anyhow on your system. For *float*, *double* and, in general, any other type you'd have to translate them to/from string representation and use string functions.

Try not to read long values into string variables and vice versa: although it just might work with *wxFileConfig*, you will get a system error with *wxRegConfig* because in the Windows registry the different types of entries are indeed used.

Final remark: the *szKey* parameter for all these functions can contain an arbitrary path (either relative or absolute), not just the key name.

Read (p. 122)

Write (p. 125)

Flush (p. 119)

4.33.8 Rename entries/groups

The functions in this section allow to rename entries or subgroups of the current group. They will return *FALSE* on error. typically because either the entry/group with the original name doesn't exist, because the entry/group with the new name already exists or because the function is not supported in this *wxConfig* implementation.

RenameEntry (p. 123)

RenameGroup (p. 123)

4.33.9 Delete entries/groups

The functions in this section delete entries and/or groups of entries from the config file. *DeleteAll()* is especially useful if you want to erase all traces of your program presence: for example, when you uninstall it.

DeleteEntry (p. 119)

DeleteGroup (p. 119)

DeleteAll (p. 119)

4.33.10 Options

Some aspects of *wxConfigBase* behaviour can be changed during run-time. The first of them is the expansion of environment variables in the string values read from the config file: for example, if you have the following in your config file:

```
# config file for my program
UserData = $HOME/data

# the following syntax is valid only under Windows
UserData = %windir%\data.dat
```

the call to `config->Read("UserData")` will return something like `"/home/zeitlin/data"` if you're lucky enough to run a Linux system ;-)

Although this feature is very useful, it may be annoying if you read a value which contains '\$' or '%' symbols (% is used for environment variables expansion under Windows) which are not used for environment variable expansion. In this situation you may call `SetExpandEnvVars(FALSE)` just before reading this value and `SetExpandEnvVars(TRUE)` just after. Another solution would be to prefix the offending symbols with a backslash.

The following functions control this option:

IsExpandingEnvVars (p. 121)

SetExpandingEnvVars (p. 124)

SetRecordDefaults (p. 124)

IsRecordingDefaults (p. 122)

4.33.11 *wxConfigBase::wxConfigBase*

wxConfigBase(const *wxString*& *appName* = *wxEmptyString*, const *wxString*& *vendorName* = *wxEmptyString*, const *wxString*& *localFilename* = *wxEmptyString*, const *wxString*& *globalFilename* = *wxEmptyString*, long *style* = 0)

This is the default and only constructor of the `wxConfigBase` class, and derived classes.

Parameters

appName

The application name. If this is empty, the class will normally use `wxApp::GetAppName` (p. 8) to set it. The application name is used in the registry key on Windows, and can be used to deduce the local filename parameter if that is missing.

vendorName

The vendor name. If this is empty, it is assumed that no vendor name is wanted, if this is optional for the current config class. The vendor name is appended to the application name for `wxRegConfig`.

localFilename

Some config classes require a local filename. If this is not present, but required, the application name will be used instead.

globalFilename

Some config classes require a global filename. If this is not present, but required, the application name will be used instead.

style

Can be one of `wxCONFIG_USE_LOCAL_FILE` and `wxCONFIG_USE_GLOBAL_FILE`. The style interpretation depends on the config class and is ignored by some. For `wxFileConfig`, these styles determine whether a local or global config file is created or used. If the flag is present but the parameter is empty, the parameter will be set to a default. If the parameter is present but the style flag not, the relevant flag will be added to the style.

Remarks

By default, environment variable expansion is on and recording defaults is off.

4.33.12 `wxConfigBase::~~wxConfigBase`

`~wxConfigBase()`

Empty but ensures that dtor of all derived classes is virtual.

4.33.13 `wxConfigBase::Create`

`static wxConfigBase * Create()`

Create a new config object: this function will create the "best" implementation of `wxConfig` available for the current platform, see comments near the definition of `wxCONFIG_WIN32_NATIVE` for details. It returns the created object and also sets it as the current one.

4.33.14 wxConfigBase::DontCreateOnDemand**void DontCreateOnDemand()**

Calling this function will prevent *Get()* from automatically creating a new config object if the current one is NULL. It might be useful to call it near the program end to prevent new config object "accidental" creation.

4.33.15 wxConfigBase::DeleteAll**bool DeleteAll()**

Delete the whole underlying object (disk file, registry key, ...). Primarily for use by desinstallation routine.

4.33.16 wxConfigBase::DeleteEntry**bool DeleteEntry(const wxString& key, bool bDeleteGroupIfEmpty = TRUE)**

Deletes the specified entry and the group it belongs to if it was the last key in it and the second parameter is true.

4.33.17 wxConfigBase::DeleteGroup**bool DeleteGroup(const wxString& key)**

Delete the group (with all subgroups)

4.33.18 wxConfigBase::Exists**bool Exists(wxString& strName) const**

returns TRUE if either a group or an entry with a given name exists

4.33.19 wxConfigBase::Flush**bool Flush(bool bCurrentOnly = FALSE)**

permanently writes all changes (otherwise, they're only written from object's destructor)

4.33.20 wxConfigBase::Get

wxConfigBase * Get()

Get the current config object. If there is no current object, creates one (using *Create*) unless DontCreateOnDemand was called previously.

4.33.21 wxConfigBase::GetAppName**wxString GetAppName() const**

Returns the application name.

4.33.22 wxConfigBase::GetFirstGroup**bool GetFirstGroup(wxString& str, long&index) const**

Gets the first group.

wxPython note:

The wxPython version of this method returns a 3-tuple consisting of the continue flag, the value string, and the index for the next call.

4.33.23 wxConfigBase::GetFirstEntry**bool GetFirstEntry(wxString& str, long&index) const**

Gets the first entry.

wxPython note:

The wxPython version of this method returns a 3-tuple consisting of the continue flag, the value string, and the index for the next call.

4.33.24 wxConfigBase::GetNextGroup**bool GetNextGroup(wxString& str, long&index) const**

Gets the next group.

wxPython note:

The wxPython version of this method returns a 3-tuple consisting of the continue flag, the value string, and the index for the next call.

4.33.25 wxConfigBase::GetNextEntry**bool GetNextEntry(wxString& str, long&index) const**

Gets the next entry.

wxPython note:

The wxPython version of this method returns a 3-tuple consisting of the continue flag, the value string, and the index for the next call.

4.33.26 wxConfigBase::GetNumberOfEntries

uint GetNumberOfEntries(bool bRecursive = FALSE) const

4.33.27 wxConfigBase::GetNumberOfGroups

uint GetNumberOfGroups(bool bRecursive = FALSE) const

Get number of entries/subgroups in the current group, with or without its subgroups.

4.33.28 wxConfigBase::GetPath

const wxString& GetPath() const

Retrieve the current path (always as absolute path).

4.33.29 wxConfigBase::GetVendorName

wxString GetVendorName() const

Returns the vendor name.

4.33.30 wxConfigBase::HasEntry

bool HasEntry(wxString& strName) const

returns TRUE if the entry by this name exists

4.33.31 wxConfigBase::HasGroup

bool HasGroup(const wxString& strName) const

returns TRUE if the group by this name exists

4.33.32 wxConfigBase::IsExpandingEnvVars

bool IsExpandingEnvVars() const

Returns TRUE if we are expanding environment variables in key values.

4.33.33 wxConfigBase::IsRecordingDefaults

bool IsRecordingDefaults() const

Returns TRUE if we are writing defaults back to the config file.

4.33.34 wxConfigBase::Read

bool Read(const wxString& key, wxString*str) const

Read a string from the key, returning TRUE if the value was read. If the key was not found, *str* is not changed.

bool Read(const wxString& key, wxString*str, const wxString& defaultVal) const

Read a string from the key. The default value is returned if the key was not found.

Returns TRUE if value was really read, FALSE if the default was used.

wxString Read(const wxString& key, const wxString& defaultVal) const

Another version of *Read()*, returning the string value directly.

bool Read(const wxString& key, long* l) const

Reads a long value, returning TRUE if the value was found. If the value was not found, *l* is not changed.

bool Read(const wxString& key, long* l, long defaultVal) const

Reads a long value, returning TRUE if the value was found. If the value was not found, *defaultVal* is used instead.

long Read(const wxString& key, long defaultVal) const

Reads a long value from the key and returns it. *defaultVal* is returned if the key is not found.

NB: writing

```
conf->Read("key", 0);
```

won't work because the call is ambiguous: compiler can not choose between two *Read* functions. Instead, write:

```
conf->Read("key", 01);
```

bool Read(const wxString& key, double* d) const

Reads a double value, returning TRUE if the value was found. If the value was not found, *d* is not changed.

bool Read(const wxString& key, double* d, double defaultVal) const

Reads a double value, returning TRUE if the value was found. If the value was not found, *defaultVal* is used instead.

bool Read(const wxString& key, bool* b) const

Reads a bool value, returning TRUE if the value was found. If the value was not found, *b* is not changed.

bool Read(const wxString& key, bool* d, bool defaultVal) const

Reads a bool value, returning TRUE if the value was found. If the value was not found, *defaultVal* is used instead.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

Read(key, default="")	Returns a string.
ReadInt(key, default=0)	Returns an int.
ReadFloat(key, default=0.0)	Returns a floating point number.

4.33.35 wxConfigBase::RenameEntry**bool RenameEntry(const wxString& oldName, const wxString& newName)**

Renames an entry in the current group. The entries names (both the old and the new one) shouldn't contain backslashes, i.e. only simple names and not arbitrary paths are accepted by this function.

Returns FALSE if the *oldName* doesn't exist or if *newName* already exists.

4.33.36 wxConfigBase::RenameGroup**bool RenameGroup(const wxString& oldName, const wxString& newName)**

Renames a subgroup of the current group. The subgroup names (both the old and the new one) shouldn't contain backslashes, i.e. only simple names and not arbitrary paths are accepted by this function.

Returns FALSE if the *oldName* doesn't exist or if *newName* already exists.

4.33.37 wxConfigBase::Set

wxConfigBase * Set(wxConfigBase *pConfig)

Sets the config object as the current one, returns the pointer to the previous current object (both the parameter and returned value may be NULL)

4.33.38 wxConfigBase::SetAppName

void SetAppName(const wxString& appName)

Sets the application name.

4.33.39 wxConfigBase::SetExpandingEnvVars

void SetExpandEnvVars (bool bDoIt = TRUE)

Determine whether we wish to expand environment variables in key values.

4.33.40 wxConfigBase::SetPath

void SetPath(const wxString& strPath)

Set current path: if the first character is '/', it's the absolute path, otherwise it's a relative path. '..' is supported. If the strPath doesn't exist it is created.

4.33.41 wxConfigBase::SetRecordDefaults

void SetRecordDefaults(bool bDoIt = TRUE)

Sets whether defaults are written back to the config file.

If on (default is off) all default values are written back to the config file. This allows the user to see what config options may be changed and is probably useful only for wxFileConfig.

4.33.42 wxConfigBase::SetVendorName

void SetVendorName(const wxString& vendorName)

Sets the vendor name.

4.33.43 wxConfigBase::Write

bool Write(const wxString& key, const wxString& value)

bool Write(const wxString& key, long value)

bool Write(const wxString& key, double value)

bool Write(const wxString& key, bool value)

These functions write the specified value to the config file and return TRUE on success.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

Write(key, value)	Writes a string.
WriteInt(key, value)	Writes an int.
WriteFloat(key, value)	Writes a floating point number.

4.34 wxControl

This is the base class for a control or 'widget'.

A control is generally a small window which processes user input and/or displays one or more item of data.

Derived from

wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/control.h>

See also

wxValidator (p. 781)

4.34.1 wxControl::Command

void Command(wxCommandEvent& event)

Simulates the effect of the user issuing a command to the item. See *wxCommandEvent* (p. 103).

4.34.2 **wxControl::GetLabel**

wxString& GetLabel()

Returns the control's text.

4.34.3 **wxControl::SetLabel**

void SetLabel(const wxString& label)

Sets the item's text.

4.35 **wxCriticalSection**

A critical section object is used exactly for the same purpose as *mutexes* (p. 459). The only difference is that under Windows platform critical sections are only visible inside one process, while mutexes may be shared between processes, so using critical sections is slightly more efficient. The terminology is also slightly different: mutex may be locked (or acquired) and unlocked (or released) while critical section is entered and left by the program.

Finally, you should try to use *wxCriticalSectionLocker* (p. 127) class whenever possible instead of directly using *wxCriticalSection* for the same reasons *wxMutexLocker* (p. 462) is preferable to *wxMutex* (p. 459) - please see *wxMutex* for an example.

Derived from

None.

Include files

<wx/thread.h>

See also

wxThread (p. 736), *wxCondition* (p. 110), *wxMutexLocker* (p. 462), *wxCriticalSection* (p. 126)

4.35.1 **wxCriticalSection::wxCriticalSection**

wxCriticalSection()

Default constructor initializes critical section object.

4.35.2 `wxCriticalSection::~~wxCriticalSection`

`~wxCriticalSection()`

Destructor frees the resources.

4.35.3 `wxCriticalSection::Enter`

`void Enter()`

Enter the critical section (same as locking a mutex). There is no error return for this function. After entering the critical section protecting some global data the thread running in critical section may safely use/modify it.

4.35.4 `wxCriticalSection::Leave`

`void Leave()`

Leave the critical section allowing other threads use the global data protected by it. There is no error return for this function.

4.36 `wxCriticalSectionLocker`

This is a small helper class to be used with *wxCriticalSection* (p. 126) objects. A `wxCriticalSectionLocker` enters the critical section in the constructor and leaves it in the destructor making it much more difficult to forget to leave a critical section (which, in general, will lead to serious and difficult to debug problems).

Derived from

None.

Include files

`<wx/thread.h>`

See also

wxCriticalSection (p. 126), *wxMutexLocker* (p. 462)

4.36.1 `wxCriticalSectionLocker::wxCriticalSectionLocker`

wxCriticalSectionLocker(wxCriticalSection *criticalsection)

Constructs a wxCriticalSectionLocker object associated with given criticalsection which must be non NULL and enters it.

4.36.2 wxCriticalSectionLocker::~~wxCriticalSectionLocker**~wxCriticalSectionLocker()**

Destuctor leaves the criticalsection.

4.37 wxCursor

A cursor is a small bitmap usually used for denoting where the mouse pointer is, with a picture that might indicate the interpretation of a mouse click. As with icons, cursors in X and MS Windows are created in a different manner. Therefore, separate cursors will be created for the different environments. Platform-specific methods for creating a **wxCursor** object are catered for, and this is an occasion where conditional compilation will probably be required (see *wx/con* (p. 318) for an example).

A single cursor object may be used in many windows (any subwindow type). The wxWindows convention is to set the cursor for a window, as in X, rather than to set it globally as in MS Windows, although a global `::wxSetCursor` (p. 857) is also available for MS Windows use.

Derived from

wxBitmap (p. 39)

wxGDIObject (p. 295)

wxObject (p. 471)

Include files

<wx/cursor.h>

Predefined objects

Objects:

wxNullCursor

Pointers:

wxSTANDARD_CURSOR
wxHOURLASS_CURSOR
wxCROSS_CURSOR

See also

wxBitmap (p. 39), *wxIcon* (p. 318), *wxWindow::SetCursor* (p. 834), *::wxSetCursor* (p. 857)

4.37.1 **wxCursor::wxCursor**

wxCursor()

Default constructor.

wxCursor(const char bits[], int width, int height, int hotSpotX=-1, int hotSpotY=-1, const char maskBits[]=NULL)

Constructs a cursor by passing an array of bits (Motif and Xt only). *maskBits* is used only under Motif.

If either *hotSpotX* or *hotSpotY* is -1, the hotspot will be the centre of the cursor image (Motif only).

wxCursor(const wxString& cursorName, long type, int hotSpotX=0, int hotSpotY=0)

Constructs a cursor by passing a string resource name or filename.

hotSpotX and *hotSpotY* are currently only used under Windows when loading from an icon file, to specify the cursor hotspot relative to the top left of the image.

wxCursor(int cursorId)

Constructs a cursor using a cursor identifier.

wxCursor(const wxCursor& cursor)

Copy constructor. This uses reference counting so is a cheap operation.

Parameters

bits

An array of bits.

maskBits

Bits for a mask bitmap.

width

Cursor width.

height

Cursor height.

hotSpotX

Hotspot x coordinate.

hotSpotY

Hotspot y coordinate.

type

Icon type to load. Under Motif, *type* defaults to **wxBITMAP_TYPE_XBM**. Under Windows, it defaults to **wxBITMAP_TYPE_CUR_RESOURCE**.

Under X, the permitted cursor types are:

wxBITMAP_TYPE_XBM Load an X bitmap file.

Under Windows, the permitted types are:

wxBITMAP_TYPE_CUR Load a cursor from a .cur cursor file (only if **USE_RESOURCE_LOADING_IN_MSW** is enabled in setup.h).

wxBITMAP_TYPE_CUR_RESOURCE Load a Windows resource (as specified in the .rc file).

wxBITMAP_TYPE_ICO Load a cursor from a .ico icon file (only if **USE_RESOURCE_LOADING_IN_MSW** is enabled in setup.h). Specify *hotSpotX* and *hotSpotY*.

cursorId

A stock cursor identifier. May be one of:

wxCURSOR_ARROW	A standard arrow cursor.
wxCURSOR_BULLSEYE	Bullseye cursor.
wxCURSOR_CHAR	Rectangular character cursor.
wxCURSOR_CROSS	A cross cursor.
wxCURSOR_HAND	A hand cursor.
wxCURSOR_IBEAM	An I-beam cursor (vertical line).
wxCURSOR_LEFT_BUTTON	Represents a mouse with the left button depressed.
wxCURSOR_MAGNIFIER	A magnifier icon.
wxCURSOR_MIDDLE_BUTTON	Represents a mouse with the middle button depressed.
wxCURSOR_NO_ENTRY	A no-entry sign cursor.
wxCURSOR_PAINT_BRUSH	A paintbrush cursor.
wxCURSOR_PENCIL	A pencil cursor.
wxCURSOR_POINT_LEFT	A cursor that points left.
wxCURSOR_POINT_RIGHT	A cursor that points right.
wxCURSOR_QUESTION_ARROW	An arrow and question mark.
wxCURSOR_RIGHT_BUTTON	Represents a mouse with the right button depressed.
wxCURSOR_SIZENESW	A sizing cursor pointing NE-SW.
wxCURSOR_SIZENS	A sizing cursor pointing N-S.
wxCURSOR_SIZENWSE	A sizing cursor pointing NW-SE.

wxCURSOR_SIZEWE	A sizing cursor pointing W-E.
wxCURSOR_SIZING	A general sizing cursor.
wxCURSOR_SPRAYCAN	A spraycan cursor.
wxCURSOR_WAIT	A wait cursor.
wxCURSOR_WATCH	A watch cursor.

Note that not all cursors are available on all platforms.

cursor

Pointer or reference to a cursor to copy.

wxPython note:

Constructors supported by wxPython are:

wxCursor(name, flags, hotSpotX=0, hotSpotY=0)	Constructs a cursor from a filename
wxStockCursor(id)	Constructs a stock cursor

4.37.2 wxCursor::~~wxCursor

~wxCursor()

Destroys the cursor. A cursor can be reused for more than one window, and does not get destroyed when the window is destroyed. wxWindows destroys all cursors on application exit, although it's best to clean them up explicitly.

4.37.3 wxCursor::Ok

bool Ok() const

Returns TRUE if cursor data is present.

4.37.4 wxCursor::operator =

wxCursor& operator =(const wxCursor& cursor)

Assignment operator, using reference counting. Returns a reference to 'this'.

4.37.5 wxCursor::operator ==

bool operator ==(const wxCursor& cursor)

Equality operator. Two cursors are equal if they contain pointers to the same underlying cursor data. It does not compare each attribute, so two independently-created cursors

using the same parameters will fail the test.

4.37.6 wxCursor::operator !=

bool operator !=(const wxCursor& *cursor*)

Inequality operator. Two cursors are not equal if they contain pointers to different underlying cursor data. It does not compare each attribute.

4.38 wxDatabase

Every database object represents an ODBC connection. The connection may be closed and reopened.

Derived from

wxObject (p. 471)

Include files

<wx/odbc.h>

See also

wxDatabase overview (p. 923), *wxRecordSet* (p. 550)

4.38.1 wxDatabase::wxDatabase

wxDatabase()

Constructor. The constructor of the first wxDatabase instance of an application initializes the ODBC manager.

4.38.2 wxDatabase::~~wxDatabase

~wxDatabase()

Destructor. Resets and destroys any associated wxRecordSet instances.

The destructor of the last wxDatabase instance will deinitialize the ODBC manager.

4.38.3 wxDatabase::BeginTrans

bool BeginTrans()

Not implemented.

4.38.4 wxDatabase::Cancel

void Cancel()

Not implemented.

4.38.5 wxDatabase::CanTransact

bool CanTransact()

Not implemented.

4.38.6 wxDatabase::CanUpdate

bool CanUpdate()

Not implemented.

4.38.7 wxDatabase::Close

bool Close()

Resets the statement handles of any associated wxRecordSet objects, and disconnects from the current data source.

4.38.8 wxDatabase::CommitTrans

bool CommitTrans()

Commits previous transactions. Not implemented.

4.38.9 wxDatabase::ErrorOccured

bool ErrorOccured()

Returns TRUE if the last action caused an error.

4.38.10 wxDatabase::ErrorSnapshot

void ErrorSnapshot(HSTMT *statement* = SQL_NULL_HSTMT)

This function will be called whenever an ODBC error occurred. It stores the error related information returned by ODBC. If a statement handle of the concerning ODBC action is available it should be passed to the function.

4.38.11 wxDatabase::GetDatabaseName

wxString GetDatabaseName()

Returns the name of the database associated with the current connection.

4.38.12 wxDatabase::GetDataSource

wxString GetDataSource()

Returns the name of the connected data source.

4.38.13 wxDatabase::GetErrorClass

wxString GetErrorClass()

Returns the error class of the last error. The error class consists of five characters where the first two characters contain the class and the other three characters contain the subclass of the ODBC error. See ODBC documentation for further details.

4.38.14 wxDatabase::GetErrorCode

wxRETCODE GetErrorCode()

Returns the error code of the last ODBC function call. This will be one of:

SQL_ERROR	General error.
SQL_INVALID_HANDLE	An invalid handle was passed to an ODBC function.
SQL_NEED_DATA	ODBC expected some data.
SQL_NO_DATA_FOUND	No data was found by this ODBC call.
SQL_SUCCESS	The call was successful.
SQL_SUCCESS_WITH_INFO	The call was successful, but further information can be obtained from the ODBC manager.

4.38.15 wxDatabase::GetErrorMessage

wxString GetErrorMessage()

Returns the last error message returned by the ODBC manager.

4.38.16 wxDatabase::GetErrorNumber**long GetErrorNumber()**

Returns the last native error. A native error is an ODBC driver dependent error number.

4.38.17 wxDatabase::GetHDBC**HDBC GetHDBC()**

Returns the current ODBC database handle.

4.38.18 wxDatabase::GetHENV**HENV GetHENV()**

Returns the ODBC environment handle.

4.38.19 wxDatabase::GetInfo**bool GetInfo(long infoType, long *buf)****bool GetInfo(long infoType, const wxString& buf, int bufSize=-1)**

Returns requested information. The return value is TRUE if successful, FALSE otherwise.

infoType is an ODBC identifier specifying the type of information to be returned.

buf is a character or long integer pointer to storage which must be allocated by the application, and which will contain the information if the function is successful.

bufSize is the size of the character buffer. A value of -1 indicates that the size should be computed by the GetInfo function.

4.38.20 wxDatabase::GetPassword**wxString GetPassword()**

Returns the password of the current user.

4.38.21 wxDatabase::GetUsername

wxString GetUsername()

Returns the current username.

4.38.22 wxDatabase::GetODBCVersionFloat**float GetODBCVersionFloat(bool implementation=TRUE)**

Returns the version of ODBC in floating point format, e.g. 2.50.

implementation should be TRUE to get the DLL version, or FALSE to get the version defined in the `sql.h` header file.

This function can return the value 0.0 if the header version number is not defined (for early versions of ODBC).

4.38.23 wxDatabase::GetODBCVersionString**wxString GetODBCVersionString(bool implementation=TRUE)**

Returns the version of ODBC in string format, e.g. "02.50".

implementation should be TRUE to get the DLL version, or FALSE to get the version defined in the `sql.h` header file.

This function can return the value "00.00" if the header version number is not defined (for early versions of ODBC).

4.38.24 wxDatabase::InWaitForDataSource**bool InWaitForDataSource()**

Not implemented.

4.38.25 wxDatabase::IsOpen**bool IsOpen()**

Returns TRUE if a connection is open.

4.38.26 wxDatabase::Open

bool Open(const wxString& datasource, bool exclusive = FALSE, bool readOnly = TRUE, const wxString& username = "ODBC", const wxString& password = "")

Connect to a data source. *datasource* contains the name of the ODBC data source. The parameters *exclusive* and *readOnly* are not used.

4.38.27 wxDatabase::OnSetOptions

void OnSetOptions(wxRecordSet *recordSet)

Not implemented.

4.38.28 wxDatabase::OnWaitForDataSource

void OnWaitForDataSource(bool stillExecuting)

Not implemented.

4.38.29 wxDatabase::RollbackTrans

bool RollbackTrans()

Sends a rollback to the ODBC driver. Not implemented.

4.38.30 wxDatabase::SetDataSource

void SetDataSource(const wxString& s)

Sets the name of the data source. Not implemented.

4.38.31 wxDatabase::SetLoginTimeout

void SetLoginTimeout(long seconds)

Sets the time to wait for an user login. Not implemented.

4.38.32 wxDatabase::SetPassword

void SetPassword(const wxString& s)

Sets the password of the current user. Not implemented.

4.38.33 wxDatabase::SetSynchronousMode

void SetSynchronousMode(bool synchronous)

Toggles between synchronous and asynchronous mode. Currently only synchronous mode is supported, so this function has no effect.

4.38.34 *wxDatabase::SetQueryTimeout*

void SetQueryTimeout(long seconds)

Sets the time to wait for a response to a query. Not implemented.

4.38.35 *wxDatabase::SetUsername*

void SetUsername(const wxString& s)

Sets the name of the current user. Not implemented.

4.39 wxDataObject

A *wxDataObject* represents data that can be copied to or from the clipboard, or dragged and dropped.

There are several predefined data object classes, such as *wxFileDataObject* (p. 247), *wxTextDataObject* (p. 723), and *wxBitmapDataObject* (p. 59) which can be used without change or can be altered (by deriving a new class from them) in order to deliver data and data size on-demand. There is no need to ever use *wxDataObject* itself or derive directly from it.

You may also derive your own data object classes from *wxPrivateDataObject* (p. 523) for user-defined types. The format of user-defined data is given as mime-type string literal, such as "application/word" or "image/png". These strings are used as they are under Unix (so far only GTK) to identify a format and are translated into their Windows equivalent under Win32 (using the OLE *IDataObject* for data exchange to and from the clipboard and for Drag'n'Drop). Note that the format string translation under Windows is not yet finished.

As mentioned above, data may be placed into the *wxClipboard* (p. 82) or a *wxDropSource* (p. 216) instance either directly or on-demand. As long as only one format is offered, putting data directly into the clipboard may be sufficient. But imagine that you paste a large piece of text to the clipboard and offer it in "text/plain", "text/rtf", "text/html", "application/word" and your own format for internal use - here offering data on-demand is required to minimize memory consumption. This would generally get implemented using a central object that contains clipboard information in the format with the maximum of information. Note that neither the GTK data transfer mechanisms for the clipboard and Drag'n'Drop nor the OLE data transfer copies any data until another application actually requests the data. This is in contrast to the "feel" offered to the user of a program who would normally think that the data resides in the clipboard after having pressed "Copy" - in reality it is only declared to be available.

Let's assume that you have written an HTML editor and want it to paste contents in the

formats "text/plain" and "text/html" to the clipboard. For offering data on-demand in "text/plain" you would derive your class from *wxTextDataObject* (p. 723) and for offering data on-demand in "text/html" you would derive your own class from *wxPrivateDataObject* (p. 523) and set its ID string identifying the format to "text/html" using *wxPrivateDataObject::SetId* (p. 524). In your two derived classes you'd then have a pointer or reference to the central data container and you'd override the methods returning the size of the available data and the *WriteData()* methods in both classes.

Derived from

wxObject (p. 471)

Include files

<wx/dataobj.h>

See also

wxFileDataObject (p. 247), *wxTextDataObject* (p. 723), *wxBitmapDataObject* (p. 59), *wxPrivateDataObject* (p. 523), *Drag and drop overview* (p. 975), *wxDropTarget* (p. 218), *wxDropSource* (p. 216), *wxTextDropTarget* (p. 726), *wxFileDropTarget* (p. 252)

4.39.1 wxDataObject::wxDataObject

wxDataObject()

Constructor.

4.39.2 wxDataObject::~~wxDataObject

~wxDataObject()

Destructor.

4.39.3 wxDataObject::WriteData

virtual void WriteData(void *dest) const

Write the data owned by this class to *dest*. This method is a pure virtual function and must be overridden.

4.39.4 wxDataObject::GetSize

virtual size_t GetSize() const

Returns the data size. This method is a pure virtual function and must be overridden.

4.40 wxDataInputStream

This class provides functions that read data types in a portable way. So, a file written by an Intel processor can be read by a Sparc or anything else.

Derived from

wxFilterInputStream (p. 262)

wxInputStream (p. 346)

wxStreamBase (p. 646)

Include files

<wx/datstrm.h>

4.40.1 wxDataInputStream::wxDataInputStream

wxDataInputStream(wxInputStream& *stream*)

Constructs a datastream object from an input stream. Only read methods will be available.

Parameters

stream

The input stream.

4.40.2 wxDataInputStream::~~wxDataInputStream

~wxDataInputStream()

Destroys the wxDataInputStream object.

4.40.3 wxDataInputStream::Read8

unsigned char Read8()

Reads a single byte from the stream.

4.40.4 wxDataInputStream::Read16

unsigned short Read16()

Reads a 16 bit integer from the stream.

4.40.5 wxDataInputStream::Read32**unsigned long Read32()**

Reads a 32 bit integer from the stream.

4.40.6 wxDataInputStream::ReadDouble**double ReadDouble()**

Reads a double (IEEE encoded) from the stream.

4.40.7 wxDataInputStream::ReadLine**wxString wxDataInputStream::ReadLine()**

Reads a line from the stream. A line is a string which ends with
n or
r
n.

4.40.8 wxDataInputStream::ReadString**wxString wxDataInputStream::ReadString()**

Reads a string from a stream. Actually, this function first reads a long integer specifying the length of the string (without the last null character) and then reads the string.

4.41 wxDataOutputStream

This class provides functions that write data types in a portable way. So, a file written by an Intel processor can be read by a Sparc or anything else.

4.41.1 wxDataOutputStream::wxDataOutputStream**wxDataInputStream(wxOutputStream& stream)**

Constructs a datastream object from an output stream. Only read methods will be

available.

Parameters

stream

The output stream.

4.41.2 wxDataOutputStream::~~wxDataOutputStream

~wxDataOutputStream()

Destroys the wxDataOutputStream object.

4.41.3 wxDataOutputStream::Write8

void wxDataOutputStream::Write8(unsigned char *i8*)

Writes the single byte *i8* to the stream.

4.41.4 wxDataOutputStream::Write16

void wxDataOutputStream::Write16(unsigned short *i16*)

Writes the 16 bit integer *i16* to the stream.

4.41.5 wxDataOutputStream::Write32

void wxDataOutputStream::Write32(unsigned long *i32*)

Writes the 32 bit integer *i32* to the stream.

4.41.6 wxDataOutputStream::WriteDouble

void wxDataOutputStream::WriteDouble(double *f*)

Writes the double *f* to the stream using the IEEE format.

4.41.7 wxDataOutputStream::WriteLine

void wxDataOutputStream::WriteLine(const wxString& *string*)

Writes *string* as a line. Depending on the operating system, it adds\n or \r\n.

4.41.8 wxDataOutputStream::WriteString

void wxDataOutputStream::WriteString(const wxString& *string*)

Writes *string* to the stream. Actually, this method writes the size of the string before writing *string* itself.

4.42 wxDate

A class for manipulating dates.

NOTE: this class should be used with caution, since it is not fully tested. It will be replaced with a new wxDateTime class in the near future.

Derived from

wxObject (p. 471)

Include files

<wx/date.h>

See also

wxTime (p. 740)

4.42.1 wxDate::wxDate

wxDate()

Default constructor.

wxDate(const wxDate& *date*)

Copy constructor.

wxDate(int *month*, int *day*, int *year*)

Constructor taking month, day and year.

wxDate(long *julian*)

Constructor taking an integer representing the Julian date. This is the number of days since 1st January 4713 B.C., so to convert from the number of days since 1st January 1901, construct a date for 1/1/1901, and add the number of days.

wxDate(const wxString& *dateString*)

Constructor taking a string representing a date. This must be either the string TODAY, or of the form MM/DD/YYYY or MM-DD-YYYY. For example:

```
wxDate date("11/26/1966");
```

Parameters

date

Date to copy.

month

Month: a number between 1 and 12.

day

Day: a number between 1 and 31.

year

Year, such as 1995, 2005.

4.42.2 wxDate::~~wxDate

void ~wxDate()

Destructor.

4.42.3 wxDate::AddMonths

wxDate& AddMonths(int months=1)

Adds the given number of months to the date, returning a reference to 'this'.

4.42.4 wxDate::AddWeeks

wxDate& AddWeeks(int weeks=1)

Adds the given number of weeks to the date, returning a reference to 'this'.

4.42.5 wxDate::AddYears

wxDate& AddYears(int years=1)

Adds the given number of months to the date, returning a reference to 'this'.

4.42.6 wxDate::FormatDate

wxString FormatDate(int type=-1) const

Formats the date according to *type* if not -1, or according to the current display type if -1.

Parameters

type

-1 or one of:

wxDAY	Format day only.
wxMONTH	Format month only.
wxMDY	Format MONTH, DAY, YEAR.
wxFULL	Format day, month and year in US style: DAYOFWEEK, MONTH, DAY, YEAR.
wxEUROPEAN	Format day, month and year in European style: DAY, MONTH, YEAR.

4.42.7 wxDate::GetDay**int GetDay() const**

Returns the numeric day (in the range 1 to 31).

4.42.8 wxDate::GetDayOfWeek**int GetDayOfWeek() const**

Returns the integer day of the week (in the range 1 to 7).

4.42.9 wxDate::GetDayOfWeekName**wxString GetDayOfWeekName() const**

Returns the name of the day of week.

4.42.10 wxDate::GetDayOfYear**long GetDayOfYear() const**

Returns the day of the year (from 1 to 365).

4.42.11 wxDate::GetDaysInMonth

int GetDaysInMonth() const

Returns the number of days in the month (in the range 1 to 31).

4.42.12 wxDate::GetFirstDayOfMonth**int GetFirstDayOfMonth() const**

Returns the day of week that is first in the month (in the range 1 to 7).

4.42.13 wxDate::GetJulianDate**long GetJulianDate() const**

Returns the Julian date.

4.42.14 wxDate::GetMonth**int GetMonth() const**

Returns the month number (in the range 1 to 12).

4.42.15 wxDate::GetMonthEnd**wxDat GetMonthEnd()**

Returns the date representing the last day of the month.

4.42.16 wxDate::GetMonthName**wxString GetMonthName() const**

Returns the name of the month. Do not delete the returned storage.

4.42.17 wxDate::GetMonthStart**wxDat GetMonthStart() const**

Returns the date representing the first day of the month.

4.42.18 wxDate::GetWeekOfMonth**int GetWeekOfMonth() const**

Returns the week of month (in the range 1 to 6).

4.42.19 wxDate::GetWeekOfYear

int GetWeekOfYear() const

Returns the week of year (in the range 1 to 52).

4.42.20 wxDate::GetYear

int GetYear() const

Returns the year as an integer (such as '1995').

4.42.21 wxDate::GetYearEnd

wxDat GetYearEnd() const

Returns the date representing the last day of the year.

4.42.22 wxDate::GetYearStart

wxDat GetYearStart() const

Returns the date representing the first day of the year.

4.42.23 wxDate::IsLeapYear

bool IsLeapYear() const

Returns TRUE if the year of this date is a leap year.

4.42.24 wxDate::Set

wxDat& Set()

Sets the date to current system date, returning a reference to 'this'.

wxDat& Set(long julian)

Sets the date to the given Julian date, returning a reference to 'this'.

wxDat& Set(int month, int day, int year)

Sets the date to the given date, returning a reference to 'this'.

month is a number from 1 to 12.

day is a number from 1 to 31.

year is a year, such as 1995, 2005.

4.42.25 **wxDate::SetFormat**

void SetFormat(int *format*)

Sets the current format type.

Parameters

format

-1 or one of:

wxDAY	Format day only.
wxMONTH	Format month only.
wxMDY	Format MONTH, DAY, YEAR.
wxFULL	Format day, month and year in US style: DAYOFWEEK, MONTH, DAY, YEAR.
wxEUROPEAN	Format day, month and year in European style: DAY, MONTH, YEAR.

4.42.26 **wxDate::SetOption**

int SetOption(int *option*, const bool *enable*=TRUE)

Enables or disables an option for formatting.

Parameters

option

May be one of:

wxNO_CENTURY	The century is not formatted.
wxDATE_ABBR	Month and day names are abbreviated to 3 characters when formatting.

4.42.27 **wxDate::operator wxString**

operator wxString()

Conversion operator, to convert wxDate to wxString by calling FormatDate.

4.42.28 wxDate::operator +

wxDate operator +(long i)

wxDate operator +(int i)

Adds an integer number of days to the date, returning a date.

4.42.29 wxDate::operator -

wxDate operator -(long i)

wxDate operator -(int i)

Subtracts an integer number of days from the date, returning a date.

long operator -(const wxDate& date)

Subtracts one date from another, return the number of intervening days.

4.42.30 wxDate::operator +=

wxDate& operator +=(long i)

Postfix operator: adds an integer number of days to the date, returning a reference to 'this' date.

4.42.31 wxDate::operator -=

wxDate& operator -=(long i)

Postfix operator: subtracts an integer number of days from the date, returning a reference to 'this' date.

4.42.32 wxDate::operator ++

wxDate& operator ++()

Increments the date (postfix or prefix).

4.42.33 wxDate::operator --**wxDat& operator --()**

Decrements the date (postfix or prefix).

4.42.34 wxDate::operator <**friend bool operator <(const wxDate& date1, const wxDate& date2)**

Function to compare two dates, returning TRUE if *date1* is earlier than *date2*.

4.42.35 wxDate::operator <=**friend bool operator <=(const wxDate& date1, const wxDate& date2)**

Function to compare two dates, returning TRUE if *date1* is earlier than or equal to *date2*.

4.42.36 wxDate::operator >**friend bool operator >(const wxDate& date1, const wxDate& date2)**

Function to compare two dates, returning TRUE if *date1* is later than *date2*.

4.42.37 wxDate::operator >=**friend bool operator >=(const wxDate& date1, const wxDate& date2)**

Function to compare two dates, returning TRUE if *date1* is later than or equal to *date2*.

4.42.38 wxDate::operator ==**friend bool operator ==(const wxDate& date1, const wxDate& date2)**

Function to compare two dates, returning TRUE if *date1* is equal to *date2*.

4.42.39 wxDate::operator !=**friend bool operator !=(const wxDate& date1, const wxDate& date2)**

Function to compare two dates, returning TRUE if *date1* is not equal to *date2*.

4.42.40 wxDate::operator <<

friend ostream& operator <<(ostream& os, const wxDate& date)

Function to output a wxDate to an ostream.

4.43 wxDC

A wxDC is a *device context* onto which graphics and text can be drawn. It is intended to represent a number of output devices in a generic way, so a window can have a device context associated with it, and a printer also has a device context. In this way, the same piece of code may write to a number of different devices, if the device context is used as a parameter.

Derived types of wxDC have documentation for specific features only, so refer to this section for most device context information.

Derived from

wxObject (p. 471)

Include files

<wx/dc.h>

See also

Overview (p. 926)

4.43.1 wxDC::wxDC

wxDC()

Constructor.

4.43.2 wxDC::~~wxDC

~wxDC()

Destructor.

4.43.3 wxDC::BeginDrawing

void BeginDrawing()

Allows optimization of drawing code under MS Windows. Enclose drawing primitives

between **BeginDrawing** and **EndDrawing** calls.

Drawing to a `wxDlg` panel device context outside of a system-generated `OnPaint` event *requires* this pair of calls to enclose drawing code. This is because a Windows dialog box does not have a retained device context associated with it, and selections such as pen and brush settings would be lost if the device context were obtained and released for each drawing operation.

4.43.4 `wxDC::Blit`

bool `Blit`(**long** *xdest*, **long** *ydest*, **long** *width*, **long** *height*, **wxDC*** *source*, **long** *xsrc*, **long** *ysrc*, **int** *logicalFunc*, **bool** *useMask*)

Copy from a source DC to this DC, specifying the destination coordinates, size of area to copy, source DC, source coordinates, and logical function.

Parameters

xdest

Destination device context x position.

ydest

Destination device context y position.

width

Width of source area to be copied.

height

Height of source area to be copied.

source

Source device context.

xsrc

Source device context x position.

ysrc

Source device context y position.

logicalFunc

Logical function to use: see `wxDC::SetLogicalFunction` (p. 164).

useMask

If `TRUE`, `Blit` does a transparent blit using the mask that is associated with the bitmap selected into the source device context. The Windows implementation does the following:

1. Creates a temporary bitmap and copies the destination area into it.
2. Copies the source area into the temporary bitmap using the specified logical function.

3. Sets the masked area in the temporary bitmap to BLACK by ANDing the mask bitmap with the temp bitmap with the foreground colour set to WHITE and the bg colour set to BLACK.
4. Sets the unmasked area in the destination area to BLACK by ANDing the mask bitmap with the destination area with the foreground colour set to BLACK and the background colour set to WHITE.
5. ORs the temporary bitmap with the destination area.
6. Deletes the temporary bitmap.

This sequence of operations ensures that the source's transparent area need not be black, and logical functions are supported.

Remarks

There is partial support for Blit in `wxPostScriptDC`, under X.

See *wxMemoryDC* (p. 415) for typical usage.

`wxheading` See also

wxMemoryDC (p. 415), *wxBitmap* (p. 39), *wxMask* (p. 403)

4.43.5 `wxDC::Clear`

void Clear()

Clears the device context using the current background brush.

4.43.6 `wxDC::CrossHair`

void CrossHair(long x, long y)

Displays a cross hair using the current pen. This is a vertical and horizontal line the height and width of the window, centred on the given point.

4.43.7 `wxDC::DestroyClippingRegion`

void DestroyClippingRegion()

Destroys the current clipping region so that none of the DC is clipped. See also *wxDC::SetClippingRegion* (p. 163).

4.43.8 `wxDC::DeviceToLogicalX`

long DeviceToLogicalX(long x)

Convert device X coordinate to logical coordinate, using the current mapping mode.

4.43.9 wxDC::DeviceToLogicalXRel

long DeviceToLogicalXRel(long x)

Convert device X coordinate to relative logical coordinate, using the current mapping mode. Use this function for converting a width, for example.

4.43.10 wxDC::DeviceToLogicalY

long DeviceToLogicalY(long y)

Converts device Y coordinate to logical coordinate, using the current mapping mode.

4.43.11 wxDC::DeviceToLogicalYRel

long DeviceToLogicalYRel(long y)

Convert device Y coordinate to relative logical coordinate, using the current mapping mode. Use this function for converting a height, for example.

4.43.12 wxDC::DrawArc

void DrawArc(long x1, long y1, long x2, long y2, double xc, double yc)

Draws an arc of a circle, centred on (xc, yc), with starting point (x1, y1) and ending at (x2, y2). The current pen is used for the outline and the current brush for filling the shape.

The arc is drawn in an anticlockwise direction from the start point to the end point.

4.43.13 wxDC::DrawBitmap

void DrawBitmap(const wxBitmap& bitmap, long x, long y, bool transparent)

Draw a bitmap on the device context at the specified point. If *transparent* is TRUE and the bitmap has a transparency mask, the bitmap will be drawn transparently.

4.43.14 wxDC::DrawEllipse

void DrawEllipse(long x, long y, long width, long height)

Draws an ellipse contained in the rectangle with the given top left corner, and with the

given size. The current pen is used for the outline and the current brush for filling the shape.

4.43.15 **wxDC::DrawEllipticArc**

void DrawEllipticArc(long x, long y, long width, long height, double start, double end)

Draws an arc of an ellipse. The current pen is used for drawing the arc and the current brush is used for drawing the pie. This function is currently only available for X window and PostScript device contexts.

x and *y* specify the *x* and *y* coordinates of the upper-left corner of the rectangle that contains the ellipse.

width and *height* specify the width and height of the rectangle that contains the ellipse.

start and *end* specify the start and end of the arc relative to the three-o'clock position from the center of the rectangle. Angles are specified in degrees (360 is a complete circle). Positive values mean counter-clockwise motion. If *start* is equal to *end*, a complete ellipse will be drawn.

4.43.16 **wxDC::DrawIcon**

void DrawIcon(const wxIcon& icon, long x, long y)

Draw an icon on the display (does nothing if the device context is PostScript). This can be the simplest way of drawing bitmaps on a window.

4.43.17 **wxDC::DrawLine**

void DrawLine(long x1, long y1, long x2, long y2)

Draws a line from the first point to the second. The current pen is used for drawing the line.

4.43.18 **wxDC::DrawLines**

void DrawLines(int n, wxPoint points[], long xoffset = 0, long yoffset = 0)

void DrawLines(wxList *points, long xoffset = 0, long yoffset = 0)

Draws lines using an array of *points* of size *n*, or list of pointers to points, adding the optional offset coordinate. The current pen is used for drawing the lines. The programmer is responsible for deleting the list of points.

wxPython note:

The wxPython version of this method accepts a Python list of wxPoint objects.

4.43.19 wxDC::DrawPolygon

```
void DrawPolygon(int n, wxPoint points[], long xoffset = 0, long yoffset = 0,  
int fill_style = wxODDEVEN_RULE)
```

```
void DrawPolygon(wxList *points, long xoffset = 0, long yoffset = 0,  
int fill_style = wxODDEVEN_RULE)
```

Draws a filled polygon using an array of *points* of size *n*, or list of pointers to points, adding the optional offset coordinate.

The last argument specifies the fill rule: **wxODDEVEN_RULE** (the default) or **wxWINDING_RULE**.

The current pen is used for drawing the outline, and the current brush for filling the shape. Using a transparent brush suppresses filling. The programmer is responsible for deleting the list of points.

Note that wxWindows automatically closes the first and last points.

wxPython note:

The wxPython version of this method accepts a Python list of wxPoint objects.

4.43.20 wxDC::DrawPoint

```
void DrawPoint(long x, long y)
```

Draws a point using the current pen.

4.43.21 wxDC::DrawRectangle

```
void DrawRectangle(long x, long y, long width, long height)
```

Draws a rectangle with the given top left corner, and with the given size. The current pen is used for the outline and the current brush for filling the shape.

4.43.22 wxDC::DrawRoundedRectangle

```
void DrawRoundedRectangle(long x, long y, long width, long height, double radius =  
20)
```

Draws a rectangle with the given top left corner, and with the given size. The corners are quarter-circles using the given radius. The current pen is used for the outline and the

current brush for filling the shape.

If *radius* is positive, the value is assumed to be the radius of the rounded corner. If *radius* is negative, the absolute value is assumed to be the *proportion* of the smallest dimension of the rectangle. This means that the corner can be a sensible size relative to the size of the rectangle, and also avoids the strange effects X produces when the corners are too big for the rectangle.

4.43.23 **wxDC::DrawSpline**

void DrawSpline(wxList *points)

Draws a spline between all given control points, using the current pen. Doesn't delete the wxList and contents. The spline is drawn using a series of lines, using an algorithm taken from the X drawing program 'XFIG'.

void DrawSpline(long x1, long y1, long x2, long y2, long x3, long y3)

Draws a three-point spline using the current pen.

wxPython note:

The wxPython version of this method accepts a Python list of wxPoint objects.

4.43.24 **wxDC::DrawText**

void DrawText(const wxString& text, long x, long y)

Draws a text string at the specified point, using the current text font, and the current text foreground and background colours.

The coordinates refer to the top-left corner of the rectangle bounding the string. See *wxDC::GetTextExtent* (p. 160) for how to get the dimensions of a text string, which can be used to position the text more precisely.

4.43.25 **wxDC::EndDoc**

void EndDoc()

Ends a document (only relevant when outputting to a printer).

4.43.26 **wxDC::EndDrawing**

void EndDrawing()

Allows optimization of drawing code under MS Windows. Enclose drawing primitives between **BeginDrawing** and **EndDrawing** calls.

4.43.27 wxDC::EndPage**void EndPage()**

Ends a document page (only relevant when outputting to a printer).

4.43.28 wxDC::FloodFill**void FloodFill(long x, long y, wxColour *colour, int style=wxFLOOD_SURFACE)**

Flood fills the device context starting from the given point, in the given colour, and using a style:

- `wxFLOOD_SURFACE`: the flooding occurs until a colour other than the given colour is encountered.
- `wxFLOOD_BORDER`: the area to be flooded is bounded by the given colour.

Note: this function is available in MS Windows only.

4.43.29 wxDC::GetBackground**wxBrush& GetBackground()**

Gets the brush used for painting the background (see *wxDC::SetBackground* (p. 162)).

4.43.30 wxDC::GetBrush**wxBrush& GetBrush()**

Gets the current brush (see *wxDC::SetBrush* (p. 163)).

4.43.31 wxDC::GetCharHeight**long GetCharHeight()**

Gets the character height of the currently set font.

4.43.32 wxDC::GetCharWidth**long GetCharWidth()**

Gets the average character width of the currently set font.

4.43.33 wxDC::GetClippingBox**void GetClippingBox(long *x, long *y, long *width, long *height)**

Gets the rectangle surrounding the current clipping region.

wxPython note:

No arguments are required and the four values defining the rectangle are returned as a tuple.

4.43.34 wxDC::GetFont**wxFont& GetFont()**

Gets the current font (see *wxDC::SetFont* (p. 163)).

4.43.35 wxDC::GetLogicalFunction**int GetLogicalFunction()**

Gets the current logical function (see *wxDC::SetLogicalFunction* (p. 164)).

4.43.36 wxDC::GetMapMode**int GetMapMode()**

Gets the *mapping mode* for the device context (see *wxDC::SetMapMode* (p. 164)).

4.43.37 wxDC::GetOptimization**bool GetOptimization()**

Returns TRUE if device context optimization is on. See *wxDC::SetOptimization* (p. 165) for details.

4.43.38 wxDC::GetPen**wxPen& GetPen()**

Gets the current pen (see *wxDC::SetPen* (p. 165)).

4.43.39 wxDC::GetPixel

bool GetPixel(long x, long y, wxColour *colour)

Sets *colour* to the colour at the specified location. Windows only; an X implementation is being worked on. Not available for wxPostScriptDC or wxMetafileDC.

4.43.40 wxDC::GetSize

void GetSize(long *width, long *height)

For a PostScript device context, this gets the maximum size of graphics drawn so far on the device context.

For a Windows printer device context, this gets the horizontal and vertical resolution. It can be used to scale graphics to fit the page when using a Windows printer device context. For example, if *maxX* and *maxY* represent the maximum horizontal and vertical 'pixel' values used in your application, the following code will scale the graphic to fit on the printer page:

```
long w, h;
dc.GetSize(&w, &h);
double scaleX=(double)(maxX/w);
double scaleY=(double)(maxY/h);
dc.SetUserScale(min(scaleX,scaleY),min(scaleX,scaleY));
```

wxPython note:

No arguments are required and the two values defining the size are returned as a tuple.

4.43.41 wxDC::GetTextBackground

wxColour& GetTextBackground()

Gets the current text background colour (see *wxDC::SetTextBackground* (p. 165)).

4.43.42 wxDC::GetTextExtent

**void GetTextExtent(const wxString& string, long *w, long *h,
long *descent = NULL, long *externalLeading = NULL, wxFont *font = NULL)**

Gets the dimensions of the string using the currently selected font. *string* is the text string to measure, *w* and *h* are the total width and height respectively, *descent* is the dimension from the baseline of the font to the bottom of the descender, and *externalLeading* is any extra vertical space added to the font by the font designer (usually is zero).

The optional parameter *font* specifies an alternative to the currently selected font: but note that this does not yet work under Windows, so you need to set a font for the device context first.

See also *wxFont* (p. 264), *wxDC::SetFont* (p. 163).

wxPython note:

The following methods are implemented in wxPython:

GetTextExtent(string) Returns a 2-tuple, (width, height)

GetFullTextExtent(string, font=NULL) Returns a 4-tuple, (width, height, descent, externalLeading)

4.43.43 wxDC::GetTextForeground

wxColour& GetTextForeground()

Gets the current text foreground colour (see *wxDC::SetTextForeground* (p. 165)).

4.43.44 wxDC::LogicalToDeviceX

long LogicalToDeviceX(long x)

Converts logical X coordinate to device coordinate, using the current mapping mode.

4.43.45 wxDC::LogicalToDeviceXRel

long LogicalToDeviceXRel(long x)

Converts logical X coordinate to relative device coordinate, using the current mapping mode. Use this for converting a width, for example.

4.43.46 wxDC::LogicalToDeviceY

long LogicalToDeviceY(long y)

Converts logical Y coordinate to device coordinate, using the current mapping mode.

4.43.47 wxDC::LogicalToDeviceYRel

long LogicalToDeviceYRel(long y)

Converts logical Y coordinate to relative device coordinate, using the current mapping mode. Use this for converting a height, for example.

4.43.48 wxDC::MaxX

long MaxX()

Gets the maximum horizontal extent used in drawing commands so far.

4.43.49 wxDC::MaxY**long MaxY()**

Gets the maximum vertical extent used in drawing commands so far.

4.43.50 wxDC::MinX**long MinX()**

Gets the minimum horizontal extent used in drawing commands so far.

4.43.51 wxDC::MinY**long MinY()**

Gets the minimum vertical extent used in drawing commands so far.

4.43.52 wxDC::Ok**bool Ok()**

Returns TRUE if the DC is ok to use.

4.43.53 wxDC::SetDeviceOrigin**void SetDeviceOrigin(long x, long y)**

Sets the device origin (i.e., the origin in pixels after scaling has been applied).

This function may be useful in Windows printing operations for placing a graphic on a page.

4.43.54 wxDC::SetBackground**void SetBackground(const wxBrush& brush)**

Sets the current background brush for the DC.

4.43.55 wxDC::SetBackgroundMode**void SetBackgroundMode(int *mode*)**

mode may be one of wxSOLID and wxTRANSPARENT. This setting determines whether text will be drawn with a background colour or not.

4.43.56 wxDC::SetClippingRegion**void SetClippingRegion(long *x*, long *y*, long *width*, long *height*)****void SetClippingRegion(const wxRegion& *region*)**

Sets the clipping region for the DC. The clipping region is an area to which drawing is restricted. Possible uses for the clipping region are for clipping text or for speeding up window redraws when only a known area of the screen is damaged.

[See also](#)

wxDC::DestroyClippingRegion (p. 153), *wxRegion* (p. 562)

4.43.57 wxDC::SetPalette**void SetPalette(const wxPalette& *palette*)**

If this is a window DC or memory DC, assigns the given palette to the window or bitmap associated with the DC. If the argument is wxNullPalette, the current palette is selected out of the device context, and the original palette restored.

See *wxPalette* (p. 484) for further details.

4.43.58 wxDC::SetBrush**void SetBrush(const wxBrush& *brush*)**

Sets the current brush for the DC.

If the argument is wxNullBrush, the current brush is selected out of the device context, and the original brush restored, allowing the current brush to be destroyed safely.

See also *wxBrush* (p. 61).

4.43.59 wxDC::SetFont**void SetFont(const wxFont& *font*)**

Sets the current font for the DC.

If the argument is `wxNullFont`, the current font is selected out of the device context, and the original font restored, allowing the current font to be destroyed safely.

See also *wxFont* (p. 264).

4.43.60 wxDC::SetLogicalFunction

void SetLogicalFunction(int function)

Sets the current logical function for the device context. This determines how a source pixel (from a pen or brush colour, or source device context if using *wxDC::Blit* (p. 152)) combines with a destination pixel in the current device context.

The possible values and their meaning in terms of source and destination pixel values are as follows:

<code>wxAND</code>	<code>src AND dst</code>
<code>wxAND_INVERT</code>	<code>(NOT src) AND dst</code>
<code>wxAND_REVERSE</code>	<code>src AND (NOT dst)</code>
<code>wxCLEAR</code>	<code>0</code>
<code>wxCOPY</code>	<code>src</code>
<code>wxEQUIV</code>	<code>(NOT src) XOR dst</code>
<code>wxINVERT</code>	<code>NOT dst</code>
<code>wxNAND</code>	<code>(NOT src) OR (NOT dst)</code>
<code>wxNOR</code>	<code>(NOT src) AND (NOT dst)</code>
<code>wxNO_OP</code>	<code>dst</code>
<code>wxOR</code>	<code>src OR dst</code>
<code>wxOR_INVERT</code>	<code>(NOT src) OR dst</code>
<code>wxOR_REVERSE</code>	<code>src OR (NOT dst)</code>
<code>wxSET</code>	<code>1</code>
<code>wxSRC_INVERT</code>	<code>NOT src</code>
<code>wxXOR</code>	<code>src XOR dst</code>

The default is `wxCOPY`, which simply draws with the current colour. The others combine the current colour and the background using a logical operation. `wxXOR` is commonly used for drawing rubber bands or moving outlines, since drawing twice reverts to the original colour.

4.43.61 wxDC::SetMapMode

void SetMapMode(int int)

The *mapping mode* of the device context defines the unit of measurement used to convert logical units to device units. Note that in X, text drawing isn't handled consistently with the mapping mode; a font is always specified in point size. However, setting the *user scale* (see *wxDC::SetUserScale* (p. 166)) scales the text appropriately. In Windows, scaleable TrueType fonts are always used; in X, results depend on availability of fonts, but usually a reasonable match is found.

Note that the coordinate origin should ideally be selectable, but for now is always at the top left of the screen/printer.

Drawing to a Windows printer device context under UNIX uses the current mapping mode, but mapping mode is currently ignored for PostScript output.

The mapping mode can be one of the following:

<code>wxMM_TWIPS</code>	Each logical unit is 1/20 of a point, or 1/1440 of an inch.
<code>wxMM_POINTS</code>	Each logical unit is a point, or 1/72 of an inch.
<code>wxMM_METRIC</code>	Each logical unit is 1 mm.
<code>wxMM_LOMETRIC</code>	Each logical unit is 1/10 of a mm.
<code>wxMM_TEXT</code>	Each logical unit is 1 pixel.

4.43.62 `wxDC::SetOptimization`

`void SetOptimization(bool optimize)`

If *optimize* is TRUE (the default), this function sets optimization mode on. This currently means that under X, the device context will not try to set a pen or brush property if it is known to be set already. This approach can fall down if non-wxWindows code is using the same device context or window, for example when the window is a panel on which the windowing system draws panel items. The wxWindows device context 'memory' will now be out of step with reality.

Setting optimization off, drawing, then setting it back on again, is a trick that must occasionally be employed.

4.43.63 `wxDC::SetPen`

`void SetPen(const wxPen& pen)`

Sets the current pen for the DC.

If the argument is `wxNullPen`, the current pen is selected out of the device context, and the original pen restored.

4.43.64 `wxDC::SetTextBackground`

`void SetTextBackground(const wxColour& colour)`

Sets the current text background colour for the DC.

4.43.65 `wxDC::SetTextForeground`

void SetTextForeground(const wxColour& colour)

Sets the current text foreground colour for the DC.

4.43.66 wxDC::SetUserScale

void SetUserScale(double xScale, double yScale)

Sets the user scaling factor, useful for applications which require 'zooming'.

4.43.67 wxDC::StartDoc

bool StartDoc(const wxString& message)

Starts a document (only relevant when outputting to a printer). Message is a message to show whilst printing.

4.43.68 wxDC::StartPage

bool StartPage()

Starts a document page (only relevant when outputting to a printer).

4.44 wxDDEClient

A wxDDEClient object represents the client part of a client-server DDE (Dynamic Data Exchange) conversation.

To create a client which can communicate with a suitable server, you need to derive a class from wxDDEConnection and another from wxDDEClient. The custom wxDDEConnection class will intercept communications in a 'conversation' with a server, and the custom wxDDEServer is required so that a user-overridden *wxDDEClient::OnMakeConnection* (p. 167) member can return a wxDDEConnection of the required class, when a connection is made.

This DDE-based implementation is available on Windows only, but a platform-independent, socket-based version of this API is available using *wxTCPClient* (p. 703).

Derived from

wxClientBase
wxObject (p. 471)

Include files

<wx/dde.h>

See also

wxDDEServer (p. 172), *wxDDEConnection* (p. 167), *Interprocess communications overview* (p. 946)

4.44.1 wxDDEClient::wxDDEClient

wxDDEClient()

Constructs a client object.

4.44.2 wxDDEClient::MakeConnection

wxConnectionBase * MakeConnection(const wxString& host, const wxString& service, const wxString& topic)

Tries to make a connection with a server specified by the host (machine name under UNIX, ignored under Windows), service name (must contain an integer port number under UNIX), and topic string. If the server allows a connection, a *wxDDEConnection* object will be returned. The type of *wxDDEConnection* returned can be altered by overriding the *wxDDEClient::OnMakeConnection* (p. 167) member to return your own derived connection object.

4.44.3 wxDDEClient::OnMakeConnection

wxConnectionBase * OnMakeConnection()

The type of *wxDDEConnection* (p. 167) returned from a *wxDDEClient::MakeConnection* (p. 167) call can be altered by deriving the **OnMakeConnection** member to return your own derived connection object. By default, a *wxDDEConnection* object is returned.

The advantage of deriving your own connection class is that it will enable you to intercept messages initiated by the server, such as *wxDDEConnection::OnAdvise* (p. 170). You may also want to store application-specific data in instances of the new class.

4.44.4 wxDDEClient::ValidHost

bool ValidHost(const wxString& host)

Returns TRUE if this is a valid host name, FALSE otherwise. This always returns TRUE under MS Windows.

4.45 wxDDEConnection

A `wxDDEConnection` object represents the connection between a client and a server. It can be created by making a connection using a `wxDDEClient` (p. 166) object, or by the acceptance of a connection by a `wxDDEServer` (p. 172) object. The bulk of a DDE (Dynamic Data Exchange) conversation is controlled by calling members in a **`wxDDEConnection`** object or by overriding its members.

An application should normally derive a new connection class from `wxDDEConnection`, in order to override the communication event handlers to do something interesting.

This DDE-based implementation is available on Windows only, but a platform-independent, socket-based version of this API is available using `wxTCPConnection` (p. 705).

Derived from

`wxConnectionBase`
`wxObject` (p. 471)

Include files

`<wx/dde.h>`

Types

`wxIPCFormat` is defined as follows:

```
enum wxIPCFormat
{
    wxIPC_INVALID =          0,
    wxIPC_TEXT =             1, /* CF_TEXT */
    wxIPC_BITMAP =           2, /* CF_BITMAP */
    wxIPC_METAFILE =         3, /* CF_METAFILEPICT */
    wxIPC_SYLK =              4,
    wxIPC_DIF =               5,
    wxIPC_TIFF =              6,
    wxIPC_OEMTEXT =           7, /* CF_OEMTEXT */
    wxIPC_DIB =               8, /* CF_DIB */
    wxIPC_PALETTE =           9,
    wxIPC_PENDATA =           10,
    wxIPC_RIFF =              11,
    wxIPC_WAVE =              12,
    wxIPC_UNICODETEXT =       13,
    wxIPC_ENHMETAFILE =       14,
    wxIPC_FILENAME =          15, /* CF_HDROP */
    wxIPC_LOCALE =            16,
    wxIPC_PRIVATE =           20
};
```

See also

`wxDDEClient` (p. 166), `wxDDEServer` (p. 172), *Interprocess communications overview* (p. 946)

4.45.1 wxDDEConnection::wxDDEConnection

wxDDEConnection()

wxDDEConnection(char* buffer, int size)

Constructs a connection object. If no user-defined connection object is to be derived from `wxDDEConnection`, then the constructor should not be called directly, since the default connection object will be provided on requesting (or accepting) a connection. However, if the user defines his or her own derived connection object, the `wxDDEServer::OnAcceptConnection` (p. 172) and/or `wxDDEClient::OnMakeConnection` (p. 167) members should be replaced by functions which construct the new connection object. If the arguments of the `wxDDEConnection` constructor are void, then a default buffer is associated with the connection. Otherwise, the programmer must provide a a buffer and size of the buffer for the connection object to use in transactions.

4.45.2 wxDDEConnection::Advise

bool Advise(const wxString& item, char* data, int size = -1, wxIPCFormat format = wxCF_TEXT)

Called by the server application to advise the client of a change in the data associated with the given item. Causes the client connection's `wxDDEConnection::OnAdvise` (p. 170) member to be called. Returns TRUE if successful.

4.45.3 wxDDEConnection::Execute

bool Execute(char* data, int size = -1, wxIPCFormat format = wxCF_TEXT)

Called by the client application to execute a command on the server. Can also be used to transfer arbitrary data to the server (similar to `wxDDEConnection::Poke` (p. 171) in that respect). Causes the server connection's `wxDDEConnection::OnExecute` (p. 170) member to be called. Returns TRUE if successful.

4.45.4 wxDDEConnection::Disconnect

bool Disconnect()

Called by the client or server application to disconnect from the other program; it causes the `wxDDEConnection::OnDisconnect` (p. 170) message to be sent to the corresponding connection object in the other program. The default behaviour of **OnDisconnect** is to delete the connection, but the calling application must explicitly delete its side of the connection having called **Disconnect**. Returns TRUE if successful.

4.45.5 wxDDEConnection::OnAdvise

virtual bool OnAdvise(const wxString& topic, const wxString& item, char* data, int size, wxIPCFormat format)

Message sent to the client application when the server notifies it of a change in the data associated with the given item.

4.45.6 wxDDEConnection::OnDisconnect

virtual bool OnDisconnect()

Message sent to the client or server application when the other application notifies it to delete the connection. Default behaviour is to delete the connection object.

4.45.7 wxDDEConnection::OnExecute

virtual bool OnExecute(const wxString& topic, char* data, int size, wxIPCFormat format)

Message sent to the server application when the client notifies it to execute the given data. Note that there is no item associated with this message.

4.45.8 wxDDEConnection::OnPoke

virtual bool OnPoke(const wxString& topic, const wxString& item, char* data, int size, wxIPCFormat format)

Message sent to the server application when the client notifies it to accept the given data.

4.45.9 wxDDEConnection::OnRequest

virtual char* OnRequest(const wxString& topic, const wxString& item, int *size, wxIPCFormat format)

Message sent to the server application when the client calls *wxDDEConnection::Request* (p. 171). The server should respond by returning a character string from **OnRequest**, or NULL to indicate no data.

4.45.10 wxDDEConnection::OnStartAdvise

virtual bool OnStartAdvise(const wxString& topic, const wxString& item)

Message sent to the server application by the client, when the client wishes to start an 'advise loop' for the given topic and item. The server can refuse to participate by returning FALSE.

4.45.11 wxDDEConnection::OnStopAdvise

virtual bool OnStopAdvise(const wxString& topic, const wxString& item)

Message sent to the server application by the client, when the client wishes to stop an 'advise loop' for the given topic and item. The server can refuse to stop the advise loop by returning FALSE, although this doesn't have much meaning in practice.

4.45.12 wxDDEConnection::Poke

bool Poke(const wxString& item, char* data, int size = -1, wxIPCFormat format = wxCF_TEXT)

Called by the client application to poke data into the server. Can be used to transfer arbitrary data to the server. Causes the server connection's *wxDDEConnection::OnPoke* (p. 170) member to be called. Returns TRUE if successful.

4.45.13 wxDDEConnection::Request

char* Request(const wxString& item, int *size, wxIPCFormat format = wxIPC_TEXT)

Called by the client application to request data from the server. Causes the server connection's *wxDDEConnection::OnRequest* (p. 170) member to be called. Returns a character string (actually a pointer to the connection's buffer) if successful, NULL otherwise.

4.45.14 wxDDEConnection::StartAdvise

bool StartAdvise(const wxString& item)

Called by the client application to ask if an advise loop can be started with the server. Causes the server connection's *wxDDEConnection::OnStartAdvise* (p. 170) member to be called. Returns TRUE if the server okays it, FALSE otherwise.

4.45.15 wxDDEConnection::StopAdvise

bool StopAdvise(const wxString& item)

Called by the client application to ask if an advise loop can be stopped. Causes the server connection's *wxDDEConnection::OnStopAdvise* (p. 171) member to be called. Returns TRUE if the server okays it, FALSE otherwise.

4.46 wxDDEServer

A wxDDEServer object represents the server part of a client-server DDE (Dynamic Data Exchange) conversation.

This DDE-based implementation is available on Windows only, but a platform-independent, socket-based version of this API is available using *wxTCPServer* (p. 709).

Derived from

wxServerBase

Include files

<wx/dde.h>

See also

wxDDEClient (p. 166), *wxDDEConnection* (p. 167), *IPC overview* (p. 946)

4.46.1 wxDDEServer::wxDDEServer

wxDDEServer()

Constructs a server object.

4.46.2 wxDDEServer::Create

bool Create(const wxString& service)

Registers the server using the given service name. Under UNIX, the string must contain an integer id which is used as an Internet port number. FALSE is returned if the call failed (for example, the port number is already in use).

4.46.3 wxDDEServer::OnAcceptConnection

virtual wxConnectionBase * OnAcceptConnection(const wxString& topic)

When a client calls **MakeConnection**, the server receives the message and this member is called. The application should derive a member to intercept this message and return a connection object of either the standard wxDDEConnection type, or of a user-derived type. If the topic is "STDIO", the application may wish to refuse the connection. Under UNIX, when a server is created the OnAcceptConnection message is always sent for standard input and output, but in the context of DDE messages it doesn't make a lot

of sense.

4.47 wxDebugContext

A class for performing various debugging and memory tracing operations. Full functionality (such as printing out objects currently allocated) is only present in a debugging build of wxWindows, i.e. if the `DEBUG` symbol is defined and non-zero. `wxDebugContext` and related functions and macros can be compiled out by setting `wxUSE_DEBUG_CONTEXT` to 0 in `setup.h`.

Derived from

No parent class.

Include files

<wx/memory.h>

See also

Overview (p. 928)

4.47.1 wxDebugContext::Check

int Check()

Checks the memory blocks for errors, starting from the currently set checkpoint.

Return value

Returns the number of errors, so a value of zero represents success. Returns -1 if an error was detected that prevents further checking.

4.47.2 wxDebugContext::Dump

bool Dump()

Performs a memory dump from the currently set checkpoint, writing to the current debug stream. Calls the **Dump** member function for each `wxObject` derived instance.

Return value

TRUE if the function succeeded, FALSE otherwise.

4.47.3 wxDebugContext::GetCheckPrevious

bool GetCheckPrevious()

Returns TRUE if the memory allocator checks all previous memory blocks for errors. By default, this is FALSE since it slows down execution considerably.

[See also](#)

wxDebugContext::SetCheckPrevious (p. 176)

4.47.4 wxDebugContext::GetDebugMode**bool GetDebugMode()**

Returns TRUE if debug mode is on. If debug mode is on, the wxObject new and delete operators store or use information about memory allocation. Otherwise, a straight malloc and free will be performed by these operators.

[See also](#)

wxDebugContext::SetDebugMode (p. 176)

4.47.5 wxDebugContext::GetLevel**int GetLevel()**

Gets the debug level (default 1). The debug level is used by the wxTraceLevel function and the WXTRACELEVEL macro to specify how detailed the trace information is; setting a different level will only have an effect if trace statements in the application specify a value other than one.

This is obsolete, replaced by *wxLog* (p. 399) functionality.

[See also](#)

wxDebugContext::SetLevel (p. 177)

4.47.6 wxDebugContext::GetStream**ostream& GetStream()**

Returns the output stream associated with the debug context.

This is obsolete, replaced by *wxLog* (p. 399) functionality.

[See also](#)

wxDebugContext::SetStream (p. 177)

4.47.7 wxDebugContext::GetStreamBuf

streambuf* *GetStreamBuf*()

Returns a pointer to the output stream buffer associated with the debug context. There may not necessarily be a stream buffer if the stream has been set by the user.

This is obsolete, replaced by *wxLog* (p. 399) functionality.

4.47.8 wxDebugContext::HasStream

bool *HasStream*()

Returns TRUE if there is a stream currently associated with the debug context.

This is obsolete, replaced by *wxLog* (p. 399) functionality.

[See also](#)

wxDebugContext::SetStream (p. 177), *wxDebugContext::GetStream* (p. 174)

4.47.9 wxDebugContext::PrintClasses

bool *PrintClasses*()

Prints a list of the classes declared in this application, giving derivation and whether instances of this class can be dynamically created.

[See also](#)

wxDebugContext::PrintStatistics (p. 175)

4.47.10 wxDebugContext::PrintStatistics

bool *PrintStatistics*(**bool** *detailed* = TRUE)

Performs a statistics analysis from the currently set checkpoint, writing to the current debug stream. The number of object and non-object allocations is printed, together with the total size.

[Parameters](#)

detailed

If TRUE, the function will also print how many objects of each class have been

allocated, and the space taken by these class instances.

See also

wxDebugContext::PrintStatistics (p. 175)

4.47.11 wxDebugContext::SetCheckpoint

void SetCheckpoint(bool *all* = FALSE)

Sets the current checkpoint: Dump and PrintStatistics operations will be performed from this point on. This allows you to ignore allocations that have been performed up to this point.

Parameters

all

If TRUE, the checkpoint is reset to include all memory allocations since the program started.

4.47.12 wxDebugContext::SetCheckPrevious

void SetCheckPrevious(bool *check*)

Tells the memory allocator to check all previous memory blocks for errors. By default, this is FALSE since it slows down execution considerably.

See also

wxDebugContext::GetCheckPrevious (p. 173)

4.47.13 wxDebugContext::SetDebugMode

void SetDebugMode(bool *debug*)

Sets the debug mode on or off. If debug mode is on, the wxObject new and delete operators store or use information about memory allocation. Otherwise, a straight malloc and free will be performed by these operators.

By default, debug mode is on if DEBUG is non-zero. If the application uses this function, it should make sure that all object memory allocated is deallocated with the same value of debug mode. Otherwise, the delete operator might try to look for memory information that does not exist.

See also

wxDebugContext::GetDebugMode (p. 174)

4.47.14 wxDebugContext::SetFile**bool SetFile(const wxString& filename)**

Sets the current debug file and creates a stream. This will delete any existing stream and stream buffer. By default, the debug context stream outputs to the debugger (Windows) or standard error (other platforms).

4.47.15 wxDebugContext::SetLevel**void SetLevel(int level)**

Sets the debug level (default 1). The debug level is used by the wxTraceLevel function and the WXTRACELEVEL macro to specify how detailed the trace information is; setting a different level will only have an effect if trace statements in the application specify a value other than one.

This is obsolete, replaced by *wxLog* (p. 399) functionality.

[See also](#)

wxDebugContext::GetLevel (p. 174)

4.47.16 wxDebugContext::SetStandardError**bool SetStandardError()**

Sets the debugging stream to be the debugger (Windows) or standard error (other platforms). This is the default setting. The existing stream will be flushed and deleted.

This is obsolete, replaced by *wxLog* (p. 399) functionality.

4.47.17 wxDebugContext::SetStream**void SetStream(ostream* stream, streambuf* streamBuf = NULL)**

Sets the stream and optionally, stream buffer associated with the debug context. This operation flushes and deletes the existing stream (and stream buffer if any).

This is obsolete, replaced by *wxLog* (p. 399) functionality.

Parameters

stream

Stream to associate with the debug context. Do not set this to NULL.

streamBuf

Stream buffer to associate with the debug context.

See also

wxDebugContext::GetStream (p. 174), *wxDebugContext::HasStream* (p. 175)

4.48 wxDebugStreamBuf

This class allows you to treat debugging output in a similar (stream-based) fashion on different platforms. Under Windows, an ostream constructed with this buffer outputs to the debugger, or other program that intercepts debugging output. On other platforms, the output goes to standard error (cerr).

This is soon to be obsolete, replaced by *wxLog* (p. 399) functionality.

Derived from

streambuf

Include files

<wx/memory.h>

Example

```
wxDebugStreamBuf streamBuf;  
ostream stream(&streamBuf);  
  
stream << "Hello world!" << endl;
```

See also

Overview (p. 928)

4.49 wxDialog

A dialog box is a window with a title bar and sometimes a system menu, which can be moved around the screen. It can contain controls and other windows.

Derived from

wxPanel (p. 488)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/dialog.h>

Remarks

There are two kinds of dialog - *modal* and *modeless*. A modal dialog blocks program flow and user input on other windows until it is dismissed, whereas a modeless dialog behaves more like a frame in that program flow continues, and input on other windows is still possible. You specify the type of dialog with the **wxDIALOG_MODAL** and **wxDIALOG_MODELESS** window styles.

A dialog may be loaded from a wxWindows resource file (extension `wxr`).

An application can define an *OnCloseWindow* (p. 819) handler for the dialog to respond to system close events.

Window styles

wxCAPTION	Puts a caption on the dialog box (Motif only).
wxDEFAULT_DIALOG_STYLE	Equivalent to a combination of wxCAPTION , wxSYSTEM_MENU and wxTHICK_FRAME
wxRESIZE_BORDER	Display a resizable frame around the window (Motif only).
wxSYSTEM_MENU	Display a system menu.
wxTHICK_FRAME	Display a thick frame around the window.
wxSTAY_ON_TOP	The dialog stays on top of all other windows (Windows only).
wxNO_3D	Under Windows, specifies that the child controls should not have 3D borders unless specified in the control.

Under Motif, MWM (the Motif Window Manager) should be running for any of these styles to have an effect.

See also *Generic window styles* (p. 959).

See also

wxDialog overview (p. 910), *wxFrame* (p. 275), *Resources* (p. 7), *Validator overview* (p. 969)

4.49.1 wxDialog::wxDialog

wxDialog()

Default constructor.

wxDialog(wxWindow* parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT_DIALOG_STYLE, const wxString& name = "dialogBox")

Constructor.

Parameters

parent

Can be NULL, a frame or another dialog box.

id

An identifier for the dialog. A value of -1 is taken to mean a default.

title

The title of the dialog.

pos

The dialog position. A value of (-1, -1) indicates a default position, chosen by either the windowing system or wxWindows, depending on platform.

size

The dialog size. A value of (-1, -1) indicates a default size, chosen by either the windowing system or wxWindows, depending on platform.

style

The window style. See *wxDialog* (p. 178).

name

Used to associate a name with the window, allowing the application user to set Motif resource values for individual dialog boxes.

See also

wxDialog::Create (p. 181)

4.49.2 wxDialog::~~wxDialog

~wxDialog()

Destructor. Deletes any child windows before deleting the physical window.

4.49.3 wxDialog::Centre

void Centre(int direction = wxBOTH)

Centres the dialog box on the display.

Parameters

direction

May be wxHORIZONTAL, wxVERTICAL or wxBOTH.

4.49.4 wxDialog::Create

bool Create(wxWindow* *parent*, wxWindowID *id*, const wxString& *title*, const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size* = wxDefaultSize, long *style* = wxDEFAULT_DIALOG_STYLE, const wxString& *name* = "dialogBox")

Used for two-step dialog box construction. See *wxDialog::wxDialog* (p. 179) for details.

4.49.5 wxDialog::EndModal

void EndModal(int *retCode*)

Ends a modal dialog, passing a value to be returned from the *wxDialog::ShowModal* (p. 185) invocation.

Parameters

retCode

The value that should be returned by **ShowModal**.

See also

wxDialog::ShowModal (p. 185), *wxWindow::GetReturnCode* (p. 811),
wxWindow::SetReturnCode (p. 837)

4.49.6 wxDialog::GetTitle

wxString GetTitle() const

Returns the title of the dialog box.

4.49.7 wxDialog::Iconize

void Iconize(const bool *iconize*)

Iconizes or restores the dialog.

Parameters

iconize

If TRUE, iconizes the dialog box; if FALSE, shows and restores it.

Remarks

Note that in Windows, iconization has no effect since dialog boxes cannot be iconized. However, applications may need to explicitly restore dialog boxes under Motif which

have user-iconizable frames, and under Windows calling `Iconize(FALSE)` will bring the window to the front, as does `Show(TRUE)`.

4.49.8 `wxDialog::IsIconized`

`bool IsIconized() const`

Returns TRUE if the dialog box is iconized.

Remarks

Always returns FALSE under Windows since dialogs cannot be iconized.

4.49.9 `wxDialog::IsModal`

`bool IsModal() const`

Returns TRUE if the dialog box is modal, FALSE otherwise.

4.49.10 `wxDialog::OnCharHook`

`void OnCharHook(wxKeyEvent& event)`

This member is called to allow the window to intercept keyboard events before they are processed by child windows.

For more information, see *wxWindow::OnCharHook* (p. 818)

Remarks

`wxDialog` implements this handler to fake a cancel command if the escape key has been pressed. This will dismiss the dialog.

4.49.11 `wxDialog::OnApply`

`void OnApply(wxCommandEvent& event)`

The default handler for the `wxID_APPLY` identifier.

Remarks

This function calls *wxWindow::Validate* (p. 842) and *wxWindow::TransferDataToWindow* (p. 842).

See also

wxDialog::OnOK (p. 183), *wxDialog::OnCancel* (p. 183)

4.49.12 **wxDialog::OnCancel**

void OnCancel(wxCommandEvent& event)

The default handler for the wxID_CANCEL identifier.

Remarks

The function either calls **EndModal(wxID_CANCEL)** if the dialog is modal, or sets the return value to wxID_CANCEL and calls **Show(FALSE)** if the dialog is modeless.

See also

wxDialog::OnOK (p. 183), *wxDialog::OnApply* (p. 182)

4.49.13 **wxDialog::OnOK**

void OnOK(wxCommandEvent& event)

The default handler for the wxID_OK identifier.

Remarks

The function calls *wxWindow::Validate* (p. 842), then *wxWindow::TransferDataFromWindow* (p. 842). If this returns TRUE, the function either calls **EndModal(wxID_OK)** if the dialog is modal, or sets the return value to wxID_OK and calls **Show(FALSE)** if the dialog is modeless.

See also

wxDialog::OnCancel (p. 183), *wxDialog::OnApply* (p. 182)

4.49.14 **wxDialog::OnSysColourChanged**

void OnSysColourChanged(wxSysColourChangedEvent& event)

The default handler for wxEVT_SYS_COLOUR_CHANGED.

Parameters

event

The colour change event.

Remarks

Changes the dialog's colour to conform to the current settings (Windows only). Add an event table entry for your dialog class if you wish the behaviour to be different (such as keeping a user-defined background colour). If you do override this function, call *wxWindow::OnSysColourChanged* (p. 828) to propagate the notification to child windows and controls.

See also

wxSysColourChangedEvent (p. 678)

4.49.15 **wxDialog::SetModal**

void SetModal(const bool *flag*)

Allows the programmer to specify whether the dialog box is modal (*wxDialog::Show* blocks control until the dialog is hidden) or modeless (control returns immediately).

Parameters

flag

If TRUE, the dialog will be modal, otherwise it will be modeless.

4.49.16 **wxDialog::SetTitle**

void SetTitle(const wxString& *title*)

Sets the title of the dialog box.

Parameters

title

The dialog box title.

4.49.17 **wxDialog::Show**

bool Show(const bool *show*)

Hides or shows the dialog.

Parameters

show

If TRUE, the dialog box is shown and brought to the front; otherwise the box is hidden. If FALSE and the dialog is modal, control is returned to the calling program.

Remarks

The preferred way of dismissing a modal dialog is to use *wxDialog::EndModal* (p. 181).

4.49.18 **wxDialog::ShowModal**

int ShowModal()

Shows a modal dialog. Program flow does not return until the dialog has been dismissed with *wxDialog::EndModal* (p. 181).

Return value

The return value is the value set with *wxWindow::SetReturnCode* (p. 837).

See also

wxDialog::EndModal (p. 181), *wxWindow::GetReturnCode* (p. 811),
wxWindow::SetReturnCode (p. 837)

4.50 **wxDirDialog**

This class represents the directory chooser dialog.

Derived from

wxDialog (p. 178)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/dirdlg.h>

See also

wxDirDialog overview (p. 918), *wxFileDialog* (p. 248)

4.50.1 **wxDirDialog::wxDirDialog**

**wxDirDialog(wxWindow* parent, const wxString& message = "Choose a directory",
const wxString& defaultPath = "", long style = 0, const wxPoint& pos =
wxDefaultPosition)**

Constructor. Use *wxDirDialog::ShowModal* (p. 187) to show the dialog.

Parameters

parent

Parent window.

message

Message to show on the dialog.

defaultPath

The default path, or the empty string.

style

A dialog style, currently unused.

pos

Dialog position. Not implemented.

4.50.2 wxDirDialog::~wxDirDialog

~wxDirDialog()

Destructor.

4.50.3 wxDirDialog::GetPath

wxString GetPath() const

Returns the default or user-selected path.

4.50.4 wxDirDialog::GetMessage

wxString GetMessage() const

Returns the message that will be displayed on the dialog.

4.50.5 wxDirDialog::GetStyle

long GetStyle() const

Returns the dialog style.

4.50.6 wxDirDialog::SetMessage

void SetMessage(const wxString& message)

Sets the message that will be displayed on the dialog.

4.50.7 `wxDirDialog::SetPath`

void SetPath(const wxString& path)

Sets the default path.

4.50.8 `wxDirDialog::SetStyle`

void SetStyle(long style)

Sets the dialog style. This is currently unused.

4.50.9 `wxDirDialog::ShowModal`

int ShowModal()

Shows the dialog, returning `wxID_OK` if the user pressed OK, and `wxOK_CANCEL` otherwise.

4.51 `wxDocChildFrame`

The `wxDocChildFrame` class provides a default frame for displaying documents on separate windows. This class can only be used for SDI (not MDI) child frames.

The class is part of the document/view framework supported by `wxWindows`, and cooperates with the `wxView` (p. 793), `wxDocument` (p. 207), `wxDocManager` (p. 189) and `wxDocTemplate` (p. 202) classes.

See the example application in `samples/docview`.

Derived from

`wxFrame` (p. 275)
`wxWindow` (p. 798)
`wxEvtHandler` (p. 224)
`wxObject` (p. 471)

Include files

`<wx/docview.h>`

See also

Document/view overview (p. 933), `wxFrame` (p. 275)

4.51.1 wxDocChildFrame::m_childDocument

wxDocument* m_childDocument

The document associated with the frame.

4.51.2 wxDocChildFrame::m_childView

wxView* m_childView

The view associated with the frame.

4.51.3 wxDocChildFrame::wxDocChildFrame

wxDocChildFrame(wxDocument* doc, wxView* view, wxFrame* parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT_FRAME_STYLE, const wxString& name = "frame")

Constructor.

4.51.4 wxDocChildFrame::~~wxDocChildFrame

~wxDocChildFrame()

Destructor.

4.51.5 wxDocChildFrame::GetDocument

wxDocument* GetDocument() const

Returns the document associated with this frame.

4.51.6 wxDocChildFrame::GetView

wxView* GetView() const

Returns the view associated with this frame.

4.51.7 wxDocChildFrame::OnActivate

void OnActivate(wxActivateEvent event)

Sets the currently active view to be the frame's view. You may need to override (but still call) this function in order to set the keyboard focus for your subwindow.

4.51.8 wxDocChildFrame::OnCloseWindow

void OnCloseWindow(wxCloseEvent& *event*)

Closes and deletes the current view and document.

4.51.9 wxDocChildFrame::SetDocument

void SetDocument(wxDocument **doc*)

Sets the document for this frame.

4.51.10 wxDocChildFrame::SetView

void SetView(wxView **view*)

Sets the view for this frame.

4.52 wxDocManager

The wxDocManager class is part of the document/view framework supported by wxWindows, and cooperates with the wxView (p. 793), wxDocument (p. 207) and wxDocTemplate (p. 202) classes.

Derived from

wxEvtHandler (p. 224)

wxObject (p. 471)

Include files

<wx/docview.h>

See also

wxDocManager overview (p. 937), *wxDocument* (p. 207), *wxView* (p. 793), *wxDocTemplate* (p. 202), *wxFileHistory* (p. 253)

4.52.1 wxDocManager::m_currentView

wxView* m_currentView

The currently active view.

4.52.2 wxDocManager::m_defaultDocumentNameCounter

int m_defaultDocumentNameCounter

Stores the integer to be used for the next default document name.

4.52.3 wxDocManager::m_fileHistory

wxFileHistory* m_fileHistory

A pointer to an instance of *wxFileHistory* (p. 253), which manages the history of recently-visited files on the File menu.

4.52.4 wxDocManager::m_maxDocsOpen

int m_maxDocsOpen

Stores the maximum number of documents that can be opened before existing documents are closed. By default, this is 10,000.

4.52.5 wxDocManager::m_docs

wxList m_docs

A list of all documents.

4.52.6 wxDocManager::m_flags

long m_flags

Stores the flags passed to the constructor.

4.52.7 wxDocManager::m_templates

wxList mnTemplates

A list of all document templates.

4.52.8 wxDocManager::wxDocManager

void wxDocManager(long flags = wxDEFAULT_DOCMAN_FLAGS, bool initialize = TRUE)

Constructor. Create a document manager instance dynamically near the start of your application before doing any document or view operations.

flags is currently unused.

If *initialize* is TRUE, the *Initialize* (p. 194) function will be called to create a default history list object. If you derive from *wxDocManager*, you may wish to call the base constructor with FALSE, and then call *Initialize* in your own constructor, to allow your own *Initialize* or *OnCreateFileHistory* functions to be called.

4.52.9 wxDocManager::~~wxDocManager

void ~wxDocManager()

Destructor.

4.52.10 wxDocManager::ActivateView

void ActivateView(wxView* doc, bool activate, bool deleting)

Sets the current view.

4.52.11 wxDocManager::AddDocument

void AddDocument(wxDocument *doc)

Adds the document to the list of documents.

4.52.12 wxDocManager::AddFileToHistory

void AddFileToHistory(const wxString& filename)

Adds a file to the file history list, if we have a pointer to an appropriate file menu.

4.52.13 wxDocManager::AssociateTemplate

void AssociateTemplate(wxDocTemplate *temp)

Adds the template to the document manager's template list.

4.52.14 wxDocManager::CreateDocument

wxDocument* CreateDocument(const wxString& path, long flags)

Creates a new document in a manner determined by the *flags* parameter, which can be:

- wxDOC_NEW Creates a fresh document.
- wxDOC_SILENT Silently loads the given document file.

If wxDOC_NEW is present, a new document will be created and returned, possibly after asking the user for a template to use if there is more than one document template. If wxDOC_SILENT is present, a new document will be created and the given file loaded into it. If neither of these flags is present, the user will be presented with a file selector for the file to load, and the template to use will be determined by the extension (Windows) or by popping up a template choice list (other platforms).

If the maximum number of documents has been reached, this function will delete the oldest currently loaded document before creating a new one.

4.52.15 wxDocManager::CreateView**wxView* CreateView(wxDocument* doc, long flags)**

Creates a new view for the given document. If more than one view is allowed for the document (by virtue of multiple templates mentioning the same document type), a choice of view is presented to the user.

4.52.16 wxDocManager::DisassociateTemplate**void DisassociateTemplate(wxDocTemplate *temp)**

Removes the template from the list of templates.

4.52.17 wxDocManager::FileHistoryAddFilesToMenu**void FileHistoryAddFilesToMenu()**

Appends the files in the history list, to all menus managed by the file history object.

void FileHistoryAddFilesToMenu(wxMenu* menu)

Appends the files in the history list, to the given menu only.

4.52.18 wxDocManager::FileHistoryLoad**void FileHistoryLoad(wxConfigBase& config)**

Loads the file history from a config object.

[See also](#)

wxConfig (p. 111)

4.52.19 wxDocManager::FileHistoryRemoveMenu

void FileHistoryRemoveMenu(wxMenu* menu)

Removes the given menu from the list of menus managed by the file history object.

4.52.20 wxDocManager::FileHistorySave

void FileHistorySave(wxConfigBase& resourceFile)

Saves the file history into a config object. This must be called explicitly by the application.

[See also](#)

wxConfig (p. 111)

4.52.21 wxDocManager::FileHistoryUseMenu

void FileHistoryUseMenu(wxMenu* menu)

Use this menu for appending recently-visited document filenames, for convenient access. Calling this function with a valid menu pointer enables the history list functionality.

Note that you can add multiple menus using this function, to be managed by the file history object.

4.52.22 wxDocManager::FindTemplateForPath

wxDocTemplate * FindTemplateForPath(const wxString& path)

Given a path, try to find template that matches the extension. This is only an approximate method of finding a template for creating a document.

4.52.23 wxDocManager::GetCurrentDocument

wxDocument * GetCurrentDocument()

Returns the document associated with the currently active view (if any).

4.52.24 wxDocManager::GetCurrentView

wxView * GetCurrentView()

Returns the currently active view

4.52.25 wxDocManager::GetDocuments

wxList& GetDocuments()

Returns a reference to the list of documents.

4.52.26 wxDocManager::GetFileHistory

wxFileHistory * GetFileHistory()

Returns a pointer to file history.

4.52.27 wxDocManager::GetMaxDocsOpen

int GetMaxDocsOpen()

Returns the number of documents that can be open simultaneously.

4.52.28 wxDocManager::GetNoHistoryFiles

int GetNoHistoryFiles()

Returns the number of files currently stored in the file history.

4.52.29 wxDocManager::Initialize

bool Initialize()

Initializes data; currently just calls OnCreateFileHistory. Some data cannot always be initialized in the constructor because the programmer must be given the opportunity to override functionality. If OnCreateFileHistory was called from the constructor, an overridden virtual OnCreateFileHistory would not be called due to C++'s 'interesting' constructor semantics. In fact Initialize *is* called from the wxDocManager constructor, but this can be vetoed by passing FALSE to the second argument, allowing the derived class's constructor to call Initialize, possibly calling a different OnCreateFileHistory from the default.

The bottom line: if you're not deriving from `Initialize`, forget it and construct `wxDocManager` with no arguments.

4.52.30 `wxDocManager::MakeDefaultName`

`bool MakeDefaultName(const wxString& buf)`

Copies a suitable default name into *buf*. This is implemented by appending an integer counter to the string **`unnamed`** and incrementing the counter.

4.52.31 `wxDocManager::OnCreateFileHistory`

`wxFileHistory * OnCreateFileHistory()`

A hook to allow a derived class to create a different type of file history. Called from *Initialize* (p. 194).

4.52.32 `wxDocManager::OnFileClose`

`void OnFileClose()`

Closes and deletes the currently active document.

4.52.33 `wxDocManager::OnFileNew`

`void OnFileNew()`

Creates a document from a list of templates (if more than one template).

4.52.34 `wxDocManager::OnFileOpen`

`void OnFileOpen()`

Creates a new document and reads in the selected file.

4.52.35 `wxDocManager::OnFileSave`

`void OnFileSave()`

Saves the current document by calling `wxDocument::Save` for the current document.

4.52.36 `wxDocManager::OnFileSaveAs`

void OnFileSaveAs()

Calls wxDocument::SaveAs for the current document.

4.52.37 wxDocManager::OnMenuCommand**void OnMenuCommand(int cmd)**

Processes menu commands routed from child or parent frames. This deals with the following predefined menu item identifiers:

- wxID_OPEN Creates a new document and opens a file into it.
- wxID_CLOSE Closes the current document.
- wxID_NEW Creates a new document.
- wxID_SAVE Saves the document.
- wxID_SAVE_AS Saves the document into a specified filename.

Unrecognized commands are routed to the currently active wxView's OnMenuCommand.

4.52.38 wxDocManager::RemoveDocument**void RemoveDocument(wxDocument *doc)**

Removes the document from the list of documents.

4.52.39 wxDocManager::SelectDocumentPath

wxDocTemplate * SelectDocumentPath(wxDocTemplate **templates, int noTemplates, const wxString& path, const wxString& bufSize, long flags, bool save)

Under Windows, pops up a file selector with a list of filters corresponding to document templates. The wxDocTemplate corresponding to the selected file's extension is returned.

On other platforms, if there is more than one document template a choice list is popped up, followed by a file selector.

This function is used in wxDocManager::CreateDocument.

4.52.40 wxDocManager::SelectDocumentType

wxDocTemplate * SelectDocumentType(wxDocTemplate **templates, int noTemplates)

Returns a document template by asking the user (if there is more than one template). This function is used in `wxDocManager::CreateDocument`.

4.52.41 `wxDocManager::SelectViewType`

`wxDocTemplate * SelectViewType(wxDocTemplate **templates, int noTemplates)`

Returns a document template by asking the user (if there is more than one template), displaying a list of valid views. This function is used in `wxDocManager::CreateView`. The dialog normally won't appear because the array of templates only contains those relevant to the document in question, and often there will only be one such.

4.52.42 `wxDocManager::SetMaxDocsOpen`

`void SetMaxDocsOpen(int n)`

Sets the maximum number of documents that can be open at a time. By default, this is 10,000. If you set it to 1, existing documents will be saved and deleted when the user tries to open or create a new one (similar to the behaviour of Windows Write, for example). Allowing multiple documents gives behaviour more akin to MS Word and other Multiple Document Interface applications.

4.53 `wxDocMDIChildFrame`

The `wxDocMDIChildFrame` class provides a default frame for displaying documents on separate windows. This class can only be used for MDI child frames.

The class is part of the document/view framework supported by `wxWindows`, and cooperates with the `wxView` (p. 793), `wxDocument` (p. 207), `wxDocManager` (p. 189) and `wxDocTemplate` (p. 202) classes.

See the example application in `samples/docview`.

Derived from

`wxMDIChildFrame` (p. 404)
`wxFrame` (p. 275)
`wxWindow` (p. 798)
`wxEvtHandler` (p. 224)
`wxObject` (p. 471)

Include files

`<wx/docmdi.h>`

See also

Document/view overview (p. 933), `wxMDIChildFrame` (p. 404)

4.53.1 wxDocMDIChildFrame::m_childDocument

wxDocument* m_childDocument

The document associated with the frame.

4.53.2 wxDocMDIChildFrame::m_childView

wxView* m_childView

The view associated with the frame.

4.53.3 wxDocMDIChildFrame::wxDocMDIChildFrame

wxDocMDIChildFrame(wxDocument* doc, wxView* view, wxFrame* parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT_FRAME_STYLE, const wxString& name = "frame")

Constructor.

4.53.4 wxDocMDIChildFrame::~~wxDocMDIChildFrame

~wxDocMDIChildFrame()

Destructor.

4.53.5 wxDocMDIChildFrame::GetDocument

wxDocument* GetDocument() const

Returns the document associated with this frame.

4.53.6 wxDocMDIChildFrame::GetView

wxView* GetView() const

Returns the view associated with this frame.

4.53.7 wxDocMDIChildFrame::OnActivate

void OnActivate(wxActivateEvent event)

Sets the currently active view to be the frame's view. You may need to override (but still call) this function in order to set the keyboard focus for your subwindow.

4.53.8 `wxDocMDIChildFrame::OnCloseWindow`

void OnCloseWindow(`wxCloseEvent& event`)

Closes and deletes the current view and document.

4.53.9 `wxDocMDIChildFrame::SetDocument`

void SetDocument(`wxDocument *doc`)

Sets the document for this frame.

4.53.10 `wxDocMDIChildFrame::SetView`

void SetView(`wxView *view`)

Sets the view for this frame.

4.54 `wxDocMDIParentFrame`

The `wxDocMDIParentFrame` class provides a default top-level frame for applications using the document/view framework. This class can only be used for MDI parent frames.

It cooperates with the `wxView` (p. 793), `wxDocument` (p. 207), `wxDocManager` (p. 189) and `wxDocTemplates` (p. 202) classes.

See the example application in `samples/docview`.

Derived from

`wxMDIParentFrame` (p. 409)
`wxFrame` (p. 275)
`wxWindow` (p. 798)
`wxEvtHandler` (p. 224)
`wxObject` (p. 471)

Include files

`<wx/docmdi.h>`

See also

Document/view overview (p. 933), `wxMDIParentFrame` (p. 409)

4.54.1 wxDocMDIParentFrame::wxDocMDIParentFrame

wxDocParentFrame(wxDocManager* *manager*, wxFrame* *parent*, wxWindowID *id*, const wxString& *title*, const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE, const wxString& *name* = "frame")

Constructor.

4.54.2 wxDocMDIParentFrame::~~wxDocMDIParentFrame

~wxDocMDIParentFrame()

Destructor.

4.54.3 wxDocMDIParentFrame::OnCloseWindow

void OnCloseWindow(wxCloseEvent& *event*)

Deletes all views and documents. If no user input cancelled the operation, the frame will be destroyed and the application will exit.

Since understanding how document/view clean-up takes place can be difficult, the implementation of this function is shown below.

```
void wxDocParentFrame::OnCloseWindow(wxCloseEvent& event)
{
    if (m_docManager->Clear(!event.CanVeto()))
    {
        this->Destroy();
    }
    else
        event.Veto();
}
```

4.55

wxDocParentFrame

The wxDocParentFrame class provides a default top-level frame for applications using the document/view framework. This class can only be used for SDI (not MDI) parent frames.

It cooperates with the *wxView* (p. 793), *wxDocument* (p. 207), *wxDocManager* (p. 189) and *wxDocTemplates* (p. 202) classes.

See the example application in `samples/docview`.

Derived from

wxFrame (p. 275)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/docview.h>

See also

Document/view overview (p. 933), *wxFrame* (p. 275)

4.55.1 wxDocParentFrame::wxDocParentFrame

wxDocParentFrame(*wxDocManager* manager*, *wxFrame* parent*, *wxWindowID id*, **const wxString& title**, **const wxPoint& pos** = *wxDefaultPosition*, **const wxSize& size** = *wxDefaultSize*, **long style** = *wxDEFAULT_FRAME_STYLE*, **const wxString& name** = *"frame"*)

Constructor.

4.55.2 wxDocParentFrame::~~wxDocParentFrame

~wxDocParentFrame()

Destructor.

4.55.3 wxDocParentFrame::OnCloseWindow

void OnCloseWindow(*wxCloseEvent& event*)

Deletes all views and documents. If no user input cancelled the operation, the frame will be destroyed and the application will exit.

Since understanding how document/view clean-up takes place can be difficult, the implementation of this function is shown below.

```
void wxDocParentFrame::OnCloseWindow(wxCloseEvent& event)
{
    if (m_docManager->Clear(!event.CanVeto()))
    {
        this->Destroy();
    }
    else
```

```
        event.Veto();  
    }
```

4.56

wxDocTemplate

The `wxDocTemplate` class is used to model the relationship between a document class and a view class.

Derived from

wxObject (p. 471)

Include files

<wx/docview.h>

See also

wxDocTemplate overview (p. 936), *wxDocument* (p. 207), *wxView* (p. 793)

4.56.1 wxDocTemplate::m_defaultExt

wxString m_defaultExt

The default extension for files of this type.

4.56.2 wxDocTemplate::m_description

wxString m_description

A short description of this template.

4.56.3 wxDocTemplate::m_directory

wxString m_directory

The default directory for files of this type.

4.56.4 wxDocTemplate::m_docClassInfo

wxClassInfo* m_docClassInfo

Run-time class information that allows document instances to be constructed dynamically.

4.56.5 wxDocTemplate::m_docTypeName**wxString m_docTypeName**

The named type of the document associated with this template.

4.56.6 wxDocTemplate::m_documentManager**wxDocTemplate* m_documentManager**

A pointer to the document manager for which this template was created.

4.56.7 wxDocTemplate::m_fileFilter**wxString m_fileFilter**

The file filter (such as *.txt) to be used in file selector dialogs.

4.56.8 wxDocTemplate::m_flags**long m_flags**

The flags passed to the constructor.

4.56.9 wxDocTemplate::m_viewClassInfo**wxClassInfo* m_viewClassInfo**

Run-time class information that allows view instances to be constructed dynamically.

4.56.10 wxDocTemplate::m_viewTypeName**wxString m_viewTypeName**

The named type of the view associated with this template.

4.56.11 wxDocTemplate::wxDocTemplate

wxDocTemplate(wxDocManager* manager, const wxString& descr, const wxString& filter, const wxString& dir, const wxString& ext, const wxString& docTypeName, const wxString& viewTypeName, wxClassInfo* docClassInfo = NULL, wxClassInfo* viewClassInfo = NULL, long flags = wxDEFAULT_TEMPLATE_FLAGS)

Constructor. Create instances dynamically near the start of your application after creating a `wxDocManager` instance, and before doing any document or view operations.

manager is the document manager object which manages this template.

descr is a short description of what the template is for. This string will be displayed in the file filter list of Windows file selectors.

filter is an appropriate file filter such as `*.txt`.

dir is the default directory to use for file selectors.

ext is the default file extension (such as `txt`).

docTypeName is a name that should be unique for a given type of document, used for gathering a list of views relevant to a particular document.

viewTypeName is a name that should be unique for a given view.

docClassInfo is a pointer to the run-time document class information as returned by the `CLASSINFO` macro, e.g. `CLASSINFO(MyDocumentClass)`. If this is not supplied, you will need to derive a new `wxDocTemplate` class and override the `CreateDocument` member to return a new document instance on demand.

viewClassInfo is a pointer to the run-time view class information as returned by the `CLASSINFO` macro, e.g. `CLASSINFO(MyViewClass)`. If this is not supplied, you will need to derive a new `wxDocTemplate` class and override the `CreateView` member to return a new view instance on demand.

flags is a bit list of the following:

- `wxTEMPLATE_VISIBLE` The template may be displayed to the user in dialogs.
- `wxTEMPLATE_INVISIBLE` The template may not be displayed to the user in dialogs.
- `wxDEFAULT_TEMPLATE_FLAGS` Defined as `wxTEMPLATE_VISIBLE`.

4.56.12 `wxDocTemplate::~~wxDocTemplate`

`void ~wxDocTemplate()`

Destructor.

4.56.13 `wxDocTemplate::CreateDocument`

`wxDocument * CreateDocument(const wxString& path, long flags = 0)`

Creates a new instance of the associated document class. If you have not supplied a `wxClassInfo` parameter to the template constructor, you will need to override this

function to return an appropriate document instance.

4.56.14 wxDocTemplate::CreateView

wxView * CreateView(wxDocument *doc, long flags = 0)

Creates a new instance of the associated view class. If you have not supplied a `wxClassInfo` parameter to the template constructor, you will need to override this function to return an appropriate view instance.

4.56.15 wxDocTemplate::GetDefaultExtension

wxString GetDefaultExtension()

Returns the default file extension for the document data, as passed to the document template constructor.

4.56.16 wxDocTemplate::GetDescription

wxString GetDescription()

Returns the text description of this template, as passed to the document template constructor.

4.56.17 wxDocTemplate::GetDirectory

wxString GetDirectory()

Returns the default directory, as passed to the document template constructor.

4.56.18 wxDocTemplate::GetDocumentManager

wxDocManager * GetDocumentManager()

Returns a pointer to the document manager instance for which this template was created.

4.56.19 wxDocTemplate::GetDocumentName

wxString GetDocumentName()

Returns the document type name, as passed to the document template constructor.

4.56.20 wxDocTemplate::GetFileFilter**wxString GetFileFilter()**

Returns the file filter, as passed to the document template constructor.

4.56.21 wxDocTemplate::GetFlags**long GetFlags()**

Returns the flags, as passed to the document template constructor.

4.56.22 wxDocTemplate::GetViewName**wxString GetViewName()**

Returns the view type name, as passed to the document template constructor.

4.56.23 wxDocTemplate::IsVisible**bool IsVisible()**

Returns TRUE if the document template can be shown in user dialogs, FALSE otherwise.

4.56.24 wxDocTemplate::SetDefaultExtension**void SetDefaultExtension(const wxString& ext)**

Sets the default file extension.

4.56.25 wxDocTemplate::SetDescription**void SetDescription(const wxString& descr)**

Sets the template description.

4.56.26 wxDocTemplate::SetDirectory**void SetDirectory(const wxString& dir)**

Sets the default directory.

4.56.27 wxDocTemplate::SetDocumentManager**void SetDocumentManager**(wxDocManager *manager)

Sets the pointer to the document manager instance for which this template was created. Should not be called by the application.

4.56.28 wxDocTemplate::SetFileFilter**void SetFileFilter**(const wxString& filter)

Sets the file filter.

4.56.29 wxDocTemplate::SetFlags**void SetFlags**(long flags)

Sets the internal document template flags (see the constructor description for more details).

4.57 wxDocument

The document class can be used to model an application's file-based data. It is part of the document/view framework supported by wxWindows, and cooperates with the *wxView* (p. 793), *wxDocTemplate* (p. 202) and *wxDocManager* (p. 189) classes.

Derived from*wxEvtHandler* (p. 224)*wxObject* (p. 471)**Include files**

<wx/docview.h>

See also*wxDocument overview* (p. 935), *wxView* (p. 793), *wxDocTemplate* (p. 202), *wxDocManager* (p. 189)**4.57.1 wxDocument::m_commandProcessor****wxCommandProcessor* m_commandProcessor**

A pointer to the command processor associated with this document.

4.57.2 wxDocument::m_documentFile**wxString m_documentFile**

Filename associated with this document ("" if none).

4.57.3 wxDocument::m_documentModified**bool m_documentModified**

TRUE if the document has been modified, FALSE otherwise.

4.57.4 wxDocument::m_documentTemplate**wxDocTemplate * m_documentTemplate**

A pointer to the template from which this document was created.

4.57.5 wxDocument::m_documentTitle**wxString m_documentTitle**

Document title. The document title is used for an associated frame (if any), and is usually constructed by the framework from the filename.

4.57.6 wxDocument::m_documentTypeName**wxString m_documentTypeName**

The document type name given to the wxDocTemplate constructor, copied to this variable when the document is created. If several document templates are created that use the same document type, this variable is used in wxDocManager::CreateView to collate a list of alternative view types that can be used on this kind of document. Do not change the value of this variable.

4.57.7 wxDocument::m_documentViews**wxList m_documentViews**

List of wxView instances associated with this document.

4.57.8 wxDocument::wxDocument

wxDocument()

Constructor. Define your own default constructor to initialize application-specific data.

4.57.9 wxDocument::~~wxDocument**~wxDocument()**

Destructor. Removes itself from the document manager.

4.57.10 wxDocument::AddView**virtual bool AddView(wxView *view)**

If the view is not already in the list of views, adds the view and calls OnChangedViewList.

4.57.11 wxDocument::Close**virtual bool Close()**

Closes the document, by calling OnSaveModified and then (if this returned TRUE) OnCloseDocument. This does not normally delete the document object: use DeleteAllViews to do this implicitly.

4.57.12 wxDocument::DeleteAllViews**virtual bool DeleteAllViews()**

Calls wxView::Close and deletes each view. Deleting the final view will implicitly delete the document itself, because the wxView destructor calls RemoveView. This in turns calls wxDocument::OnChangedViewList, whose default implementation is to save and delete the document if no views exist.

4.57.13 wxDocument::GetCommandProcessor**wxCommandProcessor* GetCommandProcessor() const**

Returns a pointer to the command processor associated with this document.

See *wxCommandProcessor* (p. 108).

4.57.14 wxDocument::GetDocumentTemplate

wxDocTemplate* GetDocumentTemplate() const

Gets a pointer to the template that created the document.

4.57.15 wxDocument::GetDocumentManager**wxDocManager* GetDocumentManager() const**

Gets a pointer to the associated document manager.

4.57.16 wxDocument::GetDocumentName**wxString GetDocumentName() const**

Gets the document type name for this document. See the comment for *documentTypeName* (p. 208).

4.57.17 wxDocument::GetDocumentWindow**wxWindow* GetDocumentWindow() const**

Intended to return a suitable window for using as a parent for document-related dialog boxes. By default, uses the frame associated with the first view.

4.57.18 wxDocument::GetFilename**wxString GetFilename() const**

Gets the filename associated with this document, or "" if none is associated.

4.57.19 wxDocument::GetFirstView**wxView* GetFirstView() const**

A convenience function to get the first view for a document, because in many cases a document will only have a single view.

4.57.20 wxDocument::GetPrintableName**virtual void GetPrintableName(wxString& *name*) const**

Copies a suitable document name into the supplied *name* buffer. The default function uses the title, or if there is no title, uses the filename; or if no filename, the string

unnamed.

4.57.21 wxDocument::GetTitle

wxString GetTitle() const

Gets the title for this document. The document title is used for an associated frame (if any), and is usually constructed by the framework from the filename.

4.57.22 wxDocument::IsModified

virtual bool IsModified() const

Returns TRUE if the document has been modified since the last save, FALSE otherwise. You may need to override this if your document view maintains its own record of being modified (for example if using `wxTextWindow` to view and edit the document).

See also *Modify* (p. 211).

4.57.23 wxDocument::LoadObject

virtual istream& LoadObject(istream& stream)

Override this function and call it from your own `LoadObject` before streaming your own data. `LoadObject` is called by the framework automatically when the document contents need to be loaded.

4.57.24 wxDocument::Modify

virtual void Modify(bool modify)

Call with TRUE to mark the document as modified since the last save, FALSE otherwise. You may need to override this if your document view maintains its own record of being modified (for example if using `wxTextWindow` to view and edit the document).

See also *IsModified* (p. 211).

4.57.25 wxDocument::OnChangedViewList

virtual void OnChangedViewList()

Called when a view is added to or deleted from this document. The default implementation saves and deletes the document if no views exist (the last one has just been removed).

4.57.26 wxDocument::OnCloseDocument**virtual bool OnCloseDocument()**

The default implementation calls `DeleteContents` (an empty implementation) sets the modified flag to `FALSE`. Override this to supply additional behaviour when the document is closed with `Close`.

4.57.27 wxDocument::OnCreate**virtual bool OnCreate(const wxString& path, long flags)**

Called just after the document object is created to give it a chance to initialize itself. The default implementation uses the template associated with the document to create an initial view. If this function returns `FALSE`, the document is deleted.

4.57.28 wxDocument::OnCreateCommandProcessor**virtual wxCommandProcessor* OnCreateCommandProcessor()**

Override this function if you want a different (or no) command processor to be created when the document is created. By default, it returns an instance of `wxCommandProcessor`.

See *wxCommandProcessor* (p. 108).

4.57.29 wxDocument::OnNewDocument**virtual bool OnNewDocument()**

The default implementation calls `OnSaveModified` and `DeleteContents`, makes a default title for the document, and notifies the views that the filename (in fact, the title) has changed.

4.57.30 wxDocument::OnOpenDocument**virtual bool OnOpenDocument(const wxString& filename)**

Constructs an input file stream for the given filename (which must not be empty), and calls `LoadObject`. If `LoadObject` returns `TRUE`, the document is set to unmodified; otherwise, an error message box is displayed. The document's views are notified that the filename has changed, to give windows an opportunity to update their titles. All of the document's views are then updated.

4.57.31 wxDocument::OnSaveDocument**virtual bool OnSaveDocument(const wxString& filename)**

Constructs an output file stream for the given filename (which must not be empty), and calls SaveObject. If SaveObject returns TRUE, the document is set to unmodified; otherwise, an error message box is displayed.

4.57.32 wxDocument::OnSaveModified**virtual bool OnSaveModified()**

If the document has been modified, prompts the user to ask if the changes should be changed. If the user replies Yes, the Save function is called. If No, the document is marked as unmodified and the function succeeds. If Cancel, the function fails.

4.57.33 wxDocument::RemoveView**virtual bool RemoveView(wxView* view)**

Removes the view from the document's list of views, and calls OnChangedViewList.

4.57.34 wxDocument::Save**virtual bool Save()**

Saves the document by calling OnSaveDocument if there is an associated filename, or SaveAs if there is no filename.

4.57.35 wxDocument::SaveAs**virtual bool SaveAs()**

Prompts the user for a file to save to, and then calls OnSaveDocument.

4.57.36 wxDocument::SaveObject**virtual ostream& SaveObject(ostream& stream)**

Override this function and call it from your own SaveObject before streaming your own data. SaveObject is called by the framework automatically when the document contents need to be saved.

4.57.37 wxDocument::SetCommandProcessor

virtual void SetCommandProcessor(wxCommandProcessor *processor)

Sets the command processor to be used for this document. The document will then be responsible for its deletion. Normally you should not call this; override `OnCreateCommandProcessor` instead.

See *wxCommandProcessor* (p. 108).

4.57.38 wxDocument::SetDocumentName

void SetDocumentName(const wxString& name)

Sets the document type name for this document. See the comment for *documentTypeName* (p. 208).

4.57.39 wxDocument::SetDocumentTemplate

void SetDocumentTemplate(wxDocTemplate* templ)

Sets the pointer to the template that created the document. Should only be called by the framework.

4.57.40 wxDocument::SetFilename

void SetFilename(const wxString& filename)

Sets the filename for this document. Usually called by the framework.

4.57.41 wxDocument::SetTitle

void SetTitle(const wxString& title)

Sets the title for this document. The document title is used for an associated frame (if any), and is usually constructed by the framework from the filename.

4.57.42 wxDocument::UpdateAllViews

void UpdateAllViews(wxView* sender = NULL)

Updates all views. If *sender* is non-NULL, does not update this view.

4.58 wxDropFilesEvent

This class is used for drop files events, that is, when files have been dropped onto the

window. This functionality is currently only available under Windows.

Important note: this is a separate implementation to the more general drag and drop implementation documented *here* (p. 975). It uses the older, Windows message-based approach of dropping files.

Derived from

wxEvt (p. 221)

wxObject (p. 471)

Include files

<wx/event.h>

Event table macros

To process a drop files event, use these event handler macros to direct input to a member function that takes a `wxDropFilesEvent` argument.

EVT_DROP_FILES(func) Process a `wxEVT_DROP_FILES` event.

See also

wxWindow::OnDropFiles (p. 820), *Event handling overview* (p. 939)

4.58.1 wxDropFilesEvent::wxDropFilesEvent

wxDropFilesEvent(WXTYPE id = 0, int noFiles = 0, wxString* files = NULL)

Constructor.

4.58.2 wxDropFilesEvent::m_files

wxString* m_files

An array of filenames.

4.58.3 wxDropFilesEvent::m_noFiles

int m_noFiles

The number of files dropped.

4.58.4 wxDropFilesEvent::m_pos

wxPoint m_pos

The point at which the drop took place.

4.58.5 wxDropFilesEvent::GetFiles

wxString* GetFiles() const

Returns an array of filenames.

4.58.6 wxDropFilesEvent::GetNumberOfFiles

int GetNumberOfFiles() const

Returns the number of files dropped.

4.58.7 wxDropFilesEvent::GetPosition

wxPoint GetPosition() const

Returns the position at which the files were dropped.

Returns an array of filenames.

4.59 wxDropSource

Overview (p. 975)

This class represents a source for a drag and drop operation.

Derived from

wxObject (p. 471)

Include files

<wx/dnd.h>

Types

wxDragResult is defined as follows:

```
enum wxDragResult
```



```
wxDragError,      // error prevented the d&d operation from completing
wxDragNone,       // drag target didn't accept the data
wxDragCopy,       // the data was successfully copied
wxDragMove,       // the data was successfully moved
wxDragCancel      // the operation was cancelled by user (not an
error)
;
```

See also

Drag and drop overview (p. 975), *wxDropTarget* (p. 218), *wxTextDropTarget* (p. 726), *wxFileDropTarget* (p. 252)

4.59.1 wxDropSource::wxDropSource

wxDropSource(wxWindow* win = NULL)

Default/wxGTK-specific constructor. If you use the default constructor you must call *wxDropSource::SetData* (p. 217) later.

win is required by wxGTK and therefore should always be set.

wxDropSource(wxDataObject& data, wxWindow* win = NULL)

Parameters

data

A reference to the *data object* (p. 138) associated with the drop source.

win

Only used by wxGTK. TODO

4.59.2 wxDropSource::~~wxDropSource

virtual ~wxDropSource()

4.59.3 wxDropSource::SetData

void SetData(wxDataObject& data)

Sets the data *data object* (p. 138) associated with the drop source.

4.59.4 wxDropSource::DoDragDrop

virtual wxDragResult DoDragDrop(bool *bAllowMove* = FALSE)

Do it (call this in response to a mouse button press, for example).

If **bAllowMove** is FALSE, data can only be copied. Under GTK, data is always copied.

4.59.5 wxDropSource::GiveFeedback

virtual bool GiveFeedback(wxDragResult *effect*, bool *bScrolling*)

Overridable: you may give some custom UI feedback during the drag and drop operation in this function. It is called on each mouse move, so your implementation must not be too slow.

Parameters

effect

The effect to implement. One of wxDragCopy, wxDragMove. MSW only.

bScrolling

TRUE if the window is scrolling. MSW only.

Return value

Return FALSE if you want default feedback, or TRUE if you implement your own feedback.

4.60 wxDropTarget

Overview (p. 975)

This class represents a target for a drag and drop operation.

Derived from

wxObject (p. 471)

Include files

<wx/dnd.h>

See also

Drag and drop overview (p. 975), *wxDropSource* (p. 216), *wxTextDropTarget* (p. 726), *wxFileDropTarget* (p. 252)

4.60.1 wxDropTarget::wxDropTarget**wxDropTarget()**

Constructor.

4.60.2 wxDropTarget::~~wxDropTarget**~wxDropTarget()**

Destructor.

4.60.3 wxDropTarget::GetFormatCount**virtual size_t GetFormatCount() const**

Override this to indicate how many formats you support.

4.60.4 wxDropTarget::GetFormat**virtual wxDataFormat GetFormat(size_t n) const**

Override this to indicate what kind of data you support.

4.60.5 wxDropTarget::OnEnter**virtual void OnEnter()**

Called when the mouse enters the drop target.

4.60.6 wxDropTarget::OnDrop**virtual bool OnDrop(long x, long y, const void* data, size_t size)**

Called when the user drops a data object on the target. Return FALSE to veto the operation.

Parameters

x
The x coordinate of the mouse.

y
The y coordinate of the mouse.

data

The data being dropped.

size

The size of the data being dropped.

Return value

Return TRUE to accept the data, FALSE to veto the operation.

4.60.7 wxDropTarget::OnLeave

virtual void OnLeave()

Called when the mouse leaves the drop target.

4.61 wxEraseEvent

An erase event is sent when a window's background needs to be repainted.

Derived from

wxEvt (p. 221)

wxObject (p. 471)

Include files

<wx/event.h>

Event table macros

To process an erase event, use this event handler macro to direct input to a member function that takes a *wxEraseEvent* argument.

EVT_ERASE_BACKGROUND(func) Process a *wxEVT_ERASE_BACKGROUND* event.

Remarks

If the **m_DC** member is non-NULL, draw into this device context.

See also

wxWindow::OnEraseBackground (p. 821), *Event handling overview* (p. 939)

4.61.1 wxEraseEvent::wxEraseEvent

wxEraseEvent(int *id* = 0, wxDC* *dc* = NULL)

Constructor.

4.61.2 wxEraseEvent::m_dc

wxDC* m_dc

The device context associated with the erase event (may be NULL).

4.61.3 wxEraseEvent::GetDC

wxDC* GetDC() const

Returns the device context to draw into. If this is non-NULL, you should draw into it to perform the erase operation.

4.62 wxEvent

An event is a structure holding information about an event passed to a callback or member function. **wxEvent** used to be a multipurpose event object, and is an abstract base class for other event classes (see below).

Derived from

wxObject (p. 471)

Include files

<wx/event.h>

See also

wxCommandEvent (p. 103), *wxMouseEvent* (p. 450)

4.62.1 wxEvent::wxEvent

wxEvent(int *id* = 0)

Constructor. Should not need to be used directly by an application.

4.62.2 wxEvent::m_eventHandle

char* m_eventHandle

Handle of an underlying windowing system event handle, such as XEvent. Not guaranteed to be instantiated.

4.62.3 wxEvent::m_eventObject**wxObject* m_eventObject**

The object (usually a window) that the event was generated from, or should be sent to.

4.62.4 wxEvent::m_eventType**WXTYPE m_eventType**

The type of the event, such as wxEVENT_TYPE_BUTTON_COMMAND.

4.62.5 wxEvent::m_id**int m_id**

Identifier for the window.

4.62.6 wxEvent::m_skipped**bool m_skipped**

Set to TRUE by **Skip** if this event should be skipped.

4.62.7 wxEvent::m_timeStamp**long m_timeStamp**

Timestamp for this event.

4.62.8 wxEvent::GetEventClass**WXTYPE GetEventClass()**

Returns the identifier of the given event class, such as wxTYPE_MOUSE_EVENT.

4.62.9 wxEvent::GetEventObject

wxObject* GetEventObject()

Returns the object associated with the event, if any.

4.62.10 wxEvent::GetEventType**WXTYPE GetEventType()**

Returns the identifier of the given event type, such as `wxEVENT_TYPE_BUTTON_COMMAND`.

4.62.11 wxEvent::GetId**int GetId()**

Returns the identifier associated with this event, such as a button command id.

4.62.12 wxEvent::GetObjectType**WXTYPE GetObjectType()**

Returns the type of the object associated with the event, such as `wxTYPE_BUTTON`.

4.62.13 wxEvent::GetSkipped**bool GetSkipped()**

Returns `TRUE` if the event handler should be skipped, `FALSE` otherwise.

4.62.14 wxEvent::GetTimestamp**long GetTimestamp()**

Gets the timestamp for the event.

4.62.15 wxEvent::SetEventObject**void SetEventObject(wxObject* object)**

Sets the originating object.

4.62.16 wxEvent::SetEventType

void SetEventType(WXTYPE *typ*)

Sets the event type.

4.62.17 wxEvent::SetId

void SetId(int *id*)

Sets the identifier associated with this event, such as a button command id.

4.62.18 wxEvent::SetTimestamp

void SetTimestamp(long *timeStamp*)

Sets the timestamp for the event.

Sets the originating object.

4.62.19 wxEvent::Skip

void Skip(bool *skip* = TRUE)

Called by an event handler to tell the event system that the event handler should be skipped, and the next valid handler used instead.

4.63 wxEvtHandler

A class that can handle events from the windowing system. `wxWindow` (and therefore all window classes) are derived from this class.

Derived from

`wxObject` (p. 471)

Include files

`<wx/event.h>`

See also

Event handling overview (p. 939)

4.63.1 wxEvtHandler::wxEvtHandler

wxEvtHandler()

Constructor.

4.63.2 wxEvtHandler::~~wxEvtHandler**~wxEvtHandler()**

Destructor. If the handler is part of a chain, the destructor will unlink itself and restore the previous and next handlers so that they point to each other.

4.63.3 wxEvtHandler::Connect

void Connect(int *id*, wxEventType *eventType*, wxObjectEventFunction *function*, wxObject* *userData* = NULL)

void Connect(int *id*, int *lastId*, wxEventType *eventType*, wxObjectEventFunction *function*, wxObject* *userData* = NULL)

Connects the given function dynamically with the event handler, *id* and event type. This is an alternative to the use of static event tables. See the 'dynamic' sample for usage.

Parameters

id

The identifier (or first of the identifier range) to be associated with the event handler function.

lastId

The second part of the identifier range to be associated with the event handler function.

eventType

The event type to be associated with this event handler.

function

The event handler function.

userData

Data to be associated with the event table entry.

Example

```
frame->Connect( wxID_EXIT,
               wxEVT_COMMAND_MENU_SELECTED,
               (wxObjectEventFunction) (wxCommandEventFunction)
               MyFrame::OnQuit );
```

4.63.4 wxEvtHandler::Default

virtual long Default()

Invokes default processing if this event handler is a window.

Return value

System dependent.

Remarks

A generic way of delegating processing to the default system behaviour. It calls a platform-dependent default function, with parameters dependent on the event or message parameters originally sent from the windowing system.

Normally the application should call a base member, such as *wxWindow::OnChar* (p. 817), which itself may call **Default**.

4.63.5 wxEvtHandler::GetClientData

char* GetClientData()

Gets user-supplied client data.

Remarks

Normally, any extra data the programmer wishes to associate with the object should be made available by deriving a new class with new data members.

See also

wxEvtHandler::SetClientData (p. 229)

4.63.6 wxEvtHandler::GetEventHandlerEnabled

bool GetEventHandlerEnabled()

Returns TRUE if the event handler is enabled, FALSE otherwise.

See also

wxEvtHandler::SetEventHandlerEnabled (p. 229)

4.63.7 wxEvtHandler::GetNextHandler

wxEvtHandler* GetNextHandler()

Gets the pointer to the next handler in the chain.

See also

wxEvtHandler::GetPreviousHandler (p. 227), *wxEvtHandler::SetPreviousHandler* (p. 230), *wxEvtHandler::SetNextHandler* (p. 230), *wxWindow::PushEventHandler* (p. 829), *wxWindow::PopEventHandler* (p. 828)

4.63.8 wxEvtHandler::GetPreviousHandler**wxEvtHandler* GetPreviousHandler()**

Gets the pointer to the previous handler in the chain.

See also

wxEvtHandler::GetPreviousHandler (p. 227), *wxEvtHandler::SetPreviousHandler* (p. 230), *wxEvtHandler::SetNextHandler* (p. 230), *wxWindow::PushEventHandler* (p. 829), *wxWindow::PopEventHandler* (p. 828)

4.63.9 wxEvtHandler::ProcessEvent**virtual bool ProcessEvent(wxEvent& event)**

Processes an event, searching event tables and calling zero or more suitable event handler function(s).

Parameters

event
Event to process.

Return value

TRUE if a suitable event handler function was found and executed, and the function did not call *wxEvent::Skip* (p. 224).

Remarks

Normally, your application would not call this function: it is called in the *wxWindows* implementation to dispatch incoming user interface events to the framework (and application).

However, you might need to call it if implementing new functionality (such as a new control) where you define new event types, as opposed to allowing the user to override virtual functions.

An instance where you might actually override the **ProcessEvent** function is where you want to direct event processing to event handlers not normally noticed by wxWindows. For example, in the document/view architecture, documents and views are potential event handlers. When an event reaches a frame, **ProcessEvent** will need to be called on the associated document and view in case event handler functions are associated with these objects. The property classes library (wxProperty) also overrides **ProcessEvent** for similar reasons.

The normal order of event table searching is as follows:

1. If the object is disabled (via a call to *wxEvtHandler::SetEvtHandlerEnabled* (p. 229)) the function skips to step (6).
2. If the object is a wxWindow, **ProcessEvent** is recursively called on the window's *wxValidator* (p. 781). If this returns TRUE, the function exits.
3. **SearchEventTable** is called for this event handler. If this fails, the base class table is tried, and so on until no more tables exist or an appropriate function was found, in which case the function exits.
4. The search is applied down the entire chain of event handlers (usually the chain has a length of one). If this succeeds, the function exits.
5. If the object is a wxWindow and the event is a wxCommandEvent, **ProcessEvent** is recursively applied to the parent window's event handler. If this returns TRUE, the function exits.
6. Finally, **ProcessEvent** is called on the wxApp object.

[See also](#)

wxEvtHandler::SearchEventTable (p. 228)

4.63.10 wxEvtHandler::SearchEventTable

bool SearchEventTable(wxEventTable& table, wxEvent& event)

Searches the event table, executing an event handler function if an appropriate one is found.

Parameters

table

Event table to be searched.

event

Event to be matched against an event table entry.

Return value

TRUE if a suitable event handler function was found and executed, and the function did not call *wxEvent::Skip* (p. 224).

Remarks

This function looks through the object's event table and tries to find an entry that will match the event.

An entry will match if:

1. The event type matches, and
2. the identifier or identifier range matches, or the event table entry's identifier is zero.

If a suitable function is called but calls *wxEvtHandler::Skip* (p. 224), this function will fail, and searching will continue.

See also

wxEvtHandler::ProcessEvent (p. 227)

4.63.11 wxEvtHandler::SetClientData

void SetClientData(char* data)

Sets user-supplied client data.

Parameters

data

Data to be associated with the event handler.

Remarks

Normally, any extra data the programmer wishes to associate with the object should be made available by deriving a new class with new data members. [See also](#)

wxEvtHandler::GetClientData (p. 226)

4.63.12 wxEvtHandler::SetEvtHandlerEnabled

void SetEvtHandlerEnabled(bool enabled)

Enables or disables the event handler.

Parameters

enabled

TRUE if the event handler is to be enabled, FALSE if it is to be disabled.

Remarks

You can use this function to avoid having to remove the event handler from the chain, for example when implementing a dialog editor and changing from edit to test mode.

See also

wxEvtHandler::GetEvtHandlerEnabled (p. 226)

4.63.13 wxEvtHandler::SetNextHandler

void SetNextHandler(*wxEvtHandler* handler*)

Sets the pointer to the next handler.

Parameters

handler

Event handler to be set as the next handler.

See also

wxEvtHandler::GetNextHandler (p. 226), *wxEvtHandler::SetPreviousHandler* (p. 230), *wxEvtHandler::GetPreviousHandler* (p. 227), *wxWindow::PushEventHandler* (p. 829), *wxWindow::PopEventHandler* (p. 828)

4.63.14 wxEvtHandler::SetPreviousHandler

void SetPreviousHandler(*wxEvtHandler* handler*)

Sets the pointer to the previous handler.

Parameters

handler

Event handler to be set as the previous handler.

See also

wxEvtHandler::GetPreviousHandler (p. 227), *wxEvtHandler::SetNextHandler* (p. 230), *wxEvtHandler::GetNextHandler* (p. 226), *wxWindow::PushEventHandler* (p. 829), *wxWindow::PopEventHandler* (p. 828)

4.64 wxExpr

The **wxExpr** class is the building brick of expressions similar to Prolog clauses, or objects. It can represent an expression of type long integer, float, string, word, or list, and lists can be nested.

Derived from

None

Include files

<wx/wxexpr.h>

See also

wxExpr overview (p. 971), *wxExprDatabase* (p. 237)

4.64.1 wxExpr::wxExpr

wxExpr(const wxString& functor)

Construct a new clause with this form, supplying the functor name. A clause is an object that will appear in the data file, with a list of attribute/value pairs.

wxExpr(wxExprType type, const wxString& wordOrString = "")

Construct a new empty list, or a word (will be output with no quotes), or a string, depending on the value of *type*.

type can be **wxExprList**, **wxExprWord**, or **wxExprString**. If *type* is **wxExprList**, the value of *wordOrString* will be ignored.

wxExpr(long value)

Construct an integer expression.

wxExpr(float value)

Construct a floating point expression.

wxExpr(wxList* value)

Construct a list expression. The list's nodes' data should themselves be **wxExprs**.

The current version of this library no longer uses the **wxList** internally, so this constructor turns the list into its internal format (assuming a non-nested list) and then deletes the supplied list.

4.64.2 wxExpr::~wxExpr

~wxExpr()

Destructor.

4.64.3 wxExpr::AddAttributeValue

Use these on clauses ONLY. Note that the functions for adding strings and words must be differentiated by function name which is why they are missing from this group (see *wxExpr::AddAttributeValueString* (p. 232) and *wxExpr::AddAttributeValueWord* (p. 232)).

void AddAttributeValue(const wxString& attribute, float value)

Adds an attribute and floating point value pair to the clause.

void AddAttributeValue(const wxString& attribute, long value)

Adds an attribute and long integer value pair to the clause.

void AddAttributeValue(const wxString& attribute, wxList* value)

Adds an attribute and list value pair to the clause, converting the list into internal form and then deleting **value**. Note that the list should not contain nested lists (except if in internal **wxExpr** form.)

void AddAttributeValue(const wxString& attribute, wxExpr* value)

Adds an attribute and wxExpr value pair to the clause. Do not delete *value* once this function has been called.

4.64.4 wxExpr::AddAttributeValueString

void AddAttributeValueString(const wxString& attribute, const wxString& value)

Adds an attribute and string value pair to the clause.

4.64.5 wxExpr::AddAttributeValueStringList

void AddAttributeValueStringList(const wxString& attribute, wxList* value)

Adds an attribute and string list value pair to the clause.

Note that the list passed to this function is a list of strings, NOT a list of **wxExprs**; it gets turned into a list of **wxExprs** automatically. This is a convenience function, since lists of strings are often manipulated in C++.

4.64.6 wxExpr::AddAttributeValueWord

void AddAttributeValueWord(const wxString& attribute, const wxString& value)

Adds an attribute and word value pair to the clause.

4.64.7 wxExpr::Append

void Append(wxExpr* value)

Append the **value** to the end of the list. 'this' must be a list.

4.64.8 wxExpr::Arg

wxExpr* Arg(wxExprType type, int n) const

Get nth arg of the given clause (starting from 1). NULL is returned if the expression is not a clause, or *n* is invalid, or the given type does not match the actual type. See also *wxExpr::Nth* (p. 235).

4.64.9 wxExpr::Insert

void Insert(wxExpr* value)

Insert the **value** at the start of the list. 'this' must be a list.

4.64.10 wxExpr::GetAttributeValue

These functions are the easiest way to retrieve attribute values, by passing a pointer to variable. If the attribute is present, the variable will be filled with the appropriate value. If not, the existing value is left alone. This style of retrieving attributes makes it easy to set variables to default values before calling these functions; no code is necessary to check whether the attribute is present or not.

bool GetAttributeValue(const wxString& attribute, wxString& value) const

Retrieve a string (or word) value.

bool GetAttributeValue(const wxString& attribute, float& value) const

Retrieve a floating point value.

bool GetAttributeValue(const wxString& attribute, int& value) const

Retrieve an integer value.

bool GetAttributeValue(const wxString& attribute, long& value) const

Retrieve a long integer value.

bool GetAttributeValue(const wxString& attribute, wxExpr value) const**

Retrieve a wxExpr pointer.

4.64.11 wxExpr::GetAttributeValueStringList

void GetAttributeValueStringList(const wxString& *attribute*, wxList* *value*) const

Use this on clauses ONLY. See above for comments on this style of attribute value retrieval. This function expects to receive a pointer to a new list (created by the calling application); it will append strings to the list if the attribute is present in the clause.

4.64.12 wxExpr::AttributeValue

wxExpr* AttributeValue(const wxString& *word*) const

Use this on clauses ONLY. Searches the clause for an attribute matching *word*, and returns the value associated with it.

4.64.13 wxExpr::Copy

wxExpr* Copy() const

Recursively copies the expression, allocating new storage space.

4.64.14 wxExpr::DeleteAttributeValue

void DeleteAttributeValue(const wxString& *attribute*)

Use this on clauses only. Deletes the attribute and its value (if any) from the clause.

4.64.15 wxExpr::Functor

wxString Functor() const

Use this on clauses only. Returns the clause's functor (object name).

4.64.16 wxExpr::GetClientData

wxObject* GetClientData() const

Retrieve arbitrary data stored with this clause. This can be useful when reading in data for storing a pointer to the C++ object, so when another clause makes a reference to this clause, its C++ object can be retrieved. See *wxExpr::SetClientData* (p. 236).

4.64.17 wxExpr::GetFirst**wxExpr* GetFirst() const**

If this is a list expression (or clause), gets the first element in the list.

See also *wxExpr::GetLast* (p. 235), *wxExpr::GetNext* (p. 235), *wxExpr::Nth* (p. 235).

4.64.18 wxExpr::GetLast**wxExpr* GetLast() const**

If this is a list expression (or clause), gets the last element in the list.

See also *wxExpr::GetFirst* (p. 235), *wxExpr::GetNext* (p. 235), *wxExpr::Nth* (p. 235).

4.64.19 wxExpr::GetNext**wxExpr* GetNext() const**

If this is a node in a list (any *wxExpr* may be a node in a list), gets the next element in the list.

See also *wxExpr::GetFirst* (p. 235), *wxExpr::GetLast* (p. 235), *wxExpr::Nth* (p. 235).

4.64.20 wxExpr::IntegerValue**long IntegerValue() const**

Returns the integer value of the expression.

4.64.21 wxExpr::Nth**wxExpr* Nth(int *n*) const**

Get *nth* arg of the given list expression (starting from 0). NULL is returned if the expression is not a list expression, or *n* is invalid. See also *wxExpr::Arg* (p. 233).

Normally, you would use attribute-value pairs to add and retrieve data from objects (clauses) in a data file. However, if the data gets complex, you may need to store attribute values as lists, and pick them apart yourself.

4.64.22 wxExpr::RealValue**float RealValue() const**

Returns the floating point value of the expression.

4.64.23 **wxExpr::SetClientData**

void SetClientData(wxObject *data)

Associate arbitrary data with this clause. This can be useful when reading in data for storing a pointer to the C++ object, so when another clause makes a reference to this clause, its C++ object can be retrieved. See *wxExpr::GetClientData* (p. 234).

4.64.24 **wxExpr::StringValue**

wxString StringValue() const

Returns the string value of the expression.

4.64.25 **wxExpr::Type**

wxExprType Type() const

Returns the type of the expression. **wxExprType** is defined as follows:

```
typedef enum {  
    wxExprNull,  
    wxExprInteger,  
    wxExprReal,  
    wxExprWord,  
    wxExprString,  
    wxExprList  
} wxExprType;
```

4.64.26 **wxExpr::WordValue**

wxString WordValue() const

Returns the word value of the expression.

4.64.27 **wxExpr::WriteLispExpr**

void WriteLispExpr(ostream& stream)

Writes the expression or clause to the given stream in LISP format. Not normally needed, since the whole **wxExprDatabase** will usually be written at once. Lists are enclosed in parentheses with no commas.

4.64.28 wxExpr::WritePrologClause

void WritePrologClause(ostream& stream)

Writes the clause to the given stream in Prolog format. Not normally needed, since the whole **wxExprDatabase** will usually be written at once. The format is: functor, open parenthesis, list of comma-separated expressions, close parenthesis, full stop.

4.64.29 wxExpr::WriteExpr

void WriteExpr(ostream& stream)

Writes the expression (not clause) to the given stream in Prolog format. Not normally needed, since the whole **wxExprDatabase** will usually be written at once. Lists are written in square bracketed, comma-delimited format.

4.64.30 Functions and macros

Below are miscellaneous functions and macros associated with wxExpr objects.

bool wxExprIsFunctor(wxExpr *expr, const wxString& functor)

Checks that the functor of *expr* is *functor*.

void wxExprCleanUp()

Cleans up the wxExpr system (YACC/LEX buffers) to avoid memory-checking warnings as the program exits.

```
#define wxMakeInteger(x) (new wxExpr((long)x))
#define wxMakeReal(x)   (new wxExpr((float)x))
#define wxMakeString(x) (new wxExpr(PrologString, x))
#define wxMakeWord(x)   (new wxExpr(PrologWord, x))
#define wxMake(x)       (new wxExpr(x))
```

Macros to help make wxExpr objects.

4.65 wxExprDatabase

The **wxExprDatabase** class represents a database, or list, of Prolog-like expressions. Instances of this class are used for reading, writing and creating data files.

Derived from

wxList (p. 367)

wxObject (p. 471)

See also

wxExpr overview (p. 971), *wxExpr* (p. 230)

4.65.1 **wxExprDatabase::wxExprDatabase**

void wxExprDatabase(*proioErrorHandler handler = 0*)

Construct a new, unhashed database, with an optional error handler. The error handler must be a function returning a bool and taking an integer and a string argument. When an error occurs when reading or writing a database, this function is called. The error is given as the first argument (currently one of WXEXPR_ERROR_GENERAL, WXEXPR_ERROR_SYNTAX) and an error message is given as the second argument. If FALSE is returned by the error handler, processing of the wxExpr operation stops.

Another way of handling errors is simply to call *wxExprDatabase::GetErrorCount* (p. 240) after the operation, to check whether errors have occurred, instead of installing an error handler. If the error count is more than zero, *wxExprDatabase::Write* (p. 241) and *wxExprDatabase::Read* (p. 240) will return FALSE to the application.

For example:

```
bool myErrorHandler(int err, char *msg)
{
    if (err == WXEXPR_ERROR_SYNTAX)
    {
        wxMessageBox(msg, "Syntax error");
    }
    return FALSE;
}
```

```
wxExprDatabase database(myErrorHandler);
```

wxExprDatabase(*wxExprType type, const wxString&attribute, int size = 500, proioErrorHandler handler = 0*)

Construct a new database hashed on a combination of the clause functor and a named attribute (often an integer identification).

See above for an explanation of the error handler.

4.65.2 **wxExprDatabase::~~wxExprDatabase**

~wxExprDatabase()

Delete the database and contents.

4.65.3 **wxExprDatabase::Append**

void Append(wxExpr* clause)

Append a clause to the end of the database. If the database is hashing, the functor and a user-specified attribute will be hashed upon, giving the option of random access in addition to linear traversal of the database.

4.65.4 wxExprDatabase::BeginFind

void BeginFind()

Reset the current position to the start of the database. Subsequent *wxExprDatabase::FindClause* (p. 239) calls will move the pointer.

4.65.5 wxExprDatabase::ClearDatabase

void ClearDatabase()

Clears the contents of the database.

4.65.6 wxExprDatabase::FindClause

Various ways of retrieving clauses from the database. A return value of NULL indicates no (more) clauses matching the given criteria. Calling the functions repeatedly retrieves more matching clauses, if any.

wxExpr* FindClause(long id)

Find a clause based on the special "id" attribute.

wxExpr* FindClause(const wxString& attribute, const wxString& value)

Find a clause which has the given attribute set to the given string or word value.

wxExpr* FindClause(const wxString& attribute, long value)

Find a clause which has the given attribute set to the given integer value.

wxExpr* FindClause(const wxString& attribute, float value)

Find a clause which has the given attribute set to the given floating point value.

4.65.7 wxExprDatabase::FindClauseByFunctor

wxExpr* FindClauseByFunctor(const wxString& functor)

Find the next clause with the specified functor.

4.65.8 wxExprDatabase::GetErrorCount

int GetErrorCount() const

Returns the number of errors encountered during the last read or write operation.

4.65.9 wxExprDatabase::HashFind

wxExpr* HashFind(const wxString& functor, long value) const

Finds the clause with the given functor and with the attribute specified in the database constructor having the given integer value.

For example,

```
// Hash on a combination of functor and integer "id" attribute when
// reading in
wxExprDatabase db(wxExprInteger, "id");

// Read it in
db.ReadProlog("data");

// Retrieve a clause with specified functor and id
wxExpr *clause = db.HashFind("node", 24);
```

This would retrieve a clause which is written: `node(id = 24, ...,)`.

wxExpr* HashFind(const wxString& functor, const wxString& value)

Finds the clause with the given functor and with the attribute specified in the database constructor having the given string value.

4.65.10 wxExprDatabase::Read

bool Read(const wxString& filename)

Reads in the given file, returning TRUE if successful.

4.65.11 wxExprDatabase::ReadFromString

bool ReadFromString(const wxString& buffer)

Reads a Prolog database from the given string buffer, returning TRUE if successful.

4.65.12 wxExprDatabase::WriteLisp

bool WriteLisp(ostream& stream)

Writes the database as a LISP-format file.

4.65.13 wxExprDatabase::Write

bool Write(ostream& stream)

bool Write(const wxString& filename)

Writes the database as a Prolog-format file.

4.66 wxFile

A wxFile performs raw file I/O. This is a very small class designed to minimize the overhead of using it - in fact, there is hardly any overhead at all, but using it brings you automatic error checking and hides differences between platforms and compilers.

Derived from

None.

Include files

<wx/file.h>

Constants

wx/file.h defines the following constants:

```
#define wxS_IRUSR 00400
#define wxS_IWUSR 00200
#define wxS_IXUSR 00100

#define wxS_IRGRP 00040
#define wxS_IWGRP 00020
#define wxS_IXGRP 00010

#define wxS_IROTH 00004
#define wxS_IWOTH 00002
#define wxS_IXOTH 00001

// default mode for the new files: corresponds to umask 022
#define wxS_DEFAULT (wxS_IRUSR | wxS_IWUSR | wxS_IRGRP | wxS_IWGRP |
wxS_IROTH | wxS_IWOTH)
```

These constants define the file access rights and are used with *wxFile::Create* (p. 243) and *wxFile::Open* (p. 245).

The *OpenMode* enumeration defines the different modes for opening a file, it's defined inside `wxFile` class so its members should be specified with `wxFile::` scope resolution prefix. It is also used with `wxFile::Access` (p. 243) function.

wxFile::read	Open file for reading or test if it can be opened for reading with <code>Access()</code>
wxFile::write	Open file for writing deleting the contents of the file if it already exists or test if it can be opened for writing with <code>Access()</code>
wxFile::read_write	Open file for reading and writing; can not be used with <code>Access()</code>
wxFile::write_append	Open file for appending: the file is opened for writing, but the old contents of the file is not erased and the file pointer is initially placed at the end of the file; can not be used with <code>Access()</code>

Other constants defined elsewhere but used by `wxFile` functions are `wxInvalidOffset` which represents an invalid value of type `off_t` and is returned by functions returning `off_t` on error and the seek mode constants used with `Seek()` (p. 245):

wxFromStart	Count offset from the start of the file
wxFromCurrent	Count offset from the current position of the file pointer
wxFromEnd	Count offset from the end of the file (backwards)

4.66.1 wxFile::wxFile

wxFile()

Default constructor.

wxFile(const char* filename, wxFile::OpenMode mode = wxFile::read)

Opens a file with the given mode. As there is no way to return whether the operation was successful or not from the constructor you should test the return value of `IsOpened` (p. 244) to check that it didn't fail.

wxFile(int fd)

Opens a file with the given file descriptor, which has already been opened.

Parameters

filename

The filename.

mode

The mode in which to open the file. May be one of **wxFile::read**, **wxFile::write** and **wxFile::read_write**.

fd

An existing file descriptor (see *Attach()* (p. 243) for the list of predefined descriptors)

4.66.2 wxFile::~~wxFile

~wxFile()

Destructor will close the file.

NB: it is not virtual so you should *not* derive from wxFile!

4.66.3 wxFile::Access

static bool Access(const char * name, OpenMode mode)

This function verifies if we may access the given file in specified mode. Only values of wxFile::read or wxFile::write really make sense here.

4.66.4 wxFile::Attach

void Attach(int fd)

Attaches an existing file descriptor to the wxFile object. Example of predefined file descriptors are 0, 1 and 2 which correspond to stdin, stdout and stderr (and have symbolic names of wxFile::fd_stdin, wxFile::fd_stdout and wxFile::fd_stderr).

The descriptor should be already opened and it will be closed by wxFile object.

4.66.5 wxFile::Close

void Close()

Closes the file.

4.66.6 wxFile::Create

bool Create(const char* filename, bool overwrite = FALSE, int access = wxS_DEFAULT)

Creates a file for writing. If the file already exists, setting **overwrite** to TRUE will ensure it is overwritten.

4.66.7 wxFile::Detach

void Detach()

Get back a file descriptor from wxFile object - the caller is responsible for closing the file if this descriptor is opened. *IsOpened()* (p. 244) will return FALSE after call to Detach().

4.66.8 wxFile::fd

int fd() const

Returns the file descriptor associated with the file.

4.66.9 wxFile::Eof

bool Eof() const

Returns TRUE if the end of the file has been reached.

4.66.10 wxFile::Exists

static bool Exists(const char* filename)

Returns TRUE if the given name specifies an existing regular file.

4.66.11 wxFile::Flush

bool Flush()

Flushes the file descriptor.

Note that wxFile::Flush is not implemented on some Windows compilers due to a missing fsync function, which reduces the usefulness of this function (it can still be called but it will do nothing on unsupported compilers).

4.66.12 wxFile::IsOpened

bool IsOpened() const

Returns TRUE if the file has been opened.

4.66.13 wxFile::Length**off_t Length() const**

Returns the length of the file.

4.66.14 wxFile::Open**bool Open(const char* filename, wxFile::OpenMode mode = wxFile::read)**

Opens the file, returning TRUE if successful.

Parameters*filename*

The filename.

mode

The mode in which to open the file. May be one of **wxFile::read**, **wxFile::write** and **wxFile::read_write**.

4.66.15 wxFile::Read**off_t Read(void* buffer, off_t count)**

Reads the specified number of bytes into a buffer, returning the actual number read.

Parameters*buffer*

A buffer to receive the data.

count

The number of bytes to read.

Return value

The number of bytes read, or the symbol **wxInvalidOffset** (-1) if there was an error.

4.66.16 wxFile::Seek**off_t Seek(off_t ofs, wxFile::SeekMode mode = wxFile::FromStart)**

Seeks to the specified position.

Parameters

ofs

Offset to seek to.

mode

One of **wxFile::FromStart**, **wxFile::FromEnd**, **wxFile::FromCurrent**.

Return value

The actual offset position achieved, or **wxInvalidOffset** on failure.

4.66.17 **wxFile::SeekEnd**

off_t SeekEnd(off_t ofs = 0)

Moves the file pointer to the specified number of bytes before the end of the file.

Parameters

ofs

Number of bytes before the end of the file.

Return value

The actual offset position achieved, or **wxInvalidOffset** on failure.

4.66.18 **wxFile::Tell**

off_t Tell() const

Returns the current position or **wxInvalidOffset** if file is not opened or if another error occurred.

4.66.19 **wxFile::Write**

bool Write(const void* buffer, off_t count)

Writes the specified number of bytes from a buffer.

Parameters

buffer

A buffer containing the data.

count

The number of bytes to write.

Return value

TRUE if the operation was successful.

4.66.20 wxFile::Write

bool Write(const wxString&s)

Writes the contents of the string to the file, returns TRUE on success.

4.67 wxFileDataObject

wxFileDataObject is a specialization of wxDataObject for file names.

Derived from

wxDataObject (p. 138)

Include files

<wx/dataobj.h>

See also

wxDataObject (p. 138)

4.67.1 wxFileDataObject::wxFileDataObject

wxFileDataObject()

Constructor.

4.67.2 wxFileDataObject::GetFormat

virtual wxDataFormat GetFormat() const

Returns wxDF_FILENAME.

4.67.3 wxFileDataObject::AddFile

virtual void AddFile(const wxString& file)

Adds a filename to the data object.

4.67.4 wxFileDataObject::GetFiles

virtual wxString GetFiles() const

Returns files as a zero-separated list.

4.68 wxFileDialog

This class represents the file chooser dialog.

Derived from

wxDialog (p. 178)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/filedlg.h>

See also

wxFileDialog overview (p. 917), *wxFileSelector* (p. 852)**Remarks**

Pops up a file selector box. In Windows, this is the common file selector dialog. In X, this is a file selector box with somewhat less functionality. The path and filename are distinct elements of a full file pathname. If path is "", the current directory will be used. If filename is "", no default filename will be supplied. The wildcard determines what files are displayed in the file selector, and file extension supplies a type extension for the required filename. Flags may be a combination of wxOPEN, wxSAVE, wxOVERWRITE_PROMPT, wxHIDE_READONLY, or 0. They are only significant at present in Windows.

Both the X and Windows versions implement a wildcard filter. Typing a filename containing wildcards (*, ?) in the filename text item, and clicking on Ok, will result in only those files matching the pattern being displayed. In the X version, supplying no default name will result in the wildcard filter being inserted in the filename text item; the filter is ignored if a default name is supplied.

Under Windows (only), the wildcard may be a specification for multiple types of file with a description for each, such as:

```
"BMP files (*.bmp) | *.bmp | GIF files (*.gif) | *.gif"
```

4.68.1 wxFileDialog::wxFileDialog

wxFileDialog(wxWindow* *parent*, const wxString& *message* = "Choose a file", const wxString& *defaultDir* = "", const wxString& *defaultFile* = "", const wxString& *wildcard* = ".*", long *style* = 0, const wxPoint& *pos* = wxDefaultPosition)

Constructor. Use `wxFileDialog::ShowModal` (p. 251) to show the dialog.

Parameters

parent

Parent window.

message

Message to show on the dialog.

defaultDir

The default directory, or the empty string.

defaultFile

The default filename, or the empty string.

wildcard

A wildcard, such as ".*".

style

A dialog style. A bitlist of:

wxOPEN

This is an open dialog (Windows only).

wxSAVE

This is a save dialog (Windows only).

wxHIDE_READONLY

Hide read-only files (Windows only).

wxOVERWRITE_PROMPT

Prompt for a conformation if a file will be overridden (Windows only).

pos

Dialog position. Not implemented.

4.68.2 wxFileDialog::~wxFileDialog

~wxFileDialog()

Destructor.

4.68.3 wxFileDialog::GetDirectory

wxString GetDirectory() const

Returns the default directory.

4.68.4 wxFileDialog::GetFilename**wxString GetFilename() const**

Returns the default filename.

4.68.5 wxFileDialog::GetFilterIndex**int GetFilterIndex() const**

Returns the index into the list of filters supplied, optionally, in the wildcard parameter. Before the dialog is shown, this is the index which will be used when the dialog is first displayed. After the dialog is shown, this is the index selected by the user.

4.68.6 wxFileDialog::GetMessage**wxString GetMessage() const**

Returns the message that will be displayed on the dialog.

4.68.7 wxFileDialog::GetPath**wxString GetPath() const**

Returns the full path (directory and filename) of the selected file.

4.68.8 wxFileDialog::GetStyle**long GetStyle() const**

Returns the dialog style.

4.68.9 wxFileDialog::GetWildcard**wxString GetWildcard() const**

Returns the file dialog wildcard.

4.68.10 wxFileDialog::SetDirectory**void SetDirectory(const wxString& *directory*)**

Sets the default directory.

4.68.11 wxFileDialog::SetFilename**void SetFilename(const wxString& *setfilename*)**

Sets the default filename.

4.68.12 wxFileDialog::SetFilterIndex**void SetFilterIndex(int *filterIndex*)**

Sets the default filter index, starting from zero. Windows only.

4.68.13 wxFileDialog::SetMessage**void SetMessage(const wxString& *message*)**

Sets the message that will be displayed on the dialog.

4.68.14 wxFileDialog::SetPath**void SetPath(const wxString& *path*)**

Sets the path (the combined directory and filename that will be returned when the dialog is dismissed).

4.68.15 wxFileDialog::SetStyle**void SetStyle(long *style*)**

Sets the dialog style. See *wxFileDialog::wxFileDialog* (p. 248) for details.

4.68.16 wxFileDialog::SetWildcard**void SetWildcard(const wxString& *wildCard*)**

Sets the wildcard, which in Windows can contain multiple file types.

4.68.17 wxFileDialog::ShowModal**int ShowModal()**

Shows the dialog, returning `wxID_OK` if the user pressed OK, and `wxOK_CANCEL`

otherwise.

4.69 wxFileDropTarget

A drop target which accepts files (dragged from File Manager or Explorer).

Derived from

wxDropTarget (p. 218)

Include files

<wx/dnd.h>

See also

Drag and drop overview (p. 975), *wxDropSource* (p. 216), *wxDropTarget* (p. 218), *wxTextDropTarget* (p. 726)

4.69.1 wxFileDropTarget::wxFileDropTarget

wxFileDropTarget()

Constructor.

4.69.2 wxFileDropTarget::GetFormatCount

virtual size_t GetFormatCount()

See *wxDropTarget::GetFormatCount* (p. 219). This function is implemented appropriately for files.

4.69.3 wxFileDropTarget::GetFormat

virtual wxDataFormat GetFormat(size_t n) const

See *wxDropTarget::GetFormat* (p. 219). This function is implemented appropriately for files.

4.69.4 wxFileDropTarget::OnDrop

virtual bool OnDrop(long x, long y, const void *data, size_t size)

See *wxDropTarget::OnDrop* (p. 219). This function is implemented appropriately for files,

and calls *wxFileDropTarget::OnDropFiles* (p. 253).

4.69.5 *wxFileDropTarget::OnDropFiles*

virtual bool OnDropFiles(long x, long y, size_t nFiles, const char * constfiles[])

Override this function to receive dropped files.

Parameters

x
The x coordinate of the mouse.

y
The y coordinate of the mouse.

nFiles
The number of files being dropped.

files
An array of filenames.

Return value

Return TRUE to accept the data, FALSE to veto the operation.

4.70 *wxFileHistory*

The *wxFileHistory* encapsulates a user interface convenience, the list of most recently visited files as shown on a menu (usually the File menu).

wxFileHistory can manage one or more file menus. More than one menu may be required in an MDI application, where the file history should appear on each MDI child menu as well as the MDI parent frame.

Derived from

wxObject (p. 471)

Include files

<wx/docview.h>

See also

wxFileHistory overview (p. 938), *wxDocManager* (p. 189)

4.70.1 wxFileHistory::m_fileHistory**char** m_fileHistory**

A character array of strings corresponding to the most recently opened files.

4.70.2 wxFileHistory::m_fileHistoryN**int m_fileHistoryN**

The number of files stored in the history array.

4.70.3 wxFileHistory::m_fileMaxFiles**int m_fileMaxFiles**

The maximum number of files to be stored and displayed on the menu.

4.70.4 wxFileHistory::m_fileMenu**wxMenu* m_fileMenu**

The file menu used to display the file history list (if enabled).

4.70.5 wxFileHistory::wxFileHistory**wxFileHistory(int maxFiles = 9)**

Constructor. Pass the maximum number of files that should be stored and displayed.

4.70.6 wxFileHistory::~~wxFileHistory**~wxFileHistory()**

Destructor.

4.70.7 wxFileHistory::AddFileToHistory**void AddFileToHistory(const wxString& filename)**

Adds a file to the file history list, if the object has a pointer to an appropriate file menu.

4.70.8 wxFileHistory::AddFilesToMenu

void AddFilesToMenu()

Appends the files in the history list, to all menus managed by the file history object.

void AddFilesToMenu(wxMenu* menu)

Appends the files in the history list, to the given menu only.

4.70.9 wxFileHistory::GetHistoryFile**wxString GetHistoryFile(int index) const**

Returns the file at this index (zero-based).

4.70.10 wxFileHistory::GetMaxFiles**int GetMaxFiles() const**

Returns the maximum number of files that can be stored.

4.70.11 wxFileHistory::GetNoHistoryFiles**int GetNoHistoryFiles() const**

Returns the number of files currently stored in the file history.

4.70.12 wxFileHistory::Load**void Load(wxConfigBase& config)**

Loads the file history from the given config object. This function should be called explicitly by the application.

See also

wxConfig (p. 111)

4.70.13 wxFileHistory::RemoveMenu**void RemoveMenu(wxMenu* menu)**

Removes this menu from the list of those managed by this object.

4.70.14 wxFileHistory::Save**void Save(wxConfigBase& config)**

Saves the file history into the given config object. This must be called explicitly by the application.

See also

wxConfig (p. 111)

4.70.15 wxFileHistory::UseMenu**void UseMenu(wxMenu* menu)**

Adds this menu to the list of those managed by this object.

4.71 wxFileInputStream**Derived from**

wxInputStream (p. 346)

Include files

<wx/wfstream.h>

See also

wxStreamBuffer (p. 648)

4.71.1 wxFileInputStream::wxFileInputStream**wxFileInputStream(const wxString& ifilename)**

Opens the specified file using its *ifilename* name in read-only mode.

wxFileInputStream(wxFile& file)

Initializes a file stream in read-only mode using the file I/O object *file*.

wxFileInputStream(int fd)

Initializes a file stream in read-only mode using the specified file descriptor.

4.71.2 wxFileInputStream::~~wxFileInputStream

~wxFileInputStream()

Destructor.

4.71.3 wxFileInputStream::Ok

bool Ok() const

Returns TRUE if the stream is initialized and ready.

4.72 wxFileOutputStream

Derived from

wxOutputStream (p. 476)

Include files

<wx/wfstream.h>

See also

wxStreamBuffer (p. 648)

4.72.1 wxFileOutputStream::wxFileOutputStream

wxFileOutputStream(const wxString& *ofilename*)

Creates a new file with *ofilename* name and initializes the stream in write-only mode.

wxFileOutputStream(wxFile& *file*)

Initializes a file stream in write-only mode using the file I/O object *file*.

wxFileOutputStream(int *fd*)

Initializes a file stream in write-only mode using the file descriptor *fd*.

4.72.2 wxFileOutputStream::~~wxFileOutputStream

~wxFileOutputStream()

Destructor.

4.72.3 wxFileOutputStream::Ok

bool Ok() const

Returns TRUE if the stream is initialized and ready.

4.73 wxFileStream

Derived from

wxFileOutputStream (p. 257), *wxFileInputStream* (p. 256)

Include files

<wx/wfstream.h>

See also

wxStreamBuffer (p. 648)

4.73.1 wxFileStream::wxFileStream

wxFileStream(const wxString& *iofileName*)

Initializes a new file stream in read-write mode using the specified *iofilename* name.

4.74 wxFileType

This class holds information about a given "file type". File type is the same as MIME type under Unix, but under Windows it corresponds more to an extension than to MIME type (in fact, several extensions may correspond to a file type). This object may be created in several different ways: the program might know the file extension and wish to find out the corresponding MIME type or, conversely, it might want to find the right extension for the file to which it writes the contents of given MIME type. Depending on how it was created some fields may be unknown so the return value of all the accessors **must** be checked: FALSE will be returned if the corresponding information couldn't be found.

The objects of this class are never created by the application code but are returned by *wxMimeTypesManager::GetFileTypeFromMimeType* (p. 445) and *wxMimeTypesManager::GetFileTypeFromExtension* (p. 445) methods. But it's your responsibility to delete the returned pointer when you're done with it!

A brief remainder about what the MIME types are (see the RFC 1341 for more information): basically, it is just a pair category/type (for example, "text/plain") where the category is a basic indication of what a file is (examples of categories are "application",

"image", "text", "binary"...), and type is a precise definition of the document format: "plain" in the example above means just ASCII text without any formatting, while "text/html" is the HTML document source.

A MIME type may have one or more associated extensions: "text/plain" will typically correspond to the extension ".txt", but may as well be associated with ".ini" or ".conf".

Derived from

No base class.

Include files

<wx/mimetype.h>

See also

wxMimeTypesManager (p. 443)

4.74.1 MessageParameters class

One of the most common usages of MIME is to encode an e-mail message. The MIME type of the encoded message is an example of a *message parameter*. These parameters are found in the message headers ("Content-XXX"). At the very least, they must specify the MIME type and the version of MIME used, but almost always they provide additional information about the message such as the original file name or the charset (for the text documents).

These parameters may be useful to the program used to open, edit, view or print the message, so, for example, an e-mail client program will have to pass them to this program. Because `wxFileType` itself can not know about these parameters, it uses `MessageParameters` class to query them. The default implementation only requires the caller to provide the file name (always used by the program to be called - it must know which file to open) and the MIME type and supposes that there are no other parameters. If you wish to supply additional parameters, you must derive your own class from `MessageParameters` and override `GetParamValue()` function, for example:

```
// provide the message parameters for the MIME type manager
class MailMessageParameters : public wxFileType::MessageParameters
{
public:
    MailMessageParameters(const wxString& filename,
                          const wxString& mimetype)
        : wxFileType::MessageParameters(filename, mimetype)
    {
    }

    virtual wxString GetParamValue(const wxString& name) const
    {
        // parameter names are not case-sensitive
    }
}
```

```
        if ( name.CmpNoCase("charset") == 0 )
            return "US-ASCII";
        else
            return wxFileType::MessageParameters::GetParamValue(name);
    }
};
```

Now you only need to create an object of this class and pass it to, for example, *GetOpenCommand* (p. 261) like this:

```
wxString command;
if ( filetype->GetOpenCommand(&command,
                             MailMessageParamaters("foo.txt",
"text/plain"))) )
{
    // the full command for opening the text documents is in 'command'
    // (it might be "notepad foo.txt" under Windows or "cat foo.txt"
under Unix)
}
else
{
    // we don't know how to handle such files...
}
```

Windows: As only the file name is used by the program associated with the given extension anyhow (but no other message parameters), there is no need to ever derive from *MessageParameters* class for a Windows-only program.

4.74.2 wxFileType::wxFileType

wxFileType()

The default constructor is private because you should never create objects of this type: they are only returned by *wxMimeTypeManager* (p. 443) methods.

4.74.3 wxFileType::~~wxFileType

~wxFileType()

The destructor of this class is not virtual, so it should not be derived from.

4.74.4 wxFileType::GetMimeType

bool GetMimeType(wxString* mimeType)

If the function returns TRUE, the string pointed to by *mimeType* is filled with full MIME type specification for this file type: for example, "text/plain".

4.74.5 wxFileType::GetExtensions

bool GetExtensions(wxArrayString& extensions)

If the function returns TRUE, the array *extensions* is filled with all extensions associated with this file type: for example, it may contain the following two elements for the MIME type "text/html" (notice the absence of the leading dot): "html" and "htm".

Windows: This function is currently not implemented: there is no (efficient) way to retrieve associated extensions from the given MIME type on this platform, so it will only return TRUE if the `wxFileType` object was created by *GetFileTypeFromExtension* (p. 445) function in the first place.

4.74.6 wxFileType::GetIcon**bool GetIcon(wxIcon* icon)**

If the function returns TRUE, the icon associated with this file type will be created and assigned to the *icon* parameter.

Unix: This function always returns FALSE under Unix.

4.74.7 wxFileType::GetDescription**bool GetDescription(wxString* desc)**

If the function returns TRUE, the string pointed to by *desc* is filled with a brief description for this file type: for example, "text document" for the "text/plain" MIME type.

4.74.8 wxFileType::GetOpenCommand**bool GetOpenCommand(wxString* command, MessageParameters& params)**

If the function returns TRUE, the string pointed to by *command* is filled with the command which must be executed (see *wxExecute* (p. 865)) in order to open the file of the given type. The name of the file is retrieved from *MessageParameters* (p. 259) class.

4.74.9 wxFileType::GetPrintCommand**bool GetPrintCommand(wxString* command, MessageParameters& params)**

If the function returns TRUE, the string pointed to by *command* is filled with the command which must be executed (see *wxExecute* (p. 865)) in order to print the file of the given type. The name of the file is retrieved from *MessageParameters* (p. 259) class.

4.74.10 wxFileType::ExpandCommand

**static wxString ExpandCommand(const wxString& *command*,
MessageParameters& *params*)**

This function is primarily intended for `GetOpenCommand` and `GetPrintCommand` usage but may be also used by the application directly if, for example, you want to use some non default command to open the file.

The function replaces all occurrences of

format specifier	with
%s	the full file name
%t	the MIME type
%{ <i>param</i> }	the value of the parameter <i>param</i>

using the `MessageParameters` object you pass to it.

If there is no '%s' in the command string (and the string is not empty), it is assumed that the command reads the data on stdin and so the effect is the same as "< %s" were appended to the string.

Unlike all other functions of this class, there is no error return for this function.

4.75 wxFilterInputStream

Derived from

wxInputStream (p. 346)
wxStreamBase (p. 646)

Include files

<wx/stream.h>

Note

The use of this class is exactly the same as of `wxInputStream`. Only a constructor differs and it is documented below.

4.75.1 wxFilterInputStream::wxFilterInputStream

wxFilterInputStream(wxInputStream& *stream*)

Initializes a "filter" stream. A filter stream has the capability of a normal stream but it can be placed on the top of another stream. So, for example, it can uncompress, decrypt the datas which are read from another stream and pass it to the requester.

4.76 wxFilterOutputStream

Derived from

wxOutputStream (p. 476)
wxStreamBase (p. 646)

Include files

<wx/stream.h>

Note

The use of this class is exactly the same as of *wxOutputStream*. Only a constructor differs and it is documented below.

4.76.1 *wxFilterOutputStream::wxFilterOutputStream*

wxFilterOutputStream(wxOutputStream& stream)

Initializes a "filter" stream. A filter stream has the capability of a normal stream but it can be placed on the top of another stream. So, for example, it can compress, crypt the datas which are passed to it and write them to another stream.

4.77 *wxFocusEvent*

A focus event is sent when a window's focus changes.

Derived from

wxEvent (p. 221)
wxObject (p. 471)

Include files

<wx/event.h>

Event table macros

To process a focus event, use these event handler macros to direct input to a member function that takes a *wxFocusEvent* argument.

EVT_SET_FOCUS(func)	Process a <i>wxEVT_SET_FOCUS</i> event.
EVT_KILL_FOCUS(func)	Process a <i>wxEVT_KILL_FOCUS</i> event.

See also

wxWindow::OnSetFocus (p. 827), *wxWindow::OnKillFocus* (p. 822), *Event handling overview* (p. 939)

4.77.1 wxFocusEvent::wxFocusEvent

wxFocusEvent(WXTYPE *eventType* = 0, int *id* = 0)

Constructor.

4.78 wxFont

A font is an object which determines the appearance of text. Fonts are used for drawing text to a device context, and setting the appearance of a window's text.

Derived from

wxGDIObject (p. 295)

wxObject (p. 471)

Include files

<wx/font.h>

Predefined objects

Objects:

wxNullFont

Pointers:

wxNORMAL_FONT

wxSMALL_FONT

wxITALIC_FONT

wxSWISS_FONT

See also

wxFont overview (p. 911), *wxDC::SetFont* (p. 163), *wxDC::DrawText* (p. 157), *wxDC::GetTextExtent* (p. 160), *wxFontDialog* (p. 273)

4.78.1 wxFont::wxFont

wxFont()

Default constructor.

wxFont(int *pointSize*, int *family*, int *style*, int *weight*, const bool *underline* = FALSE,


```
const wxString& faceName = "")
```

Creates a font object.

Parameters

pointSize

Size in points.

family

Font family, a generic way of referring to fonts without specifying actual facename.
One of:

wxDEFAULT	Chooses a default font.
wxDECORATIVE	A decorative font.
wxROMAN	A formal, serif font.
wxSCRIPT	A handwriting font.
wxSWISS	A sans-serif font.
wxMODERN	A fixed pitch font.

style

One of **wxNORMAL**, **wxSLANT** and **wxITALIC**.

weight

One of **wxNORMAL**, **wxLIGHT** and **wxBOLD**.

underline

The value can be TRUE or FALSE. At present this has an effect on Windows only.

faceName

An optional string specifying the actual typeface to be used. If the empty string, a default typeface will chosen based on the family.

Remarks

If the desired font does not exist, the closest match will be chosen. Under Windows, only scaleable TrueType fonts are used.

Underlining only works under Windows at present.

See also *wxDC::SetFont* (p. 163), *wxDC::DrawText* (p. 157) and *wxDC::GetTextExtent* (p. 160).

4.78.2 wxFont::~~wxFont

~wxFont()

Destructor.

Remarks

The destructor may not delete the underlying font object of the native windowing system, since `wxBrush` uses a reference counting system for efficiency.

Although all remaining fonts are deleted when the application exits, the application should try to clean up all fonts itself. This is because `wxWindows` cannot know if a pointer to the font object is stored in an application data structure, and there is a risk of double deletion.

4.78.3 `wxFont::GetFaceName`

`wxString GetFaceName() const`

Returns the typeface name associated with the font, or the empty string if there is no typeface information.

See also

`wxFont::SetFaceName` (p. 267)

4.78.4 `wxFont::GetFamily`

`int GetFamily() const`

Gets the font family. See *`wxFont::wxFont`* (p. 264) for a list of valid family identifiers.

See also

`wxFont::SetFamily` (p. 268)

4.78.5 `wxFont::GetFontId`

`int GetFontId() const`

Returns the font id, if the portable font system is in operation. See *Font overview* (p. 911) for further details.

4.78.6 `wxFont::GetPointSize`

`int GetPointSize() const`

Gets the point size.

See also

wxFont::SetPointSize (p. 268)

4.78.7 **wxFont::GetStyle**

int GetStyle() const

Gets the font style. See *wxFont::wxFont* (p. 264) for a list of valid styles.

[See also](#)

wxFont::SetStyle (p. 268)

4.78.8 **wxFont::GetUnderlined**

bool GetUnderlined() const

Returns TRUE if the font is underlined, FALSE otherwise.

[See also](#)

wxFont::SetUnderlined (p. 269)

4.78.9 **wxFont::GetWeight**

int GetWeight() const

Gets the font weight. See *wxFont::wxFont* (p. 264) for a list of valid weight identifiers.

[See also](#)

wxFont::SetWeight (p. 269)

4.78.10 **wxFont::SetFaceName**

void SetFaceName(const wxString& *faceName*)

Sets the facename for the font.

Parameters

faceName

A valid facename, which should be on the end-user's system.

Remarks

To avoid portability problems, don't rely on a specific face, but specify the font family

instead or as well. A suitable font will be found on the end-user's system. If both the family and the facename are specified, `wxWindows` will first search for the specific face, and then for a font belonging to the same family.

See also

`wxFont::GetFaceName` (p. 266), `wxFont::SetFamily` (p. 268)

4.78.11 `wxFont::SetFamily`

void SetFamily(int *family*)

Sets the font family.

Parameters

family

One of:

wxDEFAULT	Chooses a default font.
wxDECORATIVE	A decorative font.
wxROMAN	A formal, serif font.
wxSCRIPT	A handwriting font.
wxSWISS	A sans-serif font.
wxMODERN	A fixed pitch font.

See also

`wxFont::GetFamily` (p. 266), `wxFont::SetFaceName` (p. 267)

4.78.12 `wxFont::SetPointSize`

void SetPointSize(int *pointSize*)

Sets the point size.

Parameters

pointSize

Size in points.

See also

`wxFont::GetPointSize` (p. 266)

4.78.13 `wxFont::SetStyle`

void SetStyle(int style)

Sets the font style.

Parameters

style

One of **wxNORMAL**, **wxSLANT** and **wxITALIC**.

See also

wxFont::GetStyle (p. 267)

4.78.14 wxFont::SetUnderlined

void SetUnderlined(const bool underlined)

Sets underlining.

Parameters

underlining

TRUE to underline, FALSE otherwise.

See also

wxFont::GetUnderlined (p. 267)

4.78.15 wxFont::SetWeight

void SetWeight(int weight)

Sets the font weight.

Parameters

weight

One of **wxNORMAL**, **wxLIGHT** and **wxBOLD**.

See also

wxFont::GetWeight (p. 267)

4.78.16 wxFont::operator =

wxFont& operator =(const wxFont& font)

Assignment operator, using reference counting. Returns a reference to 'this'.

4.78.17 wxFont::operator ==

bool operator ==(const wxFont& font)

Equality operator. Two fonts are equal if they contain pointers to the same underlying font data. It does not compare each attribute, so two independently-created fonts using the same parameters will fail the test.

4.78.18 wxFont::operator !=

bool operator !=(const wxFont& font)

Inequality operator. Two fonts are not equal if they contain pointers to different underlying font data. It does not compare each attribute.

4.79 wxFontData

wxFontDialog overview (p. 916)

This class holds a variety of information related to font dialogs.

Derived from

wxObject (p. 471)

Include files

<wx/cmndata.h>

See also

Overview (p. 916), *wxFontDialog* (p. 273)

4.79.1 wxFontData::wxFontData

wxFontData()

Constructor. Initializes *fontColour* to black, *showHelp* to black, *allowSymbols* to TRUE, *enableEffects* to TRUE, *minSize* to 0 and *maxSize* to 0.

4.79.2 wxFontData::~~wxFontData

~wxFontData()

Destructor.

4.79.3 wxFontData::EnableEffects**void EnableEffects(bool *enable*)**

Enables or disables 'effects' under MS Windows only. This refers to the controls for manipulating colour, strikeout and underline properties.

The default value is TRUE.

4.79.4 wxFontData::GetAllowSymbols**bool GetAllowSymbols()**

Under MS Windows, returns a flag determining whether symbol fonts can be selected. Has no effect on other platforms.

The default value is TRUE.

4.79.5 wxFontData::GetColour**wxColour& GetColour()**

Gets the colour associated with the font dialog.

The default value is black.

4.79.6 wxFontData::GetChosenFont**wxFont GetChosenFont()**

Gets the font chosen by the user. If the user pressed OK (`wxFontDialog::Show` returned TRUE), this returns a new font which is now 'owned' by the application, and should be deleted if not required. If the user pressed Cancel (`wxFontDialog::Show` returned FALSE) or the colour dialog has not been invoked yet, this will return NULL.

4.79.7 wxFontData::GetEnableEffects**bool GetEnableEffects()**

Determines whether 'effects' are enabled under Windows. This refers to the controls for

manipulating colour, strikethrough and underline properties.

The default value is TRUE.

4.79.8 wxFontData::GetInitialFont

wxFont GetInitialFont()

Gets the font that will be initially used by the font dialog. This should have previously been set by the application.

4.79.9 wxFontData::GetShowHelp

bool GetShowHelp()

Returns TRUE if the Help button will be shown (Windows only).

The default value is FALSE.

4.79.10 wxFontData::SetAllowSymbols

void SetAllowSymbols(bool allowSymbols)

Under MS Windows, determines whether symbol fonts can be selected. Has no effect on other platforms.

The default value is TRUE.

4.79.11 wxFontData::SetChosenFont

void SetChosenFont(const wxFont& font)

Sets the font that will be returned to the user (for internal use only).

4.79.12 wxFontData::SetColour

void SetColour(const wxColour& colour)

Sets the colour that will be used for the font foreground colour.

The default colour is black.

4.79.13 wxFontData::SetInitialFont

void SetInitialFont(const wxFont& font)

Sets the font that will be initially used by the font dialog.

4.79.14 wxFontData::SetRange

void SetRange(int min, int max)

Sets the valid range for the font point size (Windows only).

The default is 0, 0 (unrestricted range).

4.79.15 wxFontData::SetShowHelp

void SetShowHelp(bool showHelp)

Determines whether the Help button will be displayed in the font dialog (Windows only).

The default value is FALSE.

4.79.16 wxFontData::operator =

void operator =(const wxFontData& data)

Assignment operator for the font data.

4.80 wxFontDialog

This class represents the font chooser dialog.

Derived from

wxDialog (p. 178)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/fontdlg.h>

See also

Overview (p. 916), *wxFontData* (p. 270)

4.80.1 wxFontDialog::wxFontDialog

wxFontDialog(wxWindow* *parent*, wxFontData* *data* = NULL)

Constructor. Pass a parent window, and optionally a pointer to a block of font data, which will be copied to the font dialog's font data.

4.80.2 wxFontDialog::~~wxFontDialog

~wxFontDialog()

Destructor.

4.80.3 wxFontDialog::GetFontData

wxFontData& GetFontData()

Returns the *font data* (p. 270) associated with the font dialog.

4.80.4 wxFontDialog::ShowModal

int ShowModal()

Shows the dialog, returning wxID_OK if the user pressed Ok, and wxID_CANCEL otherwise.

If the user cancels the dialog (ShowModal returns wxID_CANCEL), no font will be created. If the user presses OK (ShowModal returns wxID_OK), a new wxFont will be created and stored in the font dialog's wxFontData structure.

4.81 wxFontList

A font list is a list containing all fonts which have been created. There is only one instance of this class: **wxTheFontList**. Use this object to search for a previously created font of the desired type and create it if not already found. In some windowing systems, the font may be a scarce resource, so it is best to reuse old resources if possible. When an application finishes, all fonts will be deleted and their resources freed, eliminating the possibility of 'memory leaks'.

Derived from

wxList (p. 367)

wxObject (p. 471)

Include files

<wx/gdicmn.h>

[See also](#)

wxFont (p. 264)

4.81.1 **wxFontList::wxFontList**

wxFontList()

Constructor. The application should not construct its own font list: use the object pointer **wxTheFontList**.

4.81.2 **wxFontList::AddFont**

void AddFont(wxFont *font)

Used by wxWindows to add a font to the list, called in the font constructor.

4.81.3 **wxFontList::FindOrCreateFont**

wxFont * FindOrCreateFont(int point_size, int family, int style, int weight, bool underline = FALSE, const wxString& facename = NULL)

Finds a font of the given specification, or creates one and adds it to the list. See the *wxFont constructor* (p. 264) for details of the arguments.

4.81.4 **wxFontList::RemoveFont**

void RemoveFont(wxFont *font)

Used by wxWindows to remove a font from the list.

4.82 **wxFrame**

A frame is a window whose size and position can (usually) be changed by the user. It usually has thick borders and a title bar, and can optionally contain a menu bar, toolbar and status bar. A frame can contain any window that is not a frame or dialog.

A frame that has a status bar and toolbar created via the *CreateStatusBar/CreateToolBar* functions manages these windows, and adjusts the value returned by *GetClientSize* to reflect the remaining size available to application windows.

Derived from

wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/frame.h>

Window styles

wxICONIZE	Display the frame iconized (minimized) (Windows only).
wxCAPTION	Puts a caption on the frame.
wxDEFAULT_FRAME_STYLE	Defined as wxMINIMIZE_BOX wxMAXIMIZE_BOX wxTHICK_FRAME wxSYSTEM_MENU wxCAPTION .
wxMINIMIZE	Identical to wxICONIZE .
wxMINIMIZE_BOX	Displays a minimize box on the frame (Windows and Motif only).
wxMAXIMIZE	Displays the frame maximized (Windows only).
wxMAXIMIZE_BOX	Displays a maximize box on the frame (Windows and Motif only).
wxSTAY_ON_TOP	Stay on top of other windows (Windows only).
wxSYSTEM_MENU	Displays a system menu (Windows and Motif only).
wxTHICK_FRAME	Displays a thick frame around the window (Windows and Motif only).
wxRESIZE_BORDER	Displays a resizeable border around the window (Motif only).

See also *window styles overview* (p. 959). Currently the GTK port of *wxWindows* ignores all the window styles listed above as there is no standard way (yet) to inform the window manager about such options. Therefore, the only relevant window style flag which the GTK port recognizes is **wxSIMPLE_BORDER** which brings up a frame without any window decorations. This can be used for a splash screen or specialized tooltip etc.

Remarks

An application should normally define an *OnCloseWindow* (p. 819) handler for the frame to respond to system close events, for example so that related data and subwindows can be cleaned up.

See also

wxMDIParentFrame (p. 409), *wxMDIChildFrame* (p. 404), *wxMiniFrame* (p. 446), *wxDialog* (p. 178)

4.82.1 wxFrame::wxFrame

wxFrame()

Default constructor.

wxFrame(*wxWindow** parent, *wxWindowID* id, **const wxString&** title, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = *wxDEFAULT_FRAME_STYLE*, **const wxString&** name = "frame")

Constructor, creating the window.

Parameters

parent

The window parent. This may be NULL. If it is non-NULL, the frame will always be displayed on top of the parent window on Windows.

id

The window identifier. It may take a value of -1 to indicate a default value.

title

The caption to be displayed on the frame's title bar.

pos

The window position. A value of (-1, -1) indicates a default position, chosen by either the windowing system or wxWindows, depending on platform.

size

The window size. A value of (-1, -1) indicates a default size, chosen by either the windowing system or wxWindows, depending on platform.

style

The window style. See *wxFrame* (p. 275).

name

The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

Remarks

For Motif, MWM (the Motif Window Manager) should be running for any window styles to work (otherwise all styles take effect).

See also

wxFrame::Create (p. 278)

4.82.2 wxFrame::~~wxFrame

void ~wxFrame()

Destructor. Destroys all child windows and menu bar if present.

4.82.3 wxFrame::Centre**void Centre(int direction = wxBOTH)**

Centres the frame on the display.

Parameters

direction

The parameter may be wxHORIZONTAL, wxVERTICAL or wxBOTH.

4.82.4 wxFrame::Command**void Command(int id)**

Simulate a menu command.

Parameters

id

The identifier for a menu item.

4.82.5 wxFrame::Create

bool Create(wxWindow* parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT_FRAME_STYLE, const wxString& name = "frame")

Used in two-step frame construction. See *wxFrame::wxFrame* (p. 277) for further details.

4.82.6 wxFrame::CreateStatusBar

virtual wxStatusBar* CreateStatusBar(int number = 1, long style = 0, wxWindowID id = -1, const wxString& name = "statusBar")

Creates a status bar at the bottom of the frame.

Parameters

number

The number of fields to create. Specify a value greater than 1 to create a multi-field

status bar.

style

The status bar style. See *wxStatusBar* (p. 640) for a list of valid styles.

id

The status bar window identifier. If -1, an identifier will be chosen by *wxWindows*.

name

The status bar window name.

Return value

A pointer to the the status bar if it was created successfully, NULL otherwise.

Remarks

The width of the status bar is the whole width of the frame (adjusted automatically when resizing), and the height and text size are chosen by the host windowing system.

By default, the status bar is an instance of *wxStatusBar*. To use a different class, override *wxFrame::OnCreateStatusBar* (p. 282).

Note that you can put controls and other windows on the status bar if you wish.

See also

wxFrame::SetStatusText (p. 285), *wxFrame::OnCreateStatusBar* (p. 282),
wxFrame::GetStatusBar (p. 280)

4.82.7 *wxFrame::CreateToolBar*

virtual *wxToolBar CreateToolBar(long style = *wxNO_BORDER* |
wxTB_HORIZONTAL, *wxWindowID* id = -1, const *wxString&* name = "toolBar")**

Creates a toolbar at the top or left of the frame.

Parameters

style

The toolbar style. See *wxToolBar* (p. 746) for a list of valid styles.

id

The toolbar window identifier. If -1, an identifier will be chosen by *wxWindows*.

name

The toolbar window name.

Return value

A pointer to the the toolbar if it was created successfully, NULL otherwise.

Remarks

By default, the toolbar is an instance of `wxToolBar` (which is defined to be a suitable toolbar class on each platform, such as `wxToolBar95`). To use a different class, override `wxFrame::OnCreateToolBar` (p. 283).

When a toolbar has been created with this function, or made known to the frame with `wxFrame::SetToolBar` (p. 286), the frame will manage the toolbar position and adjust the return value from `wxWindow::GetClientSize` (p. 807) to reflect the available space for application windows.

See also

`wxFrame::CreateStatusBar` (p. 278), `wxFrame::OnCreateToolBar` (p. 283), `wxFrame::SetToolBar` (p. 286), `wxFrame::GetToolBar` (p. 280)

4.82.8 wxFrame::GetMenuBar

wxMenuBar* GetMenuBar() const

Returns a pointer to the menubar currently associated with the frame (if any).

See also

`wxFrame::SetMenuBar` (p. 285), `wxMenuBar` (p. 426), `wxMenu` (p. 418)

4.82.9 wxFrame::GetStatusBar

wxStatusBar* GetStatusBar()

Returns a pointer to the status bar currently associated with the frame (if any).

See also

`wxFrame::CreateStatusBar` (p. 278), `wxStatusBar` (p. 640)

4.82.10 wxFrame::GetTitle

wxString& GetTitle()

Gets a temporary pointer to the frame title. See `wxFrame::SetTitle` (p. 286).

4.82.11 wxFrame::GetToolBar

wxToolBar* GetToolBar()

Returns a pointer to the toolbar currently associated with the frame (if any).

See also

wxFrame::CreateToolBar (p. 279), *wxToolBar* (p. 746), *wxFrame::SetToolBar* (p. 286)

4.82.12 wxFrame::Iconize**void Iconize(const bool *iconize*)**

Iconizes or restores the frame.

Parameters

iconize

If TRUE, iconizes the frame; if FALSE, shows and restores it.

See also

wxFrame::IsIconized (p. 281), *wxFrame::Maximize* (p. 281).

4.82.13 wxFrame::IsIconized**bool IsIconized() const**

Returns TRUE if the frame is iconized.

4.82.14 wxFrame::IsMaximized**bool IsMaximized() const**

Returns TRUE if the frame is maximized.

4.82.15 wxFrame::Maximize**void Maximize(const bool *maximize*)**

Maximizes or restores the frame.

Parameters

maximize

If TRUE, maximizes the frame, otherwise it restores it

.

Remarks

This function only works under Windows.

See also

wxFrame::Iconize (p. 281)

4.82.16 **wxFrame::OnActivate**

void OnActivate(wxActivateEvent& event)

Called when a window is activated or deactivated (MS Windows only). See also *wxActivateEvent* (p. 5).

4.82.17 **wxFrame::OnCreateStatusBar**

virtual wxStatusBar* OnCreateStatusBar(int number, long style, wxWindowID id, const wxString& name)

Virtual function called when a status bar is requested by *wxFrame::CreateStatusBar* (p. 278).

Parameters

number

The number of fields to create.

style

The window style. See *wxStatusBar* (p. 640) for a list of valid styles.

id

The window identifier. If -1, an identifier will be chosen by wxWindows.

name

The window name.

Return value

A status bar object.

Remarks

An application can override this function to return a different kind of status bar. The default implementation returns an instance of *wxStatusBar* (p. 640).

See also

wxFrame::CreateStatusBar (p. 278), *wxStatusBar* (p. 640).

4.82.18 **wxFrame::OnCreateToolBar**

virtual wxToolBar* OnCreateToolBar(long style, wxWindowID id, const wxString& name)

Virtual function called when a toolbar is requested by *wxFrame::CreateToolBar* (p. 279).

Parameters

style

The toolbar style. See *wxToolBar* (p. 746) for a list of valid styles.

id

The toolbar window identifier. If -1, an identifier will be chosen by wxWindows.

name

The toolbar window name.

Return value

A toolbar object.

Remarks

An application can override this function to return a different kind of toolbar. The default implementation returns an instance of *wxToolBar* (p. 746).

See also

wxFrame::CreateToolBar (p. 279), *wxToolBar* (p. 746).

4.82.19 **wxFrame::OnMenuCommand**

void OnMenuCommand(wxCommandEvent& event)

See *wxWindow::OnMenuCommand* (p. 823).

4.82.20 **wxFrame::OnMenuHighlight**

void OnMenuHighlight(wxMenuEvent& event)

See *wxWindow::OnMenuHighlight* (p. 824).

4.82.21 **wxFrame::OnSize**

void OnSize(wxSizeEvent& event)

See *wxWindow::OnSize* (p. 828).

The default **wxFrame::OnSize** implementation looks for a single subwindow, and if one is found, resizes it to fit inside the frame. Override this member if more complex behaviour is required (for example, if there are several subwindows).

4.82.22 wxFrame::SetIcon

void SetIcon(const wxIcon& icon)

Sets the icon for this frame.

Parameters

icon

The icon to associate with this frame.

Remarks

The frame takes a 'copy' of *icon*, but since it uses reference counting, the copy is very quick. It is safe to delete *icon* after calling this function.

Under Windows, instead of using **SetIcon**, you can add the following lines to your MS Windows resource file:

```
wxSTD_MDIPARENTFRAME ICON icon1.ico
wxSTD_MDICHILDFRAME  ICON icon2.ico
wxSTD_FRAME           ICON icon3.ico
```

where icon1.ico will be used for the MDI parent frame, icon2.ico will be used for MDI child frames, and icon3.ico will be used for non-MDI frames.

If these icons are not supplied, and **SetIcon** is not called either, then the following defaults apply if you have included wx.rc.

```
wxDEFAULT_FRAME           ICON std.ico
wxDEFAULT_MDIPARENTFRAME  ICON mdi.ico
wxDEFAULT_MDICHILDFRAME   ICON child.ico
```

You can replace std.ico, mdi.ico and child.ico with your own defaults for all your wxWindows application. Currently they show the same icon.

Note: a wxWindows application linked with subsystem equal to 4.0 (i.e. marked as a Windows 95 application) doesn't respond properly to **wxFrame::SetIcon**. To work around this until a solution is found, mark your program as a 3.5 application. This will also ensure that Windows provides small icons for the application automatically.

See also *wxIcon* (p. 318).

4.82.23 wxFrame::SetMenuBar**void SetMenuBar(wxMenuBar* menuBar)**

Tells the frame to show the given menu bar.

Parameters*menuBar*

The menu bar to associate with the frame.

Remarks

If the frame is destroyed, the menu bar and its menus will be destroyed also, so do not delete the menu bar explicitly (except by resetting the frame's menu bar to another frame or NULL).

Under Windows, a call to *wxFrame::OnSize* (p. 283) is generated, so be sure to initialize data members properly before calling **SetMenuBar**.

Note that it is not possible to call this function twice for the same frame object.

See also

wxFrame::GetMenuBar (p. 280), *wxMenuBar* (p. 426), *wxMenu* (p. 418).

4.82.24 wxFrame::SetStatusBar**void SetStatusBar(wxStatusBar* statusBar)**

Associates a status bar with the frame.

See also

wxFrame::CreateStatusBar (p. 278), *wxStatusBar* (p. 640), *wxFrame::GetStatusBar* (p. 280)

4.82.25 wxFrame::SetStatusText**virtual void SetStatusText(const wxString& text, int number = 0)**

Sets the status bar text and redraws the status bar.

Parameters*text*

The text for the status field.

number

The status field (starting from zero).

Remarks

Use an empty string to clear the status bar.

See also

wxFrame::CreateStatusBar (p. 278), *wxStatusBar* (p. 640)

4.82.26 **wxFrame::SetStatusWidths**

virtual void SetStatusWidths(int *n*, int **widths*)

Sets the widths of the fields in the status bar.

Parameters

nThe number of fields in the status bar. It must be the same used in *CreateStatusBar* (p. 278).

widths

Must contain an array of *n* integers, each of which is a status field width in pixels. A value of -1 indicates that the field is variable width; at least one field must be -1.

You should delete this array after calling **SetStatusWidths**.

Remarks

The widths of the variable fields are calculated from the total width of all fields, minus the sum of widths of the non-variable fields, divided by the number of variable fields.

4.82.27 **wxFrame::SetToolBar**

void SetToolBar(wxToolBar* *toolBar*)

Associates a toolbar with the frame.

See also

wxFrame::CreateToolBar (p. 279), *wxToolBar* (p. 746), *wxFrame::GetToolBar* (p. 280)

4.82.28 **wxFrame::SetTitle**

virtual void SetTitle(const wxString& *title*)

Sets the frame title.

Parameters

title

The frame title.

See also

wxFrame::GetTitle (p. 280)

4.83 wxFTP

Derived from

wxProtocol (p. 528)

Include files

<wx/protocol/ftp.h>

See also

wxSocketBase (p. 607)

4.83.1 wxFTP::SendCommand

bool SendCommand(const wxString& *command*, char *ret*)

Send the specified *command* to the FTP server. *ret* specifies the expected result.

Return value

TRUE, if the command has been sent successfully, else FALSE.

4.83.2 wxFTP::GetLastResult

const wxString& GetLastResult()

Returns the last command result.

4.83.3 wxFTP::ChDir

bool ChDir(const wxString& *dir*)

Change the current FTP working directory. Returns TRUE if successful.

4.83.4 wxFTP::MkDir

bool MkDir(const wxString& *dir*)

Create the specified directory in the current FTP working directory. Returns TRUE if successful.

4.83.5 wxFTP::Rmdir

bool Rmdir(const wxString& *dir*)

Remove the specified directory from the current FTP working directory. Returns TRUE if successful.

4.83.6 wxFTP::Pwd

wxString Pwd()

Returns the current FTP working directory.

4.83.7 wxFTP::Rename

bool Rename(const wxString& *src*, const wxString& *dst*)

Rename the specified *src* element to *dst*. Returns TRUE if successful.

4.83.8 wxFTP::RmFile

bool RmFile(const wxString& *path*)

Delete the file specified by *path*. Returns TRUE if successful.

4.83.9 wxFTP::SetUser

void SetUser(const wxString& *user*)

Sets the user name to be sent to the FTP server to be allowed to log in.

Default value

The default value of the user name is "anonymous".

Remark

This parameter can be included in a URL if you want to use the URL manager. For example, you can use: "ftp://a_user:a_password@a.host:service/a_directory/a_file" to specify a user and a password.

4.83.10 wxFTP::SetPassword

void SetPassword(const wxString& passwd)

Sets the password to be sent to the FTP server to be allowed to log in.

Default value

The default value of the user name is your email address. For example, it could be "username@userhost.domain". This password is built by getting the current user name and the host name of the local machine from the system.

Remark

This parameter can be included in a URL if you want to use the URL manager. For example, you can use: "ftp://a_user:a_password@a.host:service/a_directory/a_file" to specify a user and a password.

4.83.11 wxFTP::GetList

wxList * GetList(const wxString& wildcard)

The GetList function is quite low-level. It returns the list of the files in the current directory. The list can be filtered using the *wildcard* string. If *wildcard* is a NULL string, it will return all files in directory.

The form of the list can change from one peer system to another. For example, for a UNIX peer system, it will look like this:

```
-r--r--r--  1 guilhem  lavaux      12738 Jan 16 20:17 cmndata.cpp
-r--r--r--  1 guilhem  lavaux      10866 Jan 24 16:41 config.cpp
-rw-rw-rw-  1 guilhem  lavaux      29967 Dec 21 19:17 cwlex_yy.c
-rw-rw-rw-  1 guilhem  lavaux      14342 Jan 22 19:51 cwy_tab.c
-r--r--r--  1 guilhem  lavaux      13890 Jan 29 19:18 date.cpp
-r--r--r--  1 guilhem  lavaux       3989 Feb  8 19:18 datstrm.cpp
```

But on Windows system, it will look like this:

```
winamp~1 exe      520196 02-25-1999  19:28  winamp204.exe
      1 file(s)              520 196 bytes
```

The list is a string list and one node corresponds to a line sent by the peer.

4.83.12 wxFTP::GetOutputStream

wxOutputStream * GetOutputStream(const wxString& file)

Initializes an output stream to the specified *file*. The returned stream has all but the seek functionality of wxStreams. When the user finishes writing data, he has to delete the stream to close it.

Return value

An initialized write-only stream.

See also

wxOutputStream (p. 476)

4.83.13 wxFTP::GetInputStream

wxInputStream * GetInputStream(const wxString& path)

Creates a new input stream on the the specified path. You can use all but seek functionality of wxStream. Seek isn't available on all stream. For example, http or ftp streams doesn't deal with it. Other functions like Tell aren't available for the moment for this sort of stream. You will be notified when the EOF is reached by an error.

Return value

Returns NULL if an error occurred (it could be a network failure or the fact that the file doesn't exist).

Returns the initialized stream. You will have to delete it yourself once you don't use it anymore. The destructor close the DATA stream connection but will leave the COMMAND stream connection opened. It means that you still can send new commands without reconnecting.

Example of a standalone connection (without wxURL)

```
wxFTP ftp;
wxInputStream *in_stream;
char *data;

ftp.Connect("a.host.domain");
ftp.ChDir("a_directory");
in_stream = ftp.GetInputStream("a_file_to_get");

data = new char[in_stream->StreamSize()];

in_stream->Read(data, in_stream->StreamSize());
if (in_stream->LastError() != wxStream_NOERROR) {
    // Do something.
}
```

```
delete in\_stream; /* Close the DATA connection */  
ftp.Close(); /* Close the COMMAND connection */
```

See also

wxInputStream (p. 346)

4.84 wxGauge

A gauge is a horizontal or vertical bar which shows a quantity (often time). There are no user commands for the gauge.

Derived from

wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/gauge.h>

Window styles

wxGA_HORIZONTAL	Creates a horizontal gauge.
wxGA_VERTICAL	Creates a vertical gauge.
wxGA_PROGRESSBAR	Under Windows 95, creates a horizontal progress bar.

See also *window styles overview* (p. 959).

Event handling

wxGauge is read-only so generates no events.

See also

wxSlider (p. 597), *wxScrollBar* (p. 579)

4.84.1 wxGauge::wxGauge

wxGauge()

Default constructor.

```
wxGauge(wxWindow* parent, wxWindowID id, int range, const wxPoint& pos =  
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxGA_HORIZONTAL, const wxValidator& validator = wxDefaultValidator, const  
wxString& name = "gauge")
```

Constructor, creating and showing a gauge.

Parameters

parent

Window parent.

id

Window identifier.

range

Integer range (maximum value) of the gauge.

pos

Window position.

size

Window size.

style

Gauge style. See *wxGauge* (p. 291).

name

Window name.

Remarks

Under Windows 95, there are two different styles of gauge: normal gauge, and progress bar (when the **wxGA_PROGRESSBAR** style is used). A progress bar is always horizontal.

See also

wxGauge::Create (p. 292)

4.84.2 **wxGauge::~~wxGauge**

~wxGauge()

Destructor, destroying the gauge.

4.84.3 **wxGauge::Create**

```
bool Create(wxWindow* parent, wxWindowID id, int range, const wxPoint& pos =
```

wxDefaultPosition, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = *wxGA_HORIZONTAL*, **const wxValidator&** *validator* = *wxDefaultValidator*, **const wxString&** *name* = "gauge")

Creates the gauge for two-step construction. See *wxGauge::wxGauge* (p. 291) for further details.

4.84.4 wxGauge::GetBezelFace

int GetBezelFace() const

Returns the width of the 3D bezel face.

Remarks

Windows only, not for **wxGA_PROGRESSBAR**.

See also

wxGauge::SetBezelFace (p. 294)

4.84.5 wxGauge::GetRange

int GetRange() const

Returns the maximum position of the gauge.

See also

wxGauge::SetRange (p. 294)

4.84.6 wxGauge::GetShadowWidth

int GetShadowWidth() const

Returns the 3D shadow margin width.

Remarks

Windows only, not for **wxGA_PROGRESSBAR**.

See also

wxGauge::SetShadowWidth (p. 294)

4.84.7 wxGauge::GetValue

int GetValue() const

Returns the current position of the gauge.

See also

wxGauge::SetValue (p. 294)

4.84.8 wxGauge::SetBezelFace**void SetBezelFace(int width)**

Sets the 3D bezel face width.

Remarks

Windows only, not for **wxGA_PROGRESSBAR**.

See also

wxGauge::GetBezelFace (p. 293)

4.84.9 wxGauge::SetRange**void SetRange(int range)**

Sets the range (maximum value) of the gauge.

See also

wxGauge::GetRange (p. 293)

4.84.10 wxGauge::SetShadowWidth**void SetShadowWidth(int width)**

Sets the 3D shadow width.

Remarks

Windows only, not for **wxGA_PROGRESSBAR**.

4.84.11 wxGauge::SetValue**void SetValue(int pos)**

Sets the position of the gauge.

Parameters

pos

Position for the gauge level.

See also

wxGauge::GetValue (p. 293)

4.85 wxGDIObject

This class allows platforms to implement functionality to optimise GDI objects, such as *wxPen*, *wxBrush* and *wxFont*. On Windows, the underling GDI objects are a scarce resource and are cleaned up when a usage count goes to zero. On some platforms this class may not have any special functionality.

Since the functionality of this class is platform-specific, it is not documented here in detail.

Derived from

wxObject (p. 471)

Include files

<wx/gdiobj.h>

See also

wxPen (p. 494), *wxBrush* (p. 61), *wxFont* (p. 264)

4.85.1 wxGDIObject::wxGDIObject

wxGDIObject()

Default constructor.

4.86 wxGenericValidator

wxGenericValidator performs data transfer (but not validation or filtering) for the following basic controls: *wxButton*, *wxCheckBox*, *wxListBox*, *wxStaticText*, *wxRadioButton*, *wxRadioBox*, *wxChoice*, *wxComboBox*, *wxGauge*, *wxSlider*, *wxScrollBar*, *wxSpinButton*, *wxTextCtrl*, *wxCheckListBox*.

It checks the type of the window and uses an appropriate type for that window. For example, `wxButton` and `wxTextCtrl` transfer data to and from a `wxString` variable; `wxListBox` uses a `wxArrayInt`; `wxCheckBox` uses a `bool`.

For more information, please see *Validator overview* (p. 969).

Derived from

wxValidator (p. 781)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

`<wx/valgen.h>`

See also

Validator overview (p. 969), *wxValidator* (p. 781), *wxTextValidator* (p. 728)

4.86.1 `wxGenericValidator::wxGenericValidator`

`wxGenericValidator(const wxGenericValidator& validator)`

Copy constructor.

`wxGenericValidator(bool* valPtr)`

Constructor taking a `bool` pointer. This will be used for `wxCheckBox` and `wxRadioButton`.

`wxGenericValidator(wxString* valPtr)`

Constructor taking a `wxString` pointer. This will be used for `wxButton`, `wxComboBox`, `wxStaticText`, `wxTextCtrl`.

`wxGenericValidator(int* valPtr)`

Constructor taking an integer pointer. This will be used for `wxGauge`, `wxScrollBar`, `wxRadioBox`, `wxSpinButton`, `wxChoice`.

`wxGenericValidator(wxArrayInt* valPtr)`

Constructor taking a `wxArrayInt` pointer. This will be used for `wxListBox`, `wxCheckListBox`.

Parameters

validator

Validator to copy.

valPtr

A pointer to a variable that contains the value. This variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).

4.86.2 wxGenericValidator::~~wxGenericValidator

~wxGenericValidator()

Destructor.

4.86.3 wxGenericValidator::Clone

virtual wxValidator* Clone() const

Clones the generic validator using the copy constructor.

4.86.4 wxGenericValidator::TransferFromWindow

virtual bool TransferToWindow(wxWindow* *parent*)

Transfers the value to the window.

4.86.5 wxGenericValidator::TransferToWindow

virtual bool TransferToWindow(wxWindow* *parent*)

Transfers the window value to the appropriate data type.

4.87 wxGrid

wxGrid is a class for displaying and editing tabular information.

Derived from

wxPanel (p. 488)

wxWindow (p. 798)

wxEvtHandler (p. 224)

wxObject (p. 471)

Include files

<wx/grid.h>

Window styles

There are no specific window styles for this class.

See also *window styles overview* (p. 959).

See also

wxGrid classes overview (p. 974)

4.87.1 wxGrid::wxGrid

void wxGrid(wxWindow* parent, wxWindowID id, const wxPoint& pos, const wxSize& size, long style=0, const wxString& name="grid")

Constructor. Before using a wxGrid object, you must call CreateGrid to set up the required rows and columns.

4.87.2 wxGrid::AdjustScrollbars

void AdjustScrollbars()

Call this function whenever a change has been made via the API that might alter the scrollbar characteristics: particularly when adding or deleting rows, or changing row or column dimensions. For example, removing rows might make it unnecessary to show the vertical scrollbar.

4.87.3 wxGrid::AppendCols

bool AppendCols(int n=1, bool updateLabels=TRUE)

Appends *n* columns to the grid. If *updateLabels* is TRUE, the function OnChangeLabels is called to give the application the opportunity to relabel.

4.87.4 wxGrid::AppendRows

bool AppendRows(int n=1, bool updateLabels=TRUE)

Appends *n* rows to the grid. If *updateLabels* is TRUE, the function OnChangeLabels is called to give the application the opportunity to relabel.

4.87.5 wxGrid::BeginBatch

void BeginBatch()

Start a BeginBatch/EndBatch pair between which, calls to SetCellValue or SetCellBitmap will not cause a refresh. This allows you to speed up some operations (for example, setting several hundred cell values). You can nest, but not overlap, these two functions.

See also *wxGrid::EndBatch* (p. 300), *wxGrid::GetBatchCount* (p. 300).

4.87.6 wxGrid::CellHitTest**bool CellHitTest(int x, int y, int *row, int *col)**

Returns TRUE if the x, y panel position coincides with a cell. If so, *row* and *col* are returned.

4.87.7 wxGrid::CreateGrid

bool CreateGrid(int rows, int cols, wxString **cellValues=NULL, short *widths=NULL, short defaultWidth=wxGRID_DEFAULT_CELL_WIDTH, short defaultHeight=wxGRID_DEFAULT_CELL_HEIGHT)

Creates a grid *rows* high and *cols* wide. You can optionally specify an array of initial values and widths, and/or default cell width and height.

Call this function after creating the wxGrid object.

wxPython note:

Currently the *cellValues* and *widths* parameters don't exist in the wxPython version of this method. So in other words, the definition of the wxPython version of this method looks like this:

```
CreateGrid(rows, cols,
           defaultWidth = wxGRID_DEFAULT_CELL_WIDTH,
           defaultHeight = wxGRID_DEFAULT_CELL_HEIGHT)
```

4.87.8 wxGrid::CurrentCellVisible**bool CurrentCellVisible()**

Returns TRUE if the currently selected cell is visible, FALSE otherwise.

4.87.9 wxGrid::DeleteCols

bool DeleteCols(int pos=0, int n=1, bool updateLabels=TRUE)

Deletes n columns from the grid at position pos . If *updateLabels* is TRUE, the function *OnChangeLabels* is called to give the application the opportunity to relabel.

4.87.10 **wxGrid::DeleteRows**

bool DeleteRows(int pos=0, int n=1, bool updateLabels=TRUE)

Deletes n rows from the grid at position pos . If *updateLabels* is TRUE, the function *OnChangeLabels* is called to give the application the opportunity to relabel.

4.87.11 **wxGrid::EndBatch**

void EndBatch()

End a *BeginBatch/EndBatch* pair between which, calls to *SetCellValue* or *SetCellBitmap* will not cause a refresh. This allows you to speed up some operations (for example, setting several hundred cell values). You can nest, but not overlap, these two functions.

See also *wxGrid::BeginBatch* (p. 298), *wxGrid::GetBatchCount* (p. 300).

4.87.12 **wxGrid::GetBatchCount**

int GetBatchCount() const

Return the level of batch nesting. This is initially zero, and will be incremented every time *BeginBatch* is called, and decremented when *EndBatch* is called. When the batch count is more zero, some functions (such as *SetCellValue* and *SetCellBitmap*) will not refresh the cell.

See also *wxGrid::BeginBatch* (p. 298), *wxGrid::EndBatch* (p. 300).

4.87.13 **wxGrid::GetCell**

wxGridCell * GetCell(int row, int col) const

Returns the grid cell object associated with this position.

wxGenericGrid implementation only.

4.87.14 **wxGrid::GetCellAlignment**

int GetCellAlignment(int row, int col) const

int GetCellAlignment() const

Sets the text alignment for the cell at the given position, or the global alignment value. The return value is wxLEFT, wxRIGHT or wxCENTRE.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

GetCellAlignment(row, col)
GetDefCellAlignment()

4.87.15 wxGrid::GetCellBackgroundColour

wxColour& GetCellBackgroundColour(int row, int col) const

wxColour& GetCellBackgroundColour() const

Gets the background colour for the cell at the given position, or the global background colour.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

GetCellBackgroundColour(row, col)
GetDefCellBackgroundColour()

4.87.16 wxGrid::GetCells

wxGridCell * GetCells() const**

Returns the array of grid cell object associated with this wxGrid.

4.87.17 wxGrid::GetCellTextColour

wxColour& GetCellTextColour(int row, int col) const

wxColour& GetCellTextColour() const

Gets the text colour for the cell at the given position, or the global text colour.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

GetCellTextColour(row, col)

GetDefCellTextColour()**4.87.18 wxGrid::GetCellTextFont****const wxFont& GetCellTextFont(int row, int col) const****wxFont& GetCellTextFont() const**

Gets the text font for the cell at the given position, or the global text font.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

GetCellTextFont(row, col)
GetDefCellTextFont()**4.87.19 wxGrid::GetCellValue****wxString& GetCellValue(int row, int col) const**

Returns the cell value at the given position.

4.87.20 wxGrid::GetCols**int GetCols() const**

Returns the number of columns in the grid.

4.87.21 wxGrid::GetColumnWidth**int GetColumnWidth(int col) const**

Gets the width in pixels for column *col*.

4.87.22 wxGrid::GetCurrentRect**wxRectangle * GetCurrentRect() const**

Returns a pointer to the rectangle enclosing the currently selected cell. Do not delete this pointer.

4.87.23 wxGrid::GetCursorColumn**int GetCursorColumn() const**

Returns the column position of the currently selected cell.

4.87.24 wxGrid::GetCursorRow**int GetCursorRow() const**

Returns the row position of the currently selected cell.

4.87.25 wxGrid::GetEditable**bool GetEditable() const**

Returns TRUE if the grid cells can be edited.

4.87.26 wxGrid::GetHorizScrollBar**wxScrollBar * GetHorizScrollBar() const**

Returns a pointer to the horizontal scrollbar.

4.87.27 wxGrid::GetLabelAlignment**int GetLabelAlignment(int *orientation*) const**

Gets the row or column label alignment. *orientation* should be wxHORIZONTAL to specify column label, wxVERTICAL to specify row label. *alignment* should be wxCENTRE, wxLEFT or wxRIGHT.

4.87.28 wxGrid::GetLabelBackgroundColour**wxColour& GetLabelBackgroundColour() const**

Gets a row and column label text colour.

4.87.29 wxGrid::GetLabelSize**int GetLabelSize(int *orientation*) const**

Gets the row label height, or column label width, in pixels. *orientation* should be wxHORIZONTAL to specify column label, wxVERTICAL to specify row label.

4.87.30 wxGrid::GetLabelTextColour**wxColour& GetLabelTextColour() const**

Gets a row and column label text colour.

4.87.31 wxGrid::GetLabelTextFont**wxFont& GetLabelTextFont() const**

Gets the font to be used for the row and column labels.

4.87.32 wxGrid::GetLabelValue**wxString& GetLabelValue(int orientation, int pos) const**

Gets a row or column label value. *orientation* should be wxHORIZONTAL to specify column label, wxVERTICAL to specify row label. *pos* is the label position.

4.87.33 wxGrid::GetRowHeight**int GetRowHeight(int row) const**

Gets the height in pixels for row *row*.

4.87.34 wxGrid::GetRows**int GetRows() const**

Returns the number of rows in the grid.

4.87.35 wxGrid::GetScrollPosX**int GetScrollPosX() const**

Returns the column scroll position.

4.87.36 wxGrid::GetScrollPosY**int GetScrollPosY() const**

Returns the row scroll position.

4.87.37 wxGrid::GetTextItem**wxText * GetTextItem() const**

Returns a pointer to the text item used for entering text into a cell.

4.87.38 wxGrid::GetVertScrollBar**wxScrollBar * GetVertScrollBar() const**

Returns a pointer to the vertical scrollbar.

4.87.39 wxGrid::InsertCols**bool InsertCols(int pos=0, int n=1, bool updateLabels=TRUE)**

Inserts *n* number of columns before position *pos*. If *updateLabels* is TRUE, the function `OnChangeLabels` is called to give the application the opportunity to relabel.

4.87.40 wxGrid::InsertRows**bool InsertRows(int pos=0, int n=1, bool updateLabels=TRUE)**

Inserts *n* number of rows before position *pos*. If *updateLabels* is TRUE, the function `OnChangeLabels` is called to give the application the opportunity to relabel.

4.87.41 wxGrid::OnActivate**void OnActivate(bool active)**

Sets the text item to have the focus. Call this function when the `wxGrid` window should have the focus, for example from `wxFrame::OnActivate`.

4.87.42 wxGrid::OnChangeLabels**void OnChangeLabels()**

Called when rows and columns are created or deleted, to allow the application an opportunity to update the labels. By default, columns are labelled alphabetically, and rows numerically.

4.87.43 wxGrid::OnChangeSelectionLabel

void OnChangeSelectionLabel()

Called when a cell is selected, to allow the application an opportunity to update the selection label (the label of the wxText item used for entering cell text). By default, the cell column letter and row number are concatenated to form the selection label.

4.87.44 wxGrid::OnCreateCell**wxGridCell * OnCreateCell()**

Override this virtual function if you want to replace the normal wxGridCell with a derived class.

4.87.45 wxGrid::OnCellLeftClick**void OnLeftClick(int row, int col, int x, int y, bool control, bool shift)**

Virtual function called when the left button is depressed within a cell, just after OnSelectCell is called.

4.87.46 wxGrid::OnCellRightClick**void OnRightClick(int row, int col, int x, int y, bool control, bool shift)**

Virtual function called when the right button is depressed within a cell, just after OnSelectCell is called.

4.87.47 wxGrid::OnLabelLeftClick**void OnLeftClick(int row, int col, int x, int y, bool control, bool shift)**

Virtual function called when the left button is depressed within a label.

row will be -1 if the click is in the top labels.

col will be -1 if the click is in the left labels.

row and *col* will be -1 if the click is in the upper left corner.

4.87.48 wxGrid::OnLabelRightClick**void OnRightClick(int row, int col, int x, int y, bool control, bool shift)**

Virtual function called when the right button is depressed within a label.

row will be -1 if the click is in the top labels.

col will be -1 if the click is in the left labels.

row and *col* will be -1 if the click is in the upper left corner.

4.87.49 wxGrid::OnSelectCell

void OnSelectCell(int *row*, int *col*)

Virtual function called when the user left-clicks on a cell.

4.87.50 wxGrid::OnSelectCellImplementation

void OnSelectCellImplementation(wxDC **dc*, int *row*, int *col*)

Virtual function called when the user left-clicks on a cell. If you override this function, call `wxGrid::OnSelectCell` to apply the default behaviour.

4.87.51 wxGrid::SetCellAlignment

void SetCellAlignment(int *alignment*, int *row*, int *col*)

void SetCellAlignment(int *alignment*)

Sets the text alignment for the cell at the given position, or for the whole grid. *alignment* may be `wxLEFT`, `wxRIGHT` or `wxCENTRE`.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

SetCellAlignment(*alignment*, *row*, *col*)
SetDefCellAlignment(*alignment*)

4.87.52 wxGrid::SetCellBackgroundColour

void SetCellBackgroundColour(const wxColour& *colour*, int *row*, int *col*)

void SetCellBackgroundColour(const wxColour& *colour*)

Sets the background colour for the cell at the given position, or for the whole grid.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

SetCellBackgroundColour(colour, row, col)
SetDefCellBackgroundColour(colour)

4.87.53 wxGrid::SetCellTextColour

void SetCellTextColour(const wxColour& colour, int row, int col)

void SetCellTextColour(const wxColour& colour)

Sets the text colour for the cell at the given position, or for the whole grid.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

SetCellTextColour(colour, row, col)
SetDefCellTextColour(colour)

4.87.54 wxGrid::SetCellTextFont

void SetCellTextFont(const wxFont& font, int row, int col)

void SetCellTextFont(const wxFont& font)

Sets the text font for the cell at the given position, or for the whole grid.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

SetCellTextFont(font, row, col)
SetDefCellTextFont(font)

4.87.55 wxGrid::SetCellValue

void SetCellValue(const wxString& val, int row, int col)

Sets the cell value at the given position.

4.87.56 wxGrid::SetColumnWidth

void SetColumnWidth(int *col*, int *width*)

Sets the width in pixels for column *col*.

4.87.57 wxGrid::SetDividerPen

void SetDividerPen(const wxPen& *pen*)

Specifies the pen to be used for drawing the divisions between cells. The default is a light grey. If NULL is specified, the divisions will not be drawn.

4.87.58 wxGrid::SetEditable

void SetEditable(bool *editable*)

If *editable* is TRUE (the default), the grid cells will be editable by means of the text edit control. If FALSE, the text edit control will be hidden and the user will not be able to edit the cell contents.

4.87.59 wxGrid::SetGridCursor

void SetGridCursor(int *row*, int *col*)

Sets the position of the selected cell.

4.87.60 wxGrid::SetLabelAlignment

void SetLabelAlignment(int *orientation*, int *alignment*)

Sets the row or column label alignment. *orientation* should be wxHORIZONTAL to specify column label, wxVERTICAL to specify row label. *alignment* should be wxCENTRE, wxLEFT or wxRIGHT.

4.87.61 wxGrid::SetLabelBackgroundColour

void SetLabelBackgroundColour(const wxColour& *value*)

Sets a row or column label background colour.

4.87.62 wxGrid::SetLabelSize

void SetLabelSize(int *orientation*, int *size*)

Sets the row label height, or column label width, in pixels. *orientation* should be wxHORIZONTAL to specify column label, wxVERTICAL to specify row label.

If a dimension of zero is specified, the row or column labels will not be shown.

4.87.63 wxGrid::SetLabelTextColour

void SetLabelTextColour(const wxColour& value)

Sets a row and column label text colour.

4.87.64 wxGrid::SetLabelTextFont

void SetLabelTextFont(const wxFont& font)

Sets the font to be used for the row and column labels.

4.87.65 wxGrid::SetLabelValue

void SetLabelValue(int orientation, const wxString& value, int pos)

Sets a row or column label value. *orientation* should be wxHORIZONTAL to specify column label, wxVERTICAL to specify row label. *pos* is the label position.

4.87.66 wxGrid::SetRowHeight

void SetRowHeight(int row, int height)

Sets the height in pixels for row *row*.

4.87.67 wxGrid::UpdateDimensions

void UpdateDimensions()

Call this function whenever a change has been made via the API that might alter size characteristics. You may also need to follow it with a call to AdjustScrollbars.

4.88 wxHashTable

This class provides hash table functionality for wxWindows, and for an application if it wishes. Data can be hashed on an integer or string key.

Derived from

wxObject (p. 471)

Include files

<wx/hash.h>

Example

Below is an example of using a hash table.

```
wxHashTable table(KEY_STRING);

wxPoint *point = new wxPoint(100, 200);
table.Put("point 1", point);

....

wxPoint *found_point = (wxPoint *)table.Get("point 1");
```

A hash table is implemented as an array of pointers to lists. When no data has been stored, the hash table takes only a little more space than this array (default size is 1000). When a data item is added, an integer is constructed from the integer or string key that is within the bounds of the array. If the array element is NULL, a new (keyed) list is created for the element. Then the data object is appended to the list, storing the key in case other data objects need to be stored in the list also (when a 'collision' occurs).

Retrieval involves recalculating the array index from the key, and searching along the keyed list for the data object whose stored key matches the passed key. Obviously this is quicker when there are fewer collisions, so hashing will become inefficient if the number of items to be stored greatly exceeds the size of the hash table.

See also

wxList (p. 367)

4.88.1 wxHashTable::wxHashTable

wxHashTable(unsigned int *key_type*, int *size* = 1000)

Constructor. *key_type* is one of wxKEY_INTEGER, or wxKEY_STRING, and indicates what sort of keying is required. *size* is optional.

4.88.2 wxHashTable::~~wxHashTable

~wxHashTable()

Destroys the hash table.

4.88.3 wxHashTable::BeginFind

void BeginFind()

The counterpart of *Next*. If the application wishes to iterate through all the data in the hash table, it can call *BeginFind* and then loop on *Next*.

4.88.4 wxHashTable::Clear

void Clear()

Clears the hash table of all nodes (but as usual, doesn't delete user data).

4.88.5 wxHashTable::Delete

wxObject * Delete(long key)

wxObject * Delete(const wxString& key)

Deletes entry in hash table and returns the user's data (if found).

4.88.6 wxHashTable::Get

wxObject * Get(long key)

wxObject * Get(const wxString& key)

Gets data from the hash table, using an integer or string key (depending on which hash table constructor was used).

4.88.7 wxHashTable::MakeKey

long MakeKey(const wxString& string)

Makes an integer key out of a string. An application may wish to make a key explicitly (for instance when combining two data values to form a key).

4.88.8 wxHashTable::Next

wxNode * Next()

If the application wishes to iterate through all the data in the hash table, it can call *BeginFind* and then loop on *Next*. This function returns a **wxNode** pointer (or NULL if there are no more nodes). See the description for *wxNode* (p. 463). The user will probably only wish to use the **wxNode::Data** function to retrieve the data; the node may

also be deleted.

4.88.9 wxHashTable::Put

void Put(long *key*, wxObject **object*)

void Put(const wxString& *key*, wxObject **object*)

Inserts data into the hash table, using an integer or string key (depending on which hash table constructor was used). The key string is copied and stored by the hash table implementation.

4.89 wxHelpController

This is a family of classes by which applications may invoke a help viewer to provide on-line help.

A help controller allows an application to display help, at the contents or at a particular topic, and shut the help program down on termination. This avoids proliferation of many instances of the help viewer whenever the user requests a different topic via the application's menus or buttons.

Typically, an application will create a help controller instance when it starts, and immediately call **Initialize** to associate a filename with it. The help viewer will only get run, however, just before the first call to display something.

Although all help controller classes actually derive from wxHelpControllerBase and have names of the form wxXXXHelpController, the appropriate class is aliased to the name wxHelpController for each platform.

There are currently the following help controller classes defined:

- wxWinHelpController, for controlling Windows Help.
- wxExtHelpController, for controlling external browsers under Unix. The default browser is Netscape Navigator.
- wxXLPHelpController, for controlling wxHelp (from wxWindows 1).

Derived from

wxHelpControllerBase
wxObject (p. 471)

Include files

<wx/help.h> (wxWindows chooses the appropriate help controller class)
<wx/helpbase.h> (wxHelpControllerBase class)
<wx/helpwin.h> (Windows Help controller)
<wx/generic/helpext.h> (external HTML browser controller) <wx/generic/helpxlp.h> (wxHelp controller)

4.89.1 wxHelpController::wxHelpController

wxHelpController()

Constructs a help instance object, but does not invoke the help viewer.

4.89.2 wxHelpController::~~wxHelpController

~wxHelpController()

Destroys the help instance, closing down the viewer if it is running.

4.89.3 wxHelpController::Initialize

virtual void Initialize(const wxString& file)

virtual void Initialize(const wxString& file, int server)

Initializes the help instance with a help filename, and optionally a server (socket) number if using wxHelp. Does not invoke the help viewer. This must be called directly after the help instance object is created and before any attempts to communicate with the viewer.

You may omit the file extension and a suitable one will be chosen.

4.89.4 wxHelpController::DisplayBlock

virtual bool DisplayBlock(long blockNo)

If the help viewer is not running, runs it and displays the file at the given block number.

wxHelp: this is the wxHelp block number.

WinHelp: Refers to the context number.

External HTML help: the same as for *wxHelpController::DisplaySection* (p. 315).

4.89.5 wxHelpController::DisplayContents

virtual bool DisplayContents()

If the help viewer is not running, runs it and displays the contents.

4.89.6 wxHelpController::DisplaySection

virtual bool DisplaySection(int sectionNo)

If the help viewer is not running, runs it and displays the given section.

wxHelp: Sections are numbered starting from 1. Section numbers may be viewed by running wxHelp in edit mode.

WinHelp: *sectionNo* is a context id.

External HTML help: wxExtHelpController implements *sectionNo* as an id in a map file, which is of the form:

```
0  wx.html           ; Index
1  wx34.html#classref ; Class reference
2  wx204.html        ; Function reference
```

4.89.7 wxHelpController::KeywordSearch

virtual bool KeywordSearch(const wxString& keyWord)

If the help viewer is not running, runs it, and searches for sections matching the given keyword. If one match is found, the file is displayed at this section.

wxHelp: If more than one match is found, the Search dialog is displayed with the matches.

WinHelp: If more than one match is found, the first topic is displayed.

External HTML help: If more than one match is found, a choice of topics is displayed.

4.89.8 wxHelpController::LoadFile

virtual bool LoadFile(const wxString& file = "")

If the help viewer is not running, runs it and loads the given file. If the filename is not supplied or is NULL, the file specified in **Initialize** is used. If the viewer is already displaying the specified file, it will not be reloaded. This member function may be used before each display call in case the user has opened another file.

4.89.9 wxHelpController::SetViewer

virtual void SetViewer(const wxString& viewer, long flags)

Sets detailed viewer information. So far this is only relevant to wxExtHelpController.

Parameters

viewer

This defaults to "netscape" for `wxExtHelpController`.

flags

This defaults to `wxHELP_NETSCAPE` for `wxExtHelpController`, indicating that the viewer is a variant of Netscape Navigator.

4.89.10 `wxHelpController::OnQuit`

virtual bool OnQuit()

Overridable member called when this application's viewer is quit by the user.

This does not work for all help controllers.

4.89.11 `wxHelpController::Quit`

virtual bool Quit()

If the viewer is running, quits it by disconnecting.

For Windows Help, the viewer will only close if no other application is using it.

4.90 `wxHTTP`

Derived from

wxProtocol (p. 528)

Include files

<wx/protocol/http.h>

See also

wxSocketBase (p. 607), *wxURL* (p. 779)

4.90.1 `wxHTTP::GetInputStream`

wxInputStream * GetInputStream(const wxString& path)

Creates a new input stream on the the specified path. You can use all except the seek functionality of `wxStream`. Seek isn't available on all streams. For example, http or ftp streams doesn't deal with it. Other functions like `StreamSize` and `Tell` aren't available for the moment for this sort of stream. You will be notified when the EOF is reached by an

error.

Return value

Returns the initialized stream. You will have to delete it yourself once you don't use it anymore. The destructor closes the network connection. The next time you will try to get a file the network connection will have to be reestablished: but you don't have to take care of this wxHTTP reestablishes it automatically.

See also

wxInputStream (p. 346)

4.90.2 wxHTTP::SetHeader

void SetHeader(const wxString& header, const wxString& h_data)

It sets data of a field to be sent during the next request to the HTTP server. The field name is specified by *header* and the content by *h_data*. This is a low level function and it assumes that you know what you are doing.

4.90.3 wxHTTP::SetHeader

wxString GetHeader(const wxString& header)

Returns the data attached with a field whose name is specified by *header*. If the field doesn't exist, it will return an empty string and not a NULL string.

4.91 wxIdleEvent

This class is used for idle events, which are generated when the system is idle.

Derived from

wxEvent (p. 221)

wxObject (p. 471)

Include files

<wx/event.h>

Event table macros

To process an idle event, use this event handler macro to direct input to a member function that takes a wxIdleEvent argument.

EVT_IDLE(func) Process a wxEVT_IDLE event.

Remarks

Idle events can be caught by the `wxApp` class, or by top-level window classes.

See also

wxApp::OnIdle (p. 11), *Event handling overview* (p. 939)

4.91.1 `wxIdleEvent::wxIdleEvent`

`wxIdleEvent()`

Constructor.

4.91.2 `wxIdleEvent::RequestMore`

`void RequestMore(bool needMore = TRUE)`

Tells `wxWindows` that more processing is required. This function can be called by an `OnIdle` handler for a window or window event handler to indicate that `wxApp::OnIdle` should forward the `OnIdle` event once more to the application windows. If no window calls this function during `OnIdle`, then the application will remain in a passive event loop (not calling `OnIdle`) until a new event is posted to the application by the windowing system.

See also

wxIdleEvent::MoreRequested (p. 318), *wxApp::OnIdle* (p. 11)

4.91.3 `wxIdleEvent::MoreRequested`

`bool MoreRequested() const`

Returns `TRUE` if the `OnIdle` function processing this event requested more processing time.

See also

wxIdleEvent::RequestMore (p. 318), *wxApp::OnIdle* (p. 11)

4.92 `wxIcon`

An icon is a small rectangular bitmap usually used for denoting a minimized application. It differs from a `wxBitmap` in always having a mask associated with it for transparent

drawing. On some platforms, icons and bitmaps are implemented identically, since there is no real distinction between a `wxBitmap` with a mask and an icon; and there is no specific icon format on some platforms (X-based applications usually standardize on XPMs for small bitmaps and icons). However, some platforms (such as Windows) make the distinction, so a separate class is provided.

Derived from

wxBitmap (p. 39)
wxGDIObject (p. 295)
wxObject (p. 471)

Include files

<wx/icon.h>

Predefined objects

Objects:

wxNullIcon

Remarks

It is usually desirable to associate a pertinent icon with a frame. Icons can also be used for other purposes, for example with *wxTreeCtrl* (p. 761) and *wxListCtrl* (p. 381).

Icons have different formats on different platforms. Therefore, separate icons will usually be created for the different environments. Platform-specific methods for creating a **wxIcon** structure are catered for, and this is an occasion where conditional compilation will probably be required.

Note that a new icon must be created for every time the icon is to be used for a new window. In Windows, the icon will not be reloaded if it has already been used. An icon allocated to a frame will be deleted when the frame is deleted.

For more information please see *Bitmap and icon overview* (p. 908).

See also

Bitmap and icon overview (p. 908), *supported bitmap file formats* (p. 908),
wxDC::DrawIcon (p. 155), *wxCursor* (p. 128)

4.92.1 wxIcon::wxIcon

wxIcon()

Default constructor.

wxlIcon(const wxIcon& icon)

Copy constructor.

wxlIcon(void* data, int type, int width, int height, int depth = -1)

Creates an icon from the given data, which can be of arbitrary type.

**wxlIcon(const char bits[], int width, int height
int depth = 1)**

Creates an icon from an array of bits.

wxlIcon(int width, int height, int depth = -1)

Creates a new icon.

wxlIcon(const char bits)**

Creates an icon from XPM data.

wxlIcon(const wxString& name, long type, int desiredWidth = -1, int desiredHeight = -1)

Loads an icon from a file or resource.

Parameters

bits

Specifies an array of pixel values.

width

Specifies the width of the icon.

height

Specifies the height of the icon.

desiredWidth

Specifies the desired width of the icon. This parameter only has an effect in Windows (32-bit) where icon resources can contain several icons of different sizes.

desiredHeight

Specifies the desired height of the icon. This parameter only has an effect in Windows (32-bit) where icon resources can contain several icons of different sizes.

depth

Specifies the depth of the icon. If this is omitted, the display depth of the screen is used.

name

This can refer to a resource name under MS Windows, or a filename under MS Windows and X. Its meaning is determined by the *flags* parameter.

type

May be one of the following:

wxBITMAP_TYPE_ICO	Load a Windows icon file.
wxBITMAP_TYPE_ICO_RESOURCE	Load a Windows icon from the resource database.
wxBITMAP_TYPE_GIF	Load a GIF bitmap file.
wxBITMAP_TYPE_XBM	Load an X bitmap file.
wxBITMAP_TYPE_XPM	Load an XPM bitmap file.

The validity of these flags depends on the platform and wxWindows configuration. If all possible wxWindows settings are used, the Windows platform supports ICO file, ICO resource, XPM data, and XPM file. Under wxGTK, the available formats are BMP file, XPM data, XPM file, and PNG file. Under wxMotif, the available formats are XBM data, XBM file, XPM data, XPM file.

Remarks

The first form constructs an icon object with no data; an assignment or another member function such as `Create` or `LoadFile` must be called subsequently.

The second and third forms provide copy constructors. Note that these do not copy the icon data, but instead a pointer to the data, keeping a reference count. They are therefore very efficient operations.

The fourth form constructs an icon from data whose type and value depends on the value of the *type* argument.

The fifth form constructs a (usually monochrome) icon from an array of pixel values, under both X and Windows.

The sixth form constructs a new icon.

The seventh form constructs an icon from pixmap (XPM) data, if wxWindows has been configured to incorporate this feature.

To use this constructor, you must first include an XPM file. For example, assuming that the file `mybitmap.xpm` contains an XPM array of character pointers called `mybitmap`:

```
#include "mybitmap.xpm"
```

```
...
```

```
wxIcon *icon = new wxIcon(mybitmap);
```

A macro, `wxICON`, is available which creates an icon using an XPM on the appropriate platform, or an icon resource on Windows.

```
wxIcon icon(wxICON(mondrian));
```

```
// Equivalent to:

#if defined(__WXGTK__) || defined(__WXMOTIF__)
wxIcon icon(mondrian_xpm);
#endif

#if defined(__WXMSW__)
wxIcon icon("mondrian");
#endif
```

The eighth form constructs an icon from a file or resource. *name* can refer to a resource name under MS Windows, or a filename under MS Windows and X.

Under Windows, *type* defaults to `wxBITMAP_TYPE_ICO_RESOURCE`. Under X, *type* defaults to `wxBITMAP_TYPE_XPM`.

See also

`wxIcon::LoadFile` (p. 323)

4.92.2 `wxIcon::~wxIcon`

`~wxIcon()`

Destroys the `wxIcon` object and possibly the underlying icon data. Because reference counting is used, the icon may not actually be destroyed at this point - only when the reference count is zero will the data be deleted.

If the application omits to delete the icon explicitly, the icon will be destroyed automatically by `wxWindows` when the application exits.

Do not delete an icon that is selected into a memory device context.

4.92.3 `wxIcon::GetDepth`

`int GetDepth() const`

Gets the colour depth of the icon. A value of 1 indicates a monochrome icon.

4.92.4 `wxIcon::GetHeight`

`int GetHeight() const`

Gets the height of the icon in pixels.

4.92.5 `wxIcon::GetWidth`

int GetWidth() const

Gets the width of the icon in pixels.

See also

wxIcon::GetHeight (p. 322)

4.92.6 wxIcon::LoadFile**bool LoadFile(const wxString& name, long type)**

Loads an icon from a file or resource.

Parameters

name

Either a filename or a Windows resource name. The meaning of *name* is determined by the *type* parameter.

type

One of the following values:

wxBITMAP_TYPE_ICO	Load a Windows icon file.
wxBITMAP_TYPE_ICO_RESOURCE	Load a Windows icon from the resource database.
wxBITMAP_TYPE_GIF	Load a GIF bitmap file.
wxBITMAP_TYPE_XBM	Load an X bitmap file.
wxBITMAP_TYPE_XPM	Load an XPM bitmap file.

The validity of these flags depends on the platform and wxWindows configuration.

Return value

TRUE if the operation succeeded, FALSE otherwise.

See also

wxIcon::wxIcon (p. 319)

4.92.7 wxIcon::Ok**bool Ok() const**

Returns TRUE if icon data is present.

4.92.8 wxIcon::SetDepth

void SetDepth(int *depth*)

Sets the depth member (does not affect the icon data).

Parameters

depth
Icon depth.

4.92.9 wxIcon::SetHeight

void SetHeight(int *height*)

Sets the height member (does not affect the icon data).

Parameters

height
Icon height in pixels.

4.92.10 wxIcon::SetOk

void SetOk(int *isOk*)

Sets the validity member (does not affect the icon data).

Parameters

isOk
Validity flag.

4.92.11 wxIcon::SetWidth

void SetWidth(int *width*)

Sets the width member (does not affect the icon data).

Parameters

width
Icon width in pixels.

4.92.12 wxIcon::operator =

wxIcon& operator =(const wxIcon& *icon*)

Assignment operator. This operator does not copy any data, but instead passes a pointer to the data in *icon* and increments a reference counter. It is a fast operation.

Parameters

icon
Icon to assign.

Return value

Returns 'this' object.

4.92.13 wxIcon::operator ==

bool operator ==(const wxIcon& *icon*)

Equality operator. This operator tests whether the internal data pointers are equal (a fast test).

Parameters

icon
Icon to compare with 'this'

Return value

Returns TRUE if the icons were effectively equal, FALSE otherwise.

4.92.14 wxIcon::operator !=

bool operator !=(const wxIcon& *icon*)

Inequality operator. This operator tests whether the internal data pointers are unequal (a fast test).

Parameters

icon
Icon to compare with 'this'

Return value

Returns TRUE if the icons were unequal, FALSE otherwise.

4.93 wxImage

This class encapsulates a platform-independent image. An image can be created from

data, or using the constructor taking a `wxBitmap` object. An image can be loaded from a file in a variety of formats, and is extensible to new formats via image format handlers. Functions are available to set and get image bits, so it can be used for basic image manipulation.

A `wxImage` cannot (currently) be drawn directly to a `wxDC`. Instead, a platform-specific `wxBitmap` object must be created from it, and that bitmap drawn on the `wxDC`, using `wxDC::DrawBitmap`.

Derived from

wxObject (p. 471)

Include files

`<wx/image.h>`

See also

wxBitmap (p. 39)

4.93.1 `wxImage::wxImage`

`wxImage()`

Default constructor.

`wxImage(const wxImage& image)`

Copy constructor.

`wxImage(const wxBitmap& bitmap)`

Constructs an image from a platform-dependent bitmap. This preserves mask information so that bitmaps and images can be converted back and forth without loss in that respect.

`wxImage(int width, int height)`

Creates an image with the given width and height.

`wxImage(const wxString& name, long type = wxBITMAP_TYPE_PNG)`

Loads an image from a file.

`wxImage(wxInputStream& stream, long type = wxBITMAP_TYPE_PNG)`

Loads an image from an input stream.

Parameters

width

Specifies the width of the image.

height

Specifies the height of the image.

name

This refers to an image filename. Its meaning is determined by the *type* parameter.

stream

This refers to an input stream. Its meaning is determined by the *type* parameter. It is equal to loading from file except that you provide opened stream (file, HTTP or any other custom class).

type

May be one of the following:

wxBITMAP_TYPE_BMP	Load a Windows bitmap file.
wxBITMAP_TYPE_PNG	Load a PNG bitmap file.
wxBITMAP_TYPE_JPEG	Load a JPEG bitmap file.

The validity of these flags depends on the platform and wxWindows configuration. If all possible wxWindows settings are used, the loading a BMP (Windows bitmap) file, a PNG (portable network graphics) file and a JPEG file is supported on all platforms that implement wxImage.

Note : you must call `wxImage::AddHandler(new wxJPEGHandler)` during application initialization in order to work with JPEGs.

See also

`wxImage::LoadFile` (p. 332)

4.93.2 wxImage::~wxImage

~wxImage()

Destructor.

4.93.3 wxImage::AddHandler

static void AddHandler(wxImageHandler* handler)

Adds a handler to the end of the static list of format handlers.

handler

A new image format handler object. There is usually only one instance of a given handler class in an application session.

See also

wxImageHandler (p. 335)

4.93.4 **wxImage::CleanUpHandlers**

static void CleanUpHandlers()

Deletes all image handlers.

This function is called by *wxWindows* on exit.

4.93.5 **wxImage::ConvertToBitmap**

wxBitmap ConvertToBitmap() const

Converts the image to a platform-specific bitmap object. This has to be done to actually display an image as you cannot draw an image directly on a window. The resulting bitmap will use the colour depth of the current system which entails that a (crude) colour reduction has to take place. When in 8-bit mode, this routine will use a color cube created on program start-up to look up colors. Still, the image quality won't be perfect for photo images.

4.93.6 **wxImage::Create**

bool Create(int width, int height)

Creates a fresh image.

Parameters

width

The width of the image in pixels.

height

The height of the image in pixels.

Return value

TRUE if the call succeeded, FALSE otherwise.

4.93.7 **wxImage::Destroy**

bool Destroy()

Destroys the image data.

4.93.8 wxImage::FindHandler**static wxImageHandler* FindHandler(const wxString& *name*)**

Finds the handler with the given name.

static wxImageHandler* FindHandler(const wxString& *extension*, long *imageType*)

Finds the handler associated with the given extension and type.

static wxImageHandler* FindHandler(long *imageType*)

Finds the handler associated with the given image type.

name

The handler name.

extension

The file extension, such as "bmp".

imageType

The image type, such as wxBITMAP_TYPE_BMP.

Return value

A pointer to the handler if found, NULL otherwise.

See also

wxImageHandler (p. 335)

4.93.9 wxImage::GetBlue**unsigned char GetBlue(int *x*, int *y*) const**

Returns the blue intensity at the given coordinate.

4.93.10 wxImage::GetData**unsigned char* GetData() const**

Returns the image data as an array. This is most often used when doing direct image manipulation. The return value points to an array of characters in RGBGBRGB... format.

4.93.11 wxImage::GetGreen**unsigned char GetGreen(int x, int y) const**

Returns the green intensity at the given coordinate.

4.93.12 wxImage::GetRed**unsigned char GetRed(int x, int y) const**

Returns the red intensity at the given coordinate.

4.93.13 wxImage::GetHandlers**static wxList& GetHandlers()**

Returns the static list of image format handlers.

[See also](#)

wxImageHandler (p. 335)

4.93.14 wxImage::GetHeight**int GetHeight() const**

Gets the height of the image in pixels.

4.93.15 wxImage::GetMaskBlue**unsigned char GetMaskBlue() const**

Gets the blue value of the mask colour.

4.93.16 wxImage::GetMaskGreen**unsigned char GetMaskGreen() const**

Gets the green value of the mask colour.

4.93.17 wxImage::GetMaskRed

unsigned char GetMaskRed() const

Gets the red value of the mask colour.

4.93.18 wxImage::GetWidth**int GetWidth() const**

Gets the width of the image in pixels.

[See also](#)

wxImage::GetHeight (p. 330)

4.93.19 wxImage::HasMask**bool HasMask() const**

Returns TRUE if there is a mask active, FALSE otherwise.

4.93.20 wxImage::InitStandardHandlers**static void InitStandardHandlers()**

Adds the standard image format handlers, which, depending on wxWindows configuration, can be handlers for Windows BMP (loading), PNG (loading and saving) and JPEG (loading and saving) file formats.

This function is called by wxWindows on startup.

[See also](#)

wxImageHandler (p. 335)

4.93.21 wxImage::InsertHandler**static void InsertHandler(wxImageHandler* handler)**

Adds a handler at the start of the static list of format handlers.

handler

A new image format handler object. There is usually only one instance of a given handler class in an application session.

[See also](#)

wxImageHandler (p. 335)

4.93.22 **wxImage::LoadFile**

bool LoadFile(const wxString& *name*, long *type*)

Loads an image from a file.

bool LoadFile(wxInputStream& *stream*, long *type*)

Loads an image from an input stream.

Parameters

name

A filename. The meaning of *name* is determined by the *type* parameter.

stream

An input stream. The meaning of *stream* data is determined by the *type* parameter.

type

One of the following values:

wxBITMAP_TYPE_BMP	Load a Windows image file.
wxBITMAP_TYPE_PNG	Load a PNG image file.
wxBITMAP_TYPE_JPEG	Load a JPEG image file.

The validity of these flags depends on the platform and wxWindows configuration.

Return value

TRUE if the operation succeeded, FALSE otherwise.

See also

wxImage::SaveFile (p. 333)

4.93.23 **wxImage::Ok**

bool Ok() const

Returns TRUE if image data is present.

4.93.24 **wxImage::RemoveHandler**

static bool RemoveHandler(const wxString& *name*)

Finds the handler with the given name, and removes it. The handler is not deleted.

name

The handler name.

Return value

TRUE if the handler was found and removed, FALSE otherwise.

See also

wxImageHandler (p. 335)

4.93.25 wxImage::SaveFile

bool SaveFile(const wxString& *name*, int *type*)

Saves a image in the named file.

bool SaveFile(wxOutputStream& *stream*, int *type*)

Saves a image in the given stream.

Parameters

name

A filename. The meaning of *name* is determined by the *type* parameter.

stream

An output stream. The meaning of *stream* is determined by the *type* parameter.

type

Currently two types can be used:

wxBITMAP_TYPE_PNG Save a PNG image file.

wxBITMAP_TYPE_JPEG Save a JPEG image file.

The validity of these flags depends on the platform and wxWindows configuration as well as user-added handlers.

Return value

TRUE if the operation succeeded, FALSE otherwise.

Remarks

Depending on how wxWindows has been configured, not all formats may be available.

See also

wxImage::LoadFile (p. 332)

4.93.26 wxImage::Scale

wxImage Scale(int *width*, int *height*)

Returns a scaled version of the image. This is also useful for scaling bitmaps in general as the only other way to scale bitmaps is to blit a wxMemoryDC into another wxMemoryDC. Windows can do such scaling itself but in the GTK port, scaling bitmaps is done using this routine internally.

4.93.27 wxImage::SetData

void SetData(unsigned char**data*)

Sets the image data without performing checks. The data given must have the size (width*height*3) or results will be unexpected. Don't use this method if you aren't sure you know what you are doing.

4.93.28 wxImage::SetMask

void SetMask(bool *hasMask* = *TRUE*)

Specifies whether there is a mask or not. The area of the mask is determined by the current mask colour.

4.93.29 wxImage::SetMaskColour

void SetMaskColour(unsigned char *red*, unsigned char *blue*, unsigned char *green*)

Sets the mask colour for this image (and tells the image to use the mask).

4.93.30 wxImage::SetRGB

void SetRGB(int *x*, int *y*, unsigned char *red*, unsigned char *blue*, unsigned char *green*)

Sets the pixel at the given coordinate. This routine performs bounds-checks for the coordinate so it can be considered a safe way to manipulate the data, but in some cases this might be too slow so that the data will have to be set directly. In that case you have to get that data by calling *GetData()*.

4.93.31 wxImage::operator =

wxImage& operator =(const wxImage& image)

Assignment operator. This operator does not copy any data, but instead passes a pointer to the data in *image* and increments a reference counter. It is a fast operation.

Parameters

image
Image to assign.

Return value

Returns 'this' object.

4.93.32 wxImage::operator ==

bool operator ==(const wxImage& image)

Equality operator. This operator tests whether the internal data pointers are equal (a fast test).

Parameters

image
Image to compare with 'this'

Return value

Returns TRUE if the images were effectively equal, FALSE otherwise.

4.93.33 wxImage::operator !=

bool operator !=(const wxImage& image)

Inequality operator. This operator tests whether the internal data pointers are unequal (a fast test).

Parameters

image
Image to compare with 'this'

Return value

Returns TRUE if the images were unequal, FALSE otherwise.

4.94 wxImageHandler

This is the base class for implementing image file loading/saving, and image creation from data. It is used within `wxImage` and is not normally seen by the application.

If you wish to extend the capabilities of `wxImage`, derive a class from `wxImageHandler` and add the handler using `wxImage::AddHandler` (p. 327) in your application initialisation.

Note (Legal Issue)

This software is based in part on the work of the Independent JPEG Group.

(Applies when `wxWindows` is linked with JPEG support. `wxJPEGHandler` uses libjpeg created by IJG.)

Derived from

`wxObject` (p. 471)

Include files

`<wx/image.h>`

See also

`wxImage` (p. 325)

4.94.1 `wxImageHandler::wxImageHandler`

`wxImageHandler()`

Default constructor. In your own default constructor, initialise the members `m_name`, `m_extension` and `m_type`.

4.94.2 `wxImageHandler::~~wxImageHandler`

`~wxImageHandler()`

Destroys the `wxImageHandler` object.

4.94.3 `wxImageHandler::GetName`

`wxString GetName() const`

Gets the name of this handler.

4.94.4 wxImageHandler::GetExtension

wxString GetExtension() const

Gets the file extension associated with this handler.

4.94.5 wxImageHandler::GetType

long GetType() const

Gets the image type associated with this handler.

4.94.6 wxImageHandler::LoadFile

bool LoadFile(wxImage* *image*, wxInputStream& *stream*)

Loads a image from a stream, putting the resulting data into *image*.

Parameters

image

The image object which is to be affected by this operation.

stream

Opened input stream. The meaning of *stream* is determined by the *type* parameter.

Return value

TRUE if the operation succeeded, FALSE otherwise.

See also

wxImage::LoadFile (p. 332)

wxImage::SaveFile (p. 333)

wxImageHandler::SaveFile (p. 337)

4.94.7 wxImageHandler::SaveFile

bool SaveFile(wxImage* *image*, wxOutputStream& *stream*)

Saves a image in the output stream.

Parameters

image

The image object which is to be affected by this operation.

stream

A stream. The meaning of *stream* is determined by the *type* parameter.

Return value

TRUE if the operation succeeded, FALSE otherwise.

See also

wxImage::LoadFile (p. 332)

wxImage::SaveFile (p. 333)

wxImageHandler::LoadFile (p. 337)

4.94.8 wxImageHandler::SetName

void SetName(const wxString& *name*)

Sets the handler name.

Parameters

name

Handler name.

4.94.9 wxImageHandler::SetExtension

void SetExtension(const wxString& *extension*)

Sets the handler extension.

Parameters

extension

Handler extension.

4.94.10 wxImageHandler::SetType

void SetType(long *type*)

Sets the handler type.

Parameters

name

Handler type.

4.95 wxImageList

A `wxImageList` contains a list of images, which are stored in an unspecified form. Images can have masks for transparent drawing, and can be made from a variety of sources including bitmaps and icons.

`wxImageList` is used principally in conjunction with `wxTreeCtrl` (p. 761) and `wxListCtrl` (p. 381) classes.

Derived from

`wxObject` (p. 471)

Include files

<wx/imaglist.h>

See also

`wxTreeCtrl` (p. 761), `wxListCtrl` (p. 381)

4.95.1 wxImageList::wxImageList

`wxImageList()`

Default constructor.

`wxImageList(int width, int height, const bool mask = TRUE, int initialCount = 1)`

Constructor specifying the image size, whether image masks should be created, and the initial size of the list.

Parameters

width
Width of the images in the list.

height
Height of the images in the list.

mask
TRUE if masks should be created for all images.

initialCount
The initial size of the list.

See also

wxImageList::Create (p. 340)

4.95.2 wxImageList::Add

int Add(const wxBitmap& *bitmap*, const wxBitmap& *mask* = wxNullBitmap)

Adds a new image using a bitmap and optional mask bitmap.

int Add(const wxBitmap& *bitmap*, const wxColour& *maskColour*)

Adds a new image using a bitmap and mask colour.

int Add(const wxIcon& *icon*)

Adds a new image using an icon.

Parameters

bitmap

Bitmap representing the opaque areas of the image.

mask

Monochrome mask bitmap, representing the transparent areas of the image.

maskColour

Colour indicating which parts of the image are transparent.

icon

Icon to use as the image.

Return value

The new zero-based image index.

Remarks

The original bitmap or icon is not affected by the **Add** operation, and can be deleted afterwards.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

```
Add(bitmap, mask=wxNullBitmap)
AddWithColourMask(bitmap, colour)
AddIcon(icon)
```

4.95.3 wxImageList::Create

bool Create(int width, int height, const bool mask = TRUE, int initialCount = 1)

Initializes the list. See *wxImageList::wxImageList* (p. 339) for details.

4.95.4 wxImageList::Draw

bool Draw(int index, wxDC& dc, int x, int y, int flags = wxIMAGELIST_DRAW_NORMAL, const bool solidBackground = FALSE)

Draws a specified image onto a device context.

Parameters

index

Image index, starting from zero.

dc

Device context to draw on.

x

X position on the device context.

y

Y position on the device context.

flags

How to draw the image. A bitlist of a selection of the following:

wxIMAGELIST_DRAW_NORMAL Draw the image normally.

wxIMAGELIST_DRAW_TRANSPARENT Draw the image with transparency.

wxIMAGELIST_DRAW_SELECTED Draw the image in selected state.

wxIMAGELIST_DRAW_FOCUSED Draw the image in a focussed state.

solidBackground

For optimisation - drawing can be faster if the function is told that the background is solid.

4.95.5 wxImageList::GetImageCount

int GetImageCount() const

Returns the number of images in the list.

4.95.6 wxImageList::Remove

bool Remove(int index)

Removes the image at the given position.

4.95.7 wxImageList::RemoveAll

bool RemoveAll()

Removes all the images in the list.

4.95.8 wxImageList::Replace

bool Replace(int index, const wxBitmap& bitmap, const wxBitmap& mask = wxNullBitmap)

Replaces the existing image with the new image.

bool Replace(int index, const wxIcon& icon)

Replaces the existing image with the new image.

Parameters

bitmap

Bitmap representing the opaque areas of the image.

mask

Monochrome mask bitmap, representing the transparent areas of the image.

icon

Icon to use as the image.

Return value

TRUE if the replacement was successful, FALSE otherwise.

Remarks

The original bitmap or icon is not affected by the **Replace** operation, and can be deleted afterwards.

wxPython note:

The second form is called `ReplaceIcon` in wxPython.

4.96 wxIndividualLayoutConstraint

Objects of this class are stored in the `wxIndividualLayoutConstraint` class as one of eight possible constraints that a window can be involved in.

Constraints are initially set to have the relationship `wxUnconstrained`, which means that their values should be calculated by looking at known constraints.

Derived from

`wxObject` (p. 471)

Include files

`<wx/layout.h>`

See also

Overview and examples (p. 919), *wxLayoutConstraints* (p. 365), *wxWindow::SetConstraints* (p. 835).

4.96.1 Edges and relationships

The `wxEdge` enumerated type specifies the type of edge or dimension of a window.

<code>wxLeft</code>	The left edge.
<code>wxTop</code>	The top edge.
<code>wxRight</code>	The right edge.
<code>wxBottom</code>	The bottom edge.
<code>wxCentreX</code>	The x-coordinate of the centre of the window.
<code>wxCentreY</code>	The y-coordinate of the centre of the window.

The `wxRelationship` enumerated type specifies the relationship that this edge or dimension has with another specified edge or dimension. Normally, the user doesn't use these directly because functions such as *Below* and *RightOf* are a convenience for using the more general *Set* function.

<code>wxUnconstrained</code>	The edge or dimension is unconstrained (the default for edges).
<code>wxAsIs</code>	The edge or dimension is to be taken from the current window position or size (the default for dimensions).
<code>wxAbove</code>	The edge should be above another edge.
<code>wxBelow</code>	The edge should be below another edge.
<code>wxLeftOf</code>	The edge should be to the left of another edge.
<code>wxRightOf</code>	The edge should be to the right of another edge.
<code>wxSameAs</code>	The edge or dimension should be the same as another edge or dimension.
<code>wxPercentOf</code>	The edge or dimension should be a percentage of another edge or dimension.
<code>wxAbsolute</code>	The edge or dimension should be a given absolute value.

4.96.2 wxIndividualLayoutConstraint::wxIndividualLayoutConstraint

void wxIndividualLayoutConstraint()

Constructor. Not used by the end-user.

4.96.3 wxIndividualLayoutConstraint::Above

void Above(wxWindow *otherWin, int margin = 0)

Constrains this edge to be above the given window, with an optional margin. Implicitly, this is relative to the top edge of the other window.

4.96.4 wxIndividualLayoutConstraint::Absolute

void Absolute(int value)

Constrains this edge or dimension to be the given absolute value.

4.96.5 wxIndividualLayoutConstraint::AsIs

void AsIs()

Sets this edge or constraint to be whatever the window's value is at the moment. If either of the width and height constraints are *as is*, the window will not be resized, but moved instead. This is important when considering panel items which are intended to have a default size, such as a button, which may take its size from the size of the button label.

4.96.6 wxIndividualLayoutConstraint::Below

void Below(wxWindow *otherWin, int margin = 0)

Constrains this edge to be below the given window, with an optional margin. Implicitly, this is relative to the bottom edge of the other window.

4.96.7 wxIndividualLayoutConstraint::Unconstrained

void Unconstrained()

Sets this edge or dimension to be unconstrained, that is, dependent on other edges and dimensions from which this value can be deduced.

4.96.8 wxIndividualLayoutConstraint::LeftOf

void LeftOf(wxWindow *otherWin, int margin = 0)

Constrains this edge to be to the left of the given window, with an optional margin. Implicitly, this is relative to the left edge of the other window.

4.96.9 wxIndividualLayoutConstraint::PercentOf

void PercentOf(wxWindow *otherWin, wxEdge edge, int margin = 0)

Constrains this edge or dimension to be to a percentage of the given window, with an optional margin.

4.96.10 wxIndividualLayoutConstraint::RightOf

void RightOf(wxWindow *otherWin, int margin = 0)

Constrains this edge to be to the right of the given window, with an optional margin. Implicitly, this is relative to the right edge of the other window.

4.96.11 wxIndividualLayoutConstraint::SameAs

void SameAs(wxWindow *otherWin, wxEdge edge, int margin = 0)

Constrains this edge or dimension to be to the same as the edge of the given window, with an optional margin.

4.96.12 wxIndividualLayoutConstraint::Set

void Set(wxRelationship rel, wxWindow *otherWin, wxEdge otherEdge, int value = 0, int margin = 0)

Sets the properties of the constraint. Normally called by one of the convenience functions such as Above, RightOf, SameAs.

4.97 wxInitDialogEvent

A wxInitDialogEvent is sent as a dialog or panel is being initialised. Handlers for this event can transfer data to the window.

Derived from

wxEvent (p. 221)

wxObject (p. 471)

Include files

<wx/event.h>

Event table macros

To process an activate event, use these event handler macros to direct input to a member function that takes a `wxInitDialogEvent` argument.

EVT_INIT_DIALOG(func) Process a `wxEVT_INIT_DIALOG` event.

See also

wxWindow::OnInitDialog (p. 823), *Event handling overview* (p. 939)

4.97.1 wxInitDialogEvent::wxInitDialogEvent

wxInitDialogEvent(int id = 0)

Constructor.

4.98 wxInputStream**Derived from**

wxStreamBase (p. 646)

Include files

<wx/stream.h>

See also

wxStreamBuffer (p. 648)

4.98.1 wxInputStream::wxInputStream

wxInputStream()

Creates a dummy input stream.

wxInputStream(wxStreamBuffer *sbuf)

Creates an input stream using the specified stream buffer *sbuf*. This stream buffer can point to another stream.

4.98.2 wxInputStream::~~wxInputStream

~wxInputStream()

Destructor.

4.98.3 wxInputStream::GetC

char GetC()

Returns the first character in the input queue and removes it.

4.98.4 wxInputStream::InputStreamBuffer

wxStreamBuffer* InputStreamBuffer()

Returns the stream buffer associated with the input stream.

4.98.5 wxInputStream::LastRead

size_t LastRead() const

Returns the last number of bytes read.

4.98.6 wxInputStream::Peek

char Peek()

Returns the first character in the input queue without removing it.

4.98.7 wxInputStream::Read

wxInputStream& Read(void *buffer, size_t size)

Reads the specified amount of bytes and stores the data in *buffer*.

Warning

The buffer absolutely needs to have at least the specified size.

Return value

This function returns a reference on the current object, so the user can test any states of

the stream right away.

wxInputStream& Read(wxOutputStream& *stream_out*)

Reads data from the input queue and stores it in the specified output stream. The data is read until an error is raised by one of the two streams.

Return value

This function returns a reference on the current object, so the user can test any states of the stream right away.

4.98.8 wxInputStream::SeekI

off_t SeekI(off_t *pos*, wxSeekMode *mode* = *wxFromStart*)

Changes the stream current position.

4.98.9 wxInputStream::TellI

off_t TellI() const

Returns the current stream position.

4.99 wxIPv4address

Derived from

wxSocketAddress (p. 605)

Include files

<wx/socket.h>

4.99.1 wxIPv4address::Hostname

bool Hostname(const wxString& *hostname*)

Use the specified *hostname* for the address.

Return value

Returns FALSE if something bad happens (invalid hostname, invalid IP address).

4.99.2 wxIPV4address::Hostname

wxString Hostname()

Returns the hostname which matches the IP address.

4.99.3 wxIPV4address::Service

bool Service(const **wxString**& *service*)

Use the specified *service* string for the address.

Return value

Returns FALSE if something bad happens (invalid service).

4.99.4 wxIPV4address::Service

bool Service(unsigned short *service*)

Use the specified *service* for the address.

Return value

Returns FALSE if something bad happens (invalid service).

4.99.5 wxIPV4address::Service

unsigned short Service()

Returns the current service.

4.99.6 wxIPV4address::LocalHost

bool LocalHost()

Initialize peer host to local host.

Return value

Returns FALSE if something bad happens.

4.100 wxJoystick

wxJoystick allows an application to control one or more joysticks.

Derived from

wxObject (p. 471)

Include files

<wx/joystick.h>

See also

wxJoystickEvent (p. 356)

4.100.1 wxJoystick::wxJoystick

wxJoystick(int joystick = wxJOYSTICK1)

Constructor. *joystick* may be one of wxJOYSTICK1, wxJOYSTICK2, indicating the joystick controller of interest.

4.100.2 wxJoystick::~~wxJoystick

~wxJoystick()

Destroys the wxJoystick object.

4.100.3 wxJoystick::GetButtonState

int GetButtonState() const

Returns the state of the joystick buttons. A bitlist of wxJOY_BUTTONn identifiers, where n is 1, 2, 3 or 4.

4.100.4 wxJoystick::GetManufacturerId

int GetManufacturerId() const

Returns the manufacturer id.

4.100.5 wxJoystick::GetMovementThreshold

int GetMovementThreshold() const

Returns the movement threshold, the number of steps outside which the joystick is

deemed to have moved.

4.100.6 wxJoystick::GetNumberAxes

int GetNumberAxes() const

Returns the number of axes for this joystick.

4.100.7 wxJoystick::GetNumberButtons

int GetNumberButtons() const

Returns the number of buttons for this joystick.

4.100.8 wxJoystick::GetNumberJoysticks

int GetNumberJoysticks() const

Returns the number of joysticks currently attached to the computer.

4.100.9 wxJoystick::GetPollingMax

int GetPollingMax() const

Returns the maximum polling frequency.

4.100.10 wxJoystick::GetPollingMin

int GetPollingMin() const

Returns the minimum polling frequency.

4.100.11 wxJoystick::GetProductId

int GetProductId() const

Returns the product id for the joystick.

4.100.12 wxJoystick::GetProductName

wxString GetProductName() const

Returns the product name for the joystick.

4.100.13 wxJoystick::GetPosition**wxPoint GetPosition() const**

Returns the x, y position of the joystick.

4.100.14 wxJoystick::GetPOVPosition**int GetPOVPosition() const**

Returns the point-of-view position, expressed in discrete units.

4.100.15 wxJoystick::GetPOVCTSPosition**int GetPOVCTSPosition() const**

Returns the point-of-view position, expressed in continuous, one-hundredth of a degree units.

4.100.16 wxJoystick::GetRudderMax**int GetRudderMax() const**

Returns the maximum rudder position.

4.100.17 wxJoystick::GetRudderMin**int GetRudderMin() const**

Returns the minimum rudder position.

4.100.18 wxJoystick::GetRudderPosition**int GetRudderPosition() const**

Returns the rudder position.

4.100.19 wxJoystick::GetUMax**int GetUMax() const**

Returns the maximum U position.

4.100.20 wxJoystick::GetUMin**int GetUMin() const**

Returns the minimum U position.

4.100.21 wxJoystick::GetUPosition**int GetUPosition() const**

Gets the position of the fifth axis of the joystick, if it exists.

4.100.22 wxJoystick::GetVMax**int GetVMax() const**

Returns the maximum V position.

4.100.23 wxJoystick::GetVMin**int GetVMin() const**

Returns the minimum V position.

4.100.24 wxJoystick::GetVPosition**int GetVPosition() const**

Gets the position of the sixth axis of the joystick, if it exists.

4.100.25 wxJoystick::GetXMax**int GetXMax() const**

Returns the maximum x position.

4.100.26 wxJoystick::GetXMin**int GetXMin() const**

Returns the minimum x position.

4.100.27 wxJoystick::GetYMax**int GetYMax() const**

Returns the maximum y position.

4.100.28 wxJoystick::GetYMin**int GetYMin() const**

Returns the minimum y position.

4.100.29 wxJoystick::GetZMax**int GetZMax() const**

Returns the maximum z position.

4.100.30 wxJoystick::GetZMin**int GetXMin() const**

Returns the minimum z position.

4.100.31 wxJoystick::GetZPosition**int GetZPosition() const**

Returns the z position of the joystick.

4.100.32 wxJoystick::HasPOV**bool HasPOV() const**

Returns TRUE if the joystick has a point of view control.

4.100.33 wxJoystick::HasPOV4Dir**bool HasPOV4Dir() const**

Returns TRUE if the joystick point-of-view supports discrete values (centered, forward, backward, left, and right).

4.100.34 wxJoystick::HasPOVCTS**bool HasPOVCTS() const**

Returns TRUE if the joystick point-of-view supports continuous degree bearings.

4.100.35 wxJoystick::HasRudder**bool HasRudder() const**

Returns TRUE if there is a rudder attached to the computer.

4.100.36 wxJoystick::HasU**bool HasU() const**

Returns TRUE if the joystick has a U axis.

4.100.37 wxJoystick::HasV**bool HasV() const**

Returns TRUE if the joystick has a V axis.

4.100.38 wxJoystick::HasZ**bool HasZ() const**

Returns TRUE if the joystick has a Z axis.

4.100.39 wxJoystick::IsOk**bool IsOk() const**

Returns TRUE if the joystick is functioning.

4.100.40 wxJoystick::ReleaseCapture**bool ReleaseCapture()**

Releases the capture set by **SetCapture**.

Return value

TRUE if the capture release succeeded.

See also

wxJoystick::SetCapture (p. 356), *wxJoystickEvent* (p. 356)

4.100.41 **wxJoystick::SetCapture**

bool SetCapture(wxWindow* win, int pollingFreq = 0)

Sets the capture to direct joystick events to *win*.

Parameters

win

The window that will receive joystick events.

pollingFreq

If zero, movement events are sent when above the threshold. If greater than zero, events are received every *pollingFreq* milliseconds.

Return value

TRUE if the capture succeeded.

See also

wxJoystick::ReleaseCapture (p. 355), *wxJoystickEvent* (p. 356)

4.100.42 **wxJoystick::SetMovementThreshold**

void SetMovementThreshold(int threshold)

Sets the movement threshold, the number of steps outside which the joystick is deemed to have moved.

4.101 **wxJoystickEvent**

This event class contains information about mouse events, particularly events received by windows.

Derived from

wxEvent (p. 221)

Include files

<wx/event.h>

Event table macros

To process a mouse event, use these event handler macros to direct input to member functions that take a `wxJoystickEvent` argument.

EVT_JOY_BUTTON_DOWN(func)	Process a <code>wxEVT_JOY_BUTTON_DOWN</code> event.
EVT_JOY_BUTTON_UP(func)	Process a <code>wxEVT_JOY_BUTTON_UP</code> event.
EVT_JOY_MOVE(func)	Process a <code>wxEVT_JOY_MOVE</code> event.
EVT_JOY_ZMOVE(func)	Process a <code>wxEVT_JOY_ZMOVE</code> event.

See also

wxJoystick (p. 349)

4.101.1 `wxJoystickEvent::wxJoystickEvent`

wxJoystickEvent(WXTYPE eventType = 0, int state = 0, int joystick = wxJOYSTICK1, int change = 0)

Constructor.

4.101.2 `wxJoystickEvent::ButtonDown`

bool ButtonDown(int button = wxJOY_BUTTON_ANY) const

Returns TRUE if the event was a down event from the specified button (or any button).

Parameters

button

Can be `wxJOY_BUTTONn` where *n* is 1, 2, 3 or 4; or `wxJOY_BUTTON_ANY` to indicate any button down event.

4.101.3 `wxJoystickEvent::ButtonIsDown`

bool ButtonIsDown(int button = wxJOY_BUTTON_ANY) const

Returns TRUE if the specified button (or any button) was in a down state.

Parameters

button

Can be wxJOY_BUTTONn where n is 1, 2, 3 or 4; or wxJOY_BUTTON_ANY to indicate any button down event.

4.101.4 wxJoystickEvent::ButtonUp

bool ButtonUp(int *button* = wxJOY_BUTTON_ANY) const

Returns TRUE if the event was an up event from the specified button (or any button).

Parameters

button

Can be wxJOY_BUTTONn where n is 1, 2, 3 or 4; or wxJOY_BUTTON_ANY to indicate any button down event.

4.101.5 wxJoystickEvent::GetButtonChange

int GetButtonChange() const

Returns the identifier of the button changing state. This is a wxJOY_BUTTONn identifier, where n is one of 1, 2, 3, 4.

4.101.6 wxJoystickEvent::GetButtonState

int GetButtonState() const

Returns the down state of the buttons. This is a bitlist of wxJOY_BUTTONn identifiers, where n is one of 1, 2, 3, 4.

4.101.7 wxJoystickEvent::GetJoystick

int GetJoystick() const

Returns the identifier of the joystick generating the event - one of wxJOYSTICK1 and wxJOYSTICK2.

4.101.8 wxJoystickEvent::GetPosition

wxPoint GetPosition() const

Returns the x, y position of the joystick event.

4.101.9 wxJoystickEvent::GetZPosition

int GetZPosition() const

Returns the z position of the joystick event.

4.101.10 wxJoystickEvent::IsButton**bool IsButton() const**

Returns TRUE if this was a button up or down event (*not* 'is any button down?').

4.101.11 wxJoystickEvent::IsMove**bool IsMove() const**

Returns TRUE if this was an x, y move event.

4.101.12 wxJoystickEvent::IsZMove**bool IsZMove() const**

Returns TRUE if this was a z move event.

4.102 wxKeyEvent

This event class contains information about keypress (character) events.

Derived from

wxEvent (p. 221)

Include files

<wx/event.h>

Event table macros

To process a key event, use these event handler macros to direct input to member functions that take a *wxKeyEvent* argument.

EVT_CHAR(func)	Process a <i>wxEVT_CHAR</i> event (a non-modifier key has been pressed).
EVT_KEY_DOWN(func)	Process a <i>wxEVT_KEY_DOWN</i> event (any key has been pressed).
EVT_KEY_UP(func)	Process a <i>wxEVT_KEY_UP</i> event (any key has been released).
EVT_CHAR(func)	Process a <i>wxEVT_CHAR</i> event.

EVT_CHAR_HOOK(func)

Process a wxEVT_CHAR_HOOK event.

See also

wxWindow::OnChar (p. 817), *wxWindow::OnCharHook* (p. 818),
wxWindow::OnKeyDown (p. 821), *wxWindow::OnKeyUp* (p. 822)

4.102.1 wxKeyEvent::m_altDown**bool m_altDown**

TRUE if the Alt key is pressed down.

4.102.2 wxKeyEvent::m_controlDown**bool m_controlDown**

TRUE if control is pressed down.

4.102.3 wxKeyEvent::m_keyCode**long m_keyCode**Virtual keycode. See *Keycodes* (p. 888) for a list of identifiers.**4.102.4 wxKeyEvent::m_metaDown****bool m_metaDown**

TRUE if the Meta key is pressed down.

4.102.5 wxKeyEvent::m_shiftDown**bool m_shiftDown**

TRUE if shift is pressed down.

4.102.6 wxKeyEvent::m_x**int m_x**

X position of the event.

4.102.7 wxKeyEvent::m_y

int m_y

Y position of the event.

4.102.8 wxKeyEvent::wxKeyEvent

wxKeyEvent(WXTYPE keyEventType)

Constructor. Currently, the only valid event types are wxEVT_CHAR and wxEVT_CHAR_HOOK.

4.102.9 wxKeyEvent::AltDown

bool AltDown()

Returns TRUE if the Alt key was down at the time of the key event.

4.102.10 wxKeyEvent::ControlDown

bool ControlDown()

Returns TRUE if the control key was down at the time of the key event.

4.102.11 wxKeyEvent::GetX

float GetX()

Returns the X position of the event.

4.102.12 wxKeyEvent::GetY

float GetY()

Returns the Y position of the event.

4.102.13 wxKeyEvent::KeyCode

long KeyCode()

Returns the virtual key code. ASCII events return normal ASCII values, while non-ASCII events return values such as **WXK_LEFT** for the left cursor key. See *Keycodes* (p. 888) for a full list of the virtual key codes.

4.102.14 wxKeyEvent::MetaDown

bool MetaDown()

Returns TRUE if the Meta key was down at the time of the key event.

4.102.15 wxKeyEvent::Position

void Position(float *x, float *y)

Obtains the position at which the key was pressed.

4.102.16 wxKeyEvent::ShiftDown

bool ShiftDown()

Returns TRUE if the shift key was down at the time of the key event.

4.103 wxLayoutAlgorithm

`wxLayoutAlgorithm` implements layout of subwindows in MDI or SDI frames. It sends a `wxCalculateLayoutEvent` event to children of the frame, asking them for information about their size. For MDI parent frames, the algorithm allocates the remaining space to the MDI client window (which contains the MDI child frames). For SDI (normal) frames, a 'main' window is specified as taking up the remaining space.

Because the event system is used, this technique can be applied to any windows, which are not necessarily 'aware' of the layout classes (no virtual functions in `wxWindow` refer to `wxLayoutAlgorithm` or its events). However, you may wish to use `wxSashLayoutWindow` (p. 570) for your subwindows since this class provides handlers for the required events, and accessors to specify the desired size of the window. The sash behaviour in the base class can be used, optionally, to make the windows user-resizable.

`wxLayoutAlgorithm` is typically used in IDE (integrated development environment) applications, where there are several resizable windows in addition to the MDI client window, or other primary editing window. Resizable windows might include toolbars, a project window, and a window for displaying error and warning messages.

When a window receives an `OnCalculateLayout` event, it should call `SetRect` in the given event object, to be the old supplied rectangle minus whatever space the window takes up. It should also set its own size accordingly.

`wxSashLayoutWindow::OnCalculateLayout` generates an `OnQueryLayoutInfo` event

which it sends to itself to determine the orientation, alignment and size of the window, which it gets from internal member variables set by the application.

The algorithm works by starting off with a rectangle equal to the whole frame client area. It iterates through the frame children, generating `OnCalculateLayout` events which subtract the window size and return the remaining rectangle for the next window to process. It is assumed (by `wxSashLayoutWindow::OnCalculateLayout`) that a window stretches the full dimension of the frame client, according to the orientation it specifies. For example, a horizontal window will stretch the full width of the remaining portion of the frame client area. In the other orientation, the window will be fixed to whatever size was specified by `OnQueryLayoutInfo`. An alignment setting will make the window 'stick' to the left, top, right or bottom of the remaining client area. This scheme implies that order of window creation is important. Say you wish to have an extra toolbar at the top of the frame, a project window to the left of the MDI client window, and an output window above the status bar. You should therefore create the windows in this order: toolbar, output window, project window. This ensures that the toolbar and output window take up space at the top and bottom, and then the remaining height inbetween is used for the project window.

`wxLayoutAlgorithm` is quite independent of the way in which `OnCalculateLayout` chooses to interpret a window's size and alignment. Therefore you could implement a different window class with a new `OnCalculateLayout` event handler, that has a more sophisticated way of laying out the windows. It might allow specification of whether stretching occurs in the specified orientation, for example, rather than always assuming stretching. (This could, and probably should, be added to the existing implementation).

Note: `wxLayoutAlgorithm` has nothing to do with `wxLayoutConstraints`. It is an alternative way of specifying layouts for which the normal constraint system is unsuitable.

Derived from

`wxObject` (p. 471)

Include files

`<wx/laywin.h>`

Event handling

The algorithm object does not respond to events, but itself generates the following events in order to calculate window sizes.

EVT_QUERY_LAYOUT_INFO(func)	Process a <code>wxEVT_QUERY_LAYOUT_INFO</code> event, to get size, orientation and alignment from a window. See <i>wxQueryLayoutInfoEvent</i> (p. 535).
EVT_CALCULATE_LAYOUT(func)	Process a <code>wxEVT_CALCULATE_LAYOUT</code> event, which asks the window to take a 'bite' out of a rectangle provided by the algorithm. See <i>wxCalculateLayoutEvent</i> (p. 68).

Data types

```
enum wxLayoutOrientation {  
    wxLAYOUT_HORIZONTAL,  
    wxLAYOUT_VERTICAL  
};  
  
enum wxLayoutAlignment {  
    wxLAYOUT_NONE,  
    wxLAYOUT_TOP,  
    wxLAYOUT_LEFT,  
    wxLAYOUT_RIGHT,  
    wxLAYOUT_BOTTOM,  
};
```

See also

wxSashEvent (p. 569), *wxSashLayoutWindow* (p. 570), *Event handling overview* (p. 939)

wxCalculateLayoutEvent (p. 68), *wxQueryLayoutInfoEvent* (p. 535),
wxSashLayoutWindow (p. 570), *wxSashWindow* (p. 573)

4.103.1 wxLayoutAlgorithm::wxLayoutAlgorithm

wxLayoutAlgorithm()

Default constructor.

4.103.2 wxLayoutAlgorithm::~~wxLayoutAlgorithm

~wxLayoutAlgorithm()

Destructor.

4.103.3 wxLayoutAlgorithm::LayoutFrame

bool LayoutFrame(wxFrame* frame, wxWindow* mainWindow = NULL) const

Lays out the children of a normal frame. *mainWindow* is set to occupy the remaining space.

This function simply calls *wxLayoutAlgorithm::LayoutWindow* (p. 365).

4.103.4 `wxLayoutAlgorithm::LayoutMDIFrame`

`bool LayoutMDIFrame(wxMDIParentFrame* frame, wxRect* rect = NULL) const`

Lays out the children of an MDI parent frame. If *rect* is non-NULL, the given rectangle will be used as a starting point instead of the frame's client area.

The MDI client window is set to occupy the remaining space.

4.103.5 `wxLayoutAlgorithm::LayoutWindow`

`bool LayoutWindow(wxWindow* parent, wxWindow* mainWindow = NULL) const`

Lays out the children of a normal frame or other window.

mainWindow is set to occupy the remaining space.

4.104 `wxLayoutConstraints`

Objects of this class can be associated with a window to define its layout constraints, with respect to siblings or its parent.

The class consists of the following eight constraints of class `wxIndividualLayoutConstraint`, some or all of which should be accessed directly to set the appropriate constraints.

- **left:** represents the left hand edge of the window
- **right:** represents the right hand edge of the window
- **top:** represents the top edge of the window
- **bottom:** represents the bottom edge of the window
- **width:** represents the width of the window
- **height:** represents the height of the window
- **centreX:** represents the horizontal centre point of the window
- **centreY:** represents the vertical centre point of the window

Most constraints are initially set to have the relationship `wxUnconstrained`, which means that their values should be calculated by looking at known constraints. The exceptions are *width* and *height*, which are set to `wxAsIs` to ensure that if the user does not specify a constraint, the existing width and height will be used, to be compatible with panel items which often have take a default size. If the constraint is `wxAsIs`, the dimension will not be changed.

Derived from

`wxObject` (p. 471)

Include files

`<wx/layout.h>`

See also

Overview and examples (p. 919), *wxIndividualLayoutConstraint* (p. 342), *wxWindow::SetConstraints* (p. 835)

4.104.1 wxLayoutConstraints::wxLayoutConstraints**wxLayoutConstraints()**

Constructor.

4.104.2 wxLayoutConstraints::bottom**wxIndividualLayoutConstraint bottom**

Constraint for the bottom edge.

4.104.3 wxLayoutConstraints::centreX**wxIndividualLayoutConstraint centreX**

Constraint for the horizontal centre point.

4.104.4 wxLayoutConstraints::centreY**wxIndividualLayoutConstraint centreY**

Constraint for the vertical centre point.

4.104.5 wxLayoutConstraints::height**wxIndividualLayoutConstraint height**

Constraint for the height.

4.104.6 wxLayoutConstraints::left**wxIndividualLayoutConstraint left**

Constraint for the left-hand edge.

4.104.7 wxLayoutConstraints::right**wxIndividualLayoutConstraint right**

Constraint for the right-hand edge.

4.104.8 wxLayoutConstraints::top**wxIndividualLayoutConstraint top**

Constraint for the top edge.

4.104.9 wxLayoutConstraints::width**wxIndividualLayoutConstraint width**

Constraint for the width.

4.105 wxList

`wxList` classes provide linked list functionality for `wxWindows`, and for an application if it wishes. Depending on the form of constructor used, a list can be keyed on integer or string keys to provide a primitive look-up ability. See *wxHashTable* (p. 310) for a faster method of storage when random access is required.

While `wxList` class in the previous versions of `wxWindows` only could contain elements of type `wxObject` and had essentially untyped interface (thus allowing you to put apples in the list and read back oranges from it), the new `wxList` classes family may contain elements of any type and has much more stricter type checking. Unfortunately, it also requires an additional line to be inserted in your program for each list class you use (which is the only solution short of using templates which is not done in `wxWindows` because of portability issues).

The general idea is to have the base class `wxListBase` working with `void *data` but make all of its dangerous (because untyped) functions protected, so that they can only be used from derived classes which, in turn, expose a type safe interface. With this approach a new `wxList`-like class must be defined for each list type (i.e. list of ints, of `wxStrings` or of `MyObjects`). This is done with `WX_DECLARE_LIST` and `WX_IMPLEMENT_LIST` macros like this (notice the similarity with `WX_DECLARE_OBJARRAY` and `WX_IMPLEMENT_OBJARRAY` macros):

Example

```
// this part might be in a header or source (.cpp) file
class MyListElement
{
    ... // whatever
```

```

};

// declare our list class: this macro declares and partly implements
MyList
// class (which derives from wxListBase)
WX_DECLARE_LIST(MyListElement, MyList)

...

// the only requirement for the rest is to be AFTER the full
declaration of
// MyListElement (for WX_DECLARE_LIST forward declaration is
enough), but
// usually it will be found in the source file and not in the header

#include <wx/listimpl.cpp>
WX_DEFINE_LIST(MyList)

// now MyList class may be used as a usual wxList, but all of its
methods
// will take/return the objects of the right (i.e. MyListElement)
type. You
// also have MyList::Node type which is the type-safe version of
wxNode.
MyList list;
MyListElement element;
list.Add(element);           // ok
list.Add(17);                // error: incorrect type

// let's iterate over the list
for ( MyList::Node *node = list.GetFirst(); node; node = node-
>GetNext() )
{
    MyListElement *current = node->GetData();

    ...process the current element...
}

```

For compatibility with previous versions `wxList` and `wxStringList` classes are still defined, but their usage is deprecated and they will disappear in the future versions completely.

Derived from

wxObject (p. 471)

Include files

<wx/list.h>

Example

It is very common to iterate on a list as follows:

```

...
wxWindow *win1 = new wxWindow(...);
wxWindow *win2 = new wxWindow(...);

```



```
wxList SomeList;
SomeList.Append(win1);
SomeList.Append(win2);

...

wxNode *node = SomeList.GetFirst();
while (node)
{
    wxWindow *win = (wxWindow *)node->Data();
    ...
    node = node->Next();
}
```

To delete nodes in a list as the list is being traversed, replace

```
...
node = node->Next();
...
```

with

```
...
delete win;
delete node;
node = SomeList.GetFirst();
...
```

See *wxNode* (p. 463) for members that retrieve the data associated with a node, and members for getting to the next or previous node.

Note that a cast is required when retrieving the data from a node. Although a node is defined to store objects of type **wxObject** and derived types, other types (such as `char*`) may be used with appropriate casting.

See also

wxNode (p. 463), *wxStringList* (p. 674), *wxArray* (p. 15)

4.105.1 wxList::wxList

wxList()

wxList(unsigned int *key_type*)

wxList(int *n*, wxObject **objects*[])

wxList(wxObject **object*, ...)

Constructors. *key_type* is one of `wxKEY_NONE`, `wxKEY_INTEGER`, or

`wxKEY_STRING`, and indicates what sort of keying is required (if any).

objects is an array of *n* objects with which to initialize the list.

The variable-length argument list constructor must be supplied with a terminating `NULL`.

4.105.2 `wxList::~~wxList`

`~wxList()`

Destroys the list. Also destroys any remaining nodes, but does not destroy client data held in the nodes.

4.105.3 `wxList::Append`

`wxNode * Append(wxObject *object)`

`wxNode * Append(long key, wxObject *object)`

`wxNode * Append(const wxString& key, wxObject *object)`

Appends a new **wxNode** to the end of the list and puts a pointer to the *object* in the node. The last two forms store a key with the object for later retrieval using the key. The new node is returned in each case.

The key string is copied and stored by the list implementation.

4.105.4 `wxList::Clear`

`void Clear()`

Clears the list (but does not delete the client data stored with each node).

4.105.5 `wxList::DeleteContents`

`void DeleteContents(bool destroy)`

If *destroy* is `TRUE`, instructs the list to call *delete* on the client contents of a node whenever the node is destroyed. The default is `FALSE`.

4.105.6 `wxList::DeleteNode`

`bool DeleteNode(wxNode *node)`

Deletes the given node from the list, returning `TRUE` if successful.

4.105.7 wxList::DeleteObject**bool DeleteObject(wxObject *object)**

Finds the given client *object* and deletes the appropriate node from the list, returning TRUE if successful. The application must delete the actual object separately.

4.105.8 wxList::Find**wxNode * Find(long key)****wxNode * Find(const wxString& key)**

Returns the node whose stored key matches *key*. Use on a keyed list only.

4.105.9 wxList::GetFirst**wxNode * GetFirst()**

Returns the first node in the list (NULL if the list is empty).

4.105.10 wxList::IndexOf**int IndexOf(wxObject* obj)**

Returns the index of *obj* within the list or NOT_FOUND if *obj* is not found in the list.

4.105.11 wxList::Insert**wxNode * Insert(wxObject *object)**

Insert object at front of list.

wxNode * Insert(wxNode *position, wxObject *object)

Insert object before *position*.

4.105.12 wxList::GetLast**wxNode * GetLast()**

Returns the last node in the list (NULL if the list is empty).

4.105.13 wxList::Member**wxNode * Member(wxObject *object)**

Returns the node associated with *object* if it is in the list, NULL otherwise.

4.105.14 wxList::Nth**wxNode * Nth(int n)**

Returns the *nth* node in the list, indexing from zero (NULL if the list is empty or the *nth* node could not be found).

4.105.15 wxList::Number**int Number()**

Returns the number of elements in the list.

4.105.16 wxList::Sort**void Sort(wxSortCompareFunction compfunc)**

```
// Type of compare function for list sort operation (as in 'qsort')
typedef int (*wxSortCompareFunction)(const void *elem1, const void
*elem2);
```

Allows the sorting of arbitrary lists by giving a function to compare two list elements. We use the system **qsort** function for the actual sorting process. The sort function receives pointers to wxObject pointers (wxObject **), so be careful to dereference appropriately.

Example:

```
int listcompare(const void *arg1, const void *arg2)
{
    return(compare(**(wxString **)arg1,    // use the wxString 'compare'
                  **(wxString **)arg2));  // function
}

void main()
{
    wxList list;

    list.Append(new wxString("DEF"));
    list.Append(new wxString("GHI"));
    list.Append(new wxString("ABC"));
    list.Sort(listcompare);
}
```

4.106

wxListBox

A listbox is used to select one or more of a list of strings. The strings are displayed in a scrolling box, with the selected string(s) marked in reverse video. A listbox can be single selection (if an item is selected, the previous selection is removed) or multiple selection (clicking an item toggles the item on or off independently of other selections).

List box elements are numbered from zero.

A listbox callback gets an event `wxEVT_COMMAND_LISTBOX_SELECT` for single clicks, and `wxEVT_COMMAND_LISTBOX_DOUBLE_CLICKED` for double clicks.

Derived from

wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/listbox.h>

Window styles

wxLB_SINGLE	Single-selection list.
wxLB_MULTIPLE	Multiple-selection list: the user can toggle multiple items on and off.
wxLB_EXTENDED	Extended-selection list: the user can select multiple items using the SHIFT key and the mouse or special key combinations.
wxLB_HSCROLL	Create horizontal scrollbar if contents are too wide (Windows only).
wxLB_ALWAYS_SB	Always show a vertical scrollbar.
wxLB_NEEDED_SB	Only create a vertical scrollbar if needed.
wxLB_SORT	The listbox contents are sorted in alphabetical order.

See also *window styles overview* (p. 959).

Event handling

EVT_LISTBOX(id, func)	Process a <code>wxEVT_COMMAND_LISTBOX_SELECTED</code> event, when an item on the list is selected.
EVT_LISTBOX_DCLICK(id, func)	Process a <code>wxEVT_COMMAND_LISTBOX_DOUBLE_CLICKED</code> event, when the listbox is doubleclicked.

See also

wxChoice (p. 75), *wxComboBox* (p. 94), *wxListCtrl* (p. 381), *wxCommandEvent* (p. 103)

4.106.1 **wxListBox::wxListBox**

wxListBox()

Default constructor.

wxListBox(*wxWindow** *parent*, *wxWindowID* *id*, *const wxPoint&* *pos* = *wxDefaultPosition*, *const wxSize&* *size* = *wxDefaultSize*, *int* *n*, *const wxString* *choices*[] = *NULL*, *long* *style* = 0, *const wxValidator&* *validator* = *wxDefaultValidator*, *const wxString&* *name* = "listBox")

Constructor, creating and showing a list box.

Parameters

parent

Parent window. Must not be NULL.

id

Window identifier. A value of -1 indicates a default value.

pos

Window position.

size

Window size. If the default size (-1, -1) is specified then the window is sized appropriately.

n

Number of strings with which to initialise the control.

choices

An array of strings with which to initialise the control.

style

Window style. See *wxListBox* (p. 373).

validator

Window validator.

name

Window name.

See also

wxListBox::Create (p. 375), *wxValidator* (p. 781)

wxPython note:

The *wxListBox* constructor in *wxPython* reduces the `nand` and `choices` arguments are to a single argument, which is a list of strings.

4.106.2 *wxListBox::~~wxListBox*

void *~wxListBox*()

Destructor, destroying the list box.

4.106.3 *wxListBox::Append*

void *Append*(const *wxString*& *item*)

Adds the item to the end of the list box.

void *Append*(const *wxString*& *item*, *char *clientData*)**

Adds the item to the end of the list box, associating the given data with the item.

Parameters

item

String to add.

clientData

Client data to associate with the item.

4.106.4 *wxListBox::Clear*

void *Clear*()

Clears all strings from the list box.

4.106.5 *wxListBox::Create*

bool *Create*(*wxWindow *parent*, *wxWindowID* *id*, const *wxPoint*& *pos* = *wxDefaultPosition*, const *wxSize*& *size* = *wxDefaultSize*, int *n*, const *wxString* *choices*[] = *NULL*, long *style* = 0, const *wxValidator*& *validator* = *wxDefaultValidator*, const *wxString*& *name* = "*listBox*")**

Creates the listbox for two-step construction. See *wxListBox::wxListBox* (p. 374) for further details.

4.106.6 wxListBox::Delete**void Delete(int *n*)**

Deletes an item from the listbox.

Parameters

n
The zero-based item index.

4.106.7 wxListBox::Deselect**void Deselect(int *n*)**

Deselects an item in the list box.

Parameters

n
The zero-based item to deselect.

Remarks

This applies to multiple selection listboxes only.

4.106.8 wxListBox::FindString**int FindString(const wxString& *string*)**

Finds an item matching the given string.

Parameters

string
String to find.

Return value

The zero-based position of the item, or -1 if the string was not found.

4.106.9 wxListBox::GetClientData**char* GetClientData(int *n*) const**

Returns a pointer to the client data associated with the given item (if any).

Parameters

n
The zero-based position of the item.

Return value

A pointer to the client data, or NULL if not present.

4.106.10 wxListBox::GetSelection

int GetSelection() const

Gets the position of the selected item.

Return value

The position of the current selection.

Remarks

Applicable to single selection list boxes only.

See also

wxListBox::SetSelection (p. 380), *wxListBox::GetStringSelection* (p. 378),
wxListBox::GetSelections (p. 377)

4.106.11 wxListBox::GetSelections

int GetSelections(int **selections) const

Gets an array containing the positions of the selected strings.

Parameters

selections
A pointer to an integer array, which will be allocated by the function if selects are present. Do not deallocate the returned array - it will be deallocated by the listbox.

Return value

The number of selections.

Remarks

Use this with a multiple selection listbox.

See also

wxListBox::GetSelection (p. 377), *wxListBox::GetStringSelection* (p. 378),
wxListBox::SetSelection (p. 380)

4.106.12 wxListBox::GetString

wxString GetString(int *n*) const

Returns the string at the given position.

Parameters

n

The zero-based position.

Return value

The string, or an empty string if the position was invalid.

4.106.13 wxListBox::GetStringSelection

wxString GetStringSelection() const

Gets the selected string - for single selection list boxes only. This must be copied by the calling program if long term use is to be made of it.

See also

wxListBox::GetSelection (p. 377), *wxListBox::GetSelections* (p. 377),
wxListBox::SetSelection (p. 380)

4.106.14 wxListBox::InsertItems

void InsertItems(int *nItems*, const wxString *items*, int *pos*)

Insert the given number of strings before the specified position.

Parameters

nItems

Number of items in the array *items*

items

Labels of items to be inserted

pos

Position before which to insert the items: for example, if *pos* is 0 the items will be inserted in the beginning of the listbox

4.106.15 **wxListBox::Number**

int Number() **const**

Returns the number of items in the listbox.

4.106.16 **wxListBox::Selected**

bool Selected(int *n*) **const**

Determines whether an item is selected.

Parameters

n
The zero-based item index.

Return value

TRUE if the given item is selected, FALSE otherwise.

4.106.17 **wxListBox::Set**

void Set(int *n*, const wxString* *choices*)

Clears the list box and adds the given strings.

Parameters

n
The number of strings to set.

choices
An array of strings to set.

Remarks

Deallocate the array from the calling program after this function has been called.

4.106.18 **wxListBox::SetClientData**

void SetClientData(int *n*, char* *data*)

Associates the given client data pointer with the given item.

Parameters

n
The zero-based item index.

data
The client data to associate with the item.

4.106.19 **wxListBox::SetFirstItem**

void SetFirstItem(int *n*)

void SetFirstItem(const wxString& *string*)

Set the specified item to be the first visible item.

Parameters

n
The zero-based item index.

string
The string that should be visible.

4.106.20 **wxListBox::SetSelection**

void SetSelection(int *n*, const bool *select* = TRUE)

Selects or deselects the given item.

Parameters

n
The zero-based item index.

select
If TRUE, will select the item. If FALSE, will deselect it.

4.106.21 **wxListBox::SetString**

void SetString(int *n*, const wxString& *string*)

Sets the string value of an item.

Parameters

n
The zero-based item index.

string
The string to set.

4.106.22 wxListBox::SetStringSelection

void SetStringSelection(const wxString& *string*, const bool *select* = TRUE)

Sets the current selection.

Parameters

string
The item to select.

select
If TRUE, will select the item. If FALSE, will deselect it.

4.107 wxListCtrl

A list control presents lists in a number of formats: list view, report view, icon view and small icon view. Elements are numbered from zero.

To intercept events from a list control, use the event table macros described in *wxListEvent* (p. 394).

Derived from

wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/listctrl.h>

Window styles

wxLC_LIST	multicolumn list view, with optional small icons. Columns are computed automatically, i.e. you don't set columns as in wxLC_REPORT. In other words, the list wraps, unlike a wxListBox.
wxLC_REPORT	single or multicolumn report view, with optional header.
wxLC_ICON	Large icon view, with optional labels.
wxLC_SMALL_ICON	Small icon view, with optional labels.
wxLC_ALIGN_TOP	Icons align to the top (default).

wxLC_ALIGN_LEFT	Icons align to the left.
wxLC_AUTOARRANGE	Icons arrange themselves.
wxLC_USER_TEXT	The application provides label text on demand, except for column headers.
wxLC_EDIT_LABELS	Labels are editable: the application will be notified when editing starts.
wxLC_NO_HEADER	No header in report mode.
wxLC_SINGLE_SEL	Single selection.
wxLC_SORT_ASCENDING	Sort in ascending order (must still supply a comparison callback in <code>SortItems</code>).
wxLC_SORT_DESCENDING	Sort in descending order (must still supply a comparison callback in <code>SortItems</code>).

See also *window styles overview* (p. 959).

Event handling

To process input from a list control, use these event handler macros to direct input to member functions that take a *wxCommandEvent* (p. 394) argument.

EVT_LIST_BEGIN_DRAG(id, func)	Begin dragging with the left mouse button.
EVT_LIST_BEGIN_RDRAG(id, func)	Begin dragging with the right mouse button.
EVT_LIST_BEGIN_LABEL_EDIT(id, func)	Begin editing a label.
EVT_LIST_END_LABEL_EDIT(id, func)	Finish editing a label.
EVT_LIST_DELETE_ITEM(id, func)	Delete an item.
EVT_LIST_DELETE_ALL_ITEMS(id, func)	Delete all items.
EVT_LIST_GET_INFO(id, func)	Request information from the application, usually the item text.
EVT_LIST_SET_INFO(id, func)	Information is being supplied (not implemented).
EVT_LIST_ITEM_SELECTED(id, func)	The item has been selected.
EVT_LIST_ITEM_DESELECTED(id, func)	The item has been deselected.
EVT_LIST_KEY_DOWN(id, func)	A key has been pressed.
EVT_LIST_INSERT_ITEM(id, func)	An item has been inserted.
EVT_LIST_COL_CLICK(id, func)	A column (m_col) has been left-clicked.

See also

wxListCtrl overview (p. 914), *wxListBox* (p. 373), *wxTreeCtrl* (p. 761), *wxImageList* (p. 339), *wxCommandEvent* (p. 394)

4.107.1 wxListCtrl::wxListCtrl

wxListCtrl()

Default constructor.

wxListCtrl(*wxWindow** parent, *wxWindowID* id, *const wxPoint&* pos = *wxDefaultPosition*, *const wxSize&* size = *wxDefaultSize*, *long* style = *wxLC_ICON*, *const wxValidator&* validator = *wxDefaultValidator*, *const wxString&* name = "listCtrl")

Constructor, creating and showing a list control.

Parameters

parent

Parent window. Must not be NULL.

id

Window identifier. A value of -1 indicates a default value.

pos

Window position.

size

Window size. If the default size (-1, -1) is specified then the window is sized appropriately.

style

Window style. See *wxListCtrl* (p. 381).

validator

Window validator.

name

Window name.

See also

wxListCtrl::Create (p. 384), *wxValidator* (p. 781)

4.107.2 **wxListCtrl::~~wxListCtrl**

void ~wxListCtrl()

Destructor, destroying the list control.

4.107.3 **wxListCtrl::Arrange**

bool Arrange(*int* flag = *wxLIST_ALIGN_DEFAULT*)

Arranges the items in icon or small icon view. *flag* is one of:

wxLIST_ALIGN_DEFAULT Default alignment.

wxLIST_ALIGN_LEFT Align to the left side of the control.

`wxLIST_ALIGN_TOP` Align to the top side of the control.
`wxLIST_ALIGN_SNAP_TO_GRID` Snap to grid.

4.107.4 **wxListCtrl::Create**

bool Create(*wxWindow* parent*, *wxWindowID id*, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = *wxLC_ICON*, **const wxValidator&** *validator* = *wxDefaultValidator*, **const wxString&** *name* = "listCtrl")

Creates the list control. See *wxListCtrl::wxListCtrl* (p. 382) for further details.

4.107.5 **wxListCtrl::DeleteItem**

bool DeleteItem(*long item*)

Deletes the specified item.

4.107.6 **wxListCtrl::DeleteAllItems**

bool DeleteAllItems()

Deletes all the items in the list control.

4.107.7 **wxListCtrl::DeleteColumn**

bool DeleteColumn(*int col*)

Deletes a column.

4.107.8 **wxListCtrl::Edit**

wxTextCtrl& Edit(*long item*)

Starts editing a label.

4.107.9 **wxListCtrl::EnsureVisible**

bool EnsureVisible(*long item*)

Ensures this item is visible.

4.107.10 **wxListCtrl::FindItem**

long FindItem(long start, const wxString& str, const bool partial = FALSE)

Find an item whose label matches this string, starting from the item after *start* or the beginning if *start* is -1.

long FindItem(long start, long data)

Find an item whose data matches this data, starting from the item after *start* or the beginning if *start* is -1.

long FindItem(long start, const wxPoint& pt, int direction)

Find an item nearest this position in the specified direction, starting from the item after *start* or the beginning if *start* is -1.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

```
FindItem(start, str, partial=FALSE)
FindItemData(start, data)
FindItemAtPos(start, point, direction)
```

4.107.11 wxListCtrl::GetColumn

bool GetColumn(int col, wxListItem& item) const

Gets information about this column. See *wxListCtrl::SetItem* (p. 391) for more information.

4.107.12 wxListCtrl::GetColumnWidth

int GetColumnWidth(int col) const

Gets the column width (report view only).

4.107.13 wxListCtrl::GetCountPerPage

int GetCountPerPage() const

Gets the number of items that can fit vertically in the visible area of the list control (list or report view) or the total number of items in the list control (icon or small icon view).

4.107.14 wxListCtrl::GetEditControl

wxTextCtrl& GetEditControl() const

Gets the edit control for editing labels.

4.107.15 wxListCtrl::GetImageList**wxImageList* GetImageList(int *which*) const**

Returns the specified image list. *which* may be one of:

wxIMAGE_LIST_NORMAL The normal (large icon) image list.

wxIMAGE_LIST_SMALL The small icon image list.

wxIMAGE_LIST_STATE The user-defined state image list (unimplemented).

4.107.16 wxListCtrl::GetItem**bool GetItem(wxListItem& *info*) const**

Gets information about the item. See *wxListCtrl::SetItem* (p. 391) for more information.

wxPython note:

The wxPython version of this method takes a parameter representing the item ID, and returns the wxListItem object.

4.107.17 wxListCtrl::GetItemData**long GetItemData(long *item*) const**

Gets the application-defined data associated with this item.

4.107.18 wxListCtrl::GetItemPosition**bool GetItemPosition(long *item*, wxPoint& *pos*) const**

Returns the position of the item, in icon or small icon view.

wxPython note:

The wxPython version of this method accepts only the item ID and returns the wxPoint.

4.107.19 wxListCtrl::GetItemRect**bool GetItemRect(long *item*, wxRect& *rect*, int *code* = wxLIST_RECT_BOUNDS)**

const

Returns the rectangle representing the item's size and position, in client coordinates.

code is one of `wxLIST_RECT_BOUNDS`, `wxLIST_RECT_ICON`, `wxLIST_RECT_LABEL`.

wxPython note:

The wxPython version of this method accepts only the item ID and returns the `wxRect`.

4.107.20 wxListCtrl::GetItemState

int GetItemState(long *item*, long *stateMask*) const

Gets the item state. For a list of state flags, see `wxListCtrl::SetItem` (p. 391).

The **stateMask** indicates which state flags are of interest.

4.107.21 wxListCtrl::GetItemCount

int GetItemCount() const

Returns the number of items in the list control.

4.107.22 wxListCtrl::GetItemSpacing

int GetItemSpacing(bool *isSmall*) const

Retrieves the spacing between icons in pixels. If *small* is TRUE, gets the spacing for the small icon view, otherwise the large icon view.

4.107.23 wxListCtrl::GetItemText

wxString GetItemText(long *item*) const

Gets the item text for this item.

4.107.24 wxListCtrl::GetNextItem

long GetNextItem(long *item*, int *geometry* = `wxLIST_NEXT_ALL`, int *state* = `wxLIST_STATE_DONTCARE`) const

Searches for an item with the given geometry or state, starting from *item*. *item* can be -1 to find the first item that matches the specified flags.

Returns the item or -1 if unsuccessful.

geometry can be one of:

<code>wxLIST_NEXT_ABOVE</code>	Searches for an item above the specified item.
<code>wxLIST_NEXT_ALL</code>	Searches for subsequent item by index.
<code>wxLIST_NEXT_BELOW</code>	Searches for an item below the specified item.
<code>wxLIST_NEXT_LEFT</code>	Searches for an item to the left of the specified item.
<code>wxLIST_NEXT_RIGHT</code>	Searches for an item to the right of the specified item.

state can be a bitlist of the following:

<code>wxLIST_STATE_DONTCARE</code>	Don't care what the state is.
<code>wxLIST_STATE_DROPHILITED</code>	The item indicates it is a drop target.
<code>wxLIST_STATE_FOCUSED</code>	The item has the focus.
<code>wxLIST_STATE_SELECTED</code>	The item is selected.
<code>wxLIST_STATE_CUT</code>	The item is selected as part of a cut and paste operation.

4.107.25 `wxListCtrl::GetSelectedItemCount`

`int GetSelectedItemCount() const`

Returns the number of selected items in the list control.

4.107.26 `wxListCtrl::GetTextColour`

`wxColour GetTextColour() const`

Gets the text colour of the list control.

4.107.27 `wxListCtrl::GetTopItem`

`long GetTopItem() const`

Gets the index of the topmost visible item when in list or report view.

4.107.28 `wxListCtrl::HitTest`

`long HitTest(const wxPoint& point, int& flags)`

Determines which item (if any) is at the specified point, giving details in *flags*. *flags* will be a combination of the following flags:

wxLIST_HITTEST_ABOVE Above the client area.
wxLIST_HITTEST_BELOW Below the client area.
wxLIST_HITTEST_NOWHERE In the client area but below the last item.
wxLIST_HITTEST_ONITEMICON On the bitmap associated with an item.
wxLIST_HITTEST_ONITEMLABEL On the label (string) associated with an item.
wxLIST_HITTEST_ONITEMRIGHT In the area to the right of an item.
wxLIST_HITTEST_ONITEMSTATEICON On the state icon for a tree view item that is in a user-defined state.
wxLIST_HITTEST_TOLEFT To the right of the client area.
wxLIST_HITTEST_TORIGHT To the left of the client area.
wxLIST_HITTEST_ONITEM Combination of **wxLIST_HITTEST_ONITEMICON**, **wxLIST_HITTEST_ONITEMLABEL**, and **wxLIST_HITTEST_ONITEMSTATEICON**.

4.107.29 **wxListCtrl::InsertColumn**

long InsertColumn(long col, wxListItem& info)

For list view mode (only), inserts a column. For more details, see *wxListCtrl::SetItem* (p. 391).

long InsertColumn(long col, const wxString& heading, int format = wxLIST_FORMAT_LEFT, int width = -1)

For list view mode (only), inserts a column. For more details, see *wxListCtrl::SetItem* (p. 391).

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

InsertColumn(col, heading, format=wxLIST_FORMAT_LEFT, width=-1)
 Creates a column using a header string only.
InsertColumnInfo(col, item) Creates a column using a wxListItem.

4.107.30 **wxListCtrl::InsertItem**

long InsertItem(wxListItem& info)

Inserts an item, returning the index of the new item if successful, -1 otherwise.

long InsertItem(long index, const wxString& label)

Inserts a string item.

long InsertItem(long index, int imageIndex)

Inserts an image item.

long InsertItem(long *index*, const wxString& *label*, int *imageIndex*)

Insert an image/string item.

Parameters

info

wxListItem object

index

Index of the new item, supplied by the application

label

String label

imageIndex

index into the image list associated with this control and view style

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

InsertItem(item)	Inserts an item using a wxListItem.
InsertStringItem(index, label)	Inserts a string item.
InsertImageItem(index, imageIndex)	Inserts an image item.
InsertImageStringItem(index, label, imageIndex)	Insert an image/string item.

4.107.31 wxListCtrl::ScrollList

bool ScrollList(int *dx*, int *dy*)

Scrolls the list control. If in icon, small icon or report view mode, *dx* specifies the number of pixels to scroll. If in list view mode, *dx* specifies the number of columns to scroll.

If in icon, small icon or list view mode, *dy* specifies the number of pixels to scroll. If in report view mode, *dy* specifies the number of lines to scroll.

4.107.32 wxListCtrl::SetBackgroundColour

void SetBackgroundColour(const wxColour& *col*)

Sets the background colour (GetBackgroundColour already implicit in wxWindow class).

4.107.33 wxListCtrl::SetColumn**bool SetColumn(int col, wxListItem& item)**

Sets information about this column. See *wxListCtrl::SetItem* (p. 391) for more information.

4.107.34 wxListCtrl::SetColumnWidth**bool SetColumnWidth(int col, int width)**

Sets the column width.

width can be a width in pixels or `wxLIST_AUTOSIZE` (-1) or `wxLIST_AUTOSIZE_USEHEADER` (-2).

In small or normal icon view, *col* must be -1, and the column width is set for all columns.

4.107.35 wxListCtrl::SetImageList**void SetImageList(wxImageList* imageList, int which)**

Sets the image list associated with the control. *which* is one of `wxIMAGE_LIST_NORMAL`, `wxIMAGE_LIST_SMALL`, `wxIMAGE_LIST_STATE` (the last is unimplemented).

4.107.36 wxListCtrl::SetItem**bool SetItem(wxListItem& info)**

Sets information about the item.

`wxListItem` is a class with the following members:

<code>long m_mask</code>	Indicates which fields are valid. See the list of valid mask flags below.
<code>long m_itemId</code>	The zero-based item position.
<code>int m_col</code>	Zero-based column, if in report mode.
<code>long m_state</code>	The state of the item. See the list of valid state flags below.
<code>long m_stateMask</code>	A mask indicating which state flags are valid. See the list of valid state flags below.
<code>wxString m_text</code>	The label/header text.
<code>int m_image</code>	The zero-based index into an image list.
<code>long m_data</code>	Application-defined data.
<code>int m_format</code>	For columns only: the format. Can be <code>wxLIST_FORMAT_LEFT</code> , <code>wxLIST_FORMAT_RIGHT</code> or <code>wxLIST_FORMAT_CENTRE</code> .

`int m_width` For columns only: the column width.

The **m_mask** member contains a bitlist specifying which of the other fields are valid. The flags are:

<code>wxLIST_MASK_STATE</code>	The m_state field is valid.
<code>wxLIST_MASK_TEXT</code>	The m_text field is valid.
<code>wxLIST_MASK_IMAGE</code>	The m_image field is valid.
<code>wxLIST_MASK_DATA</code>	The m_data field is valid.
<code>wxLIST_MASK_WIDTH</code>	The m_width field is valid.
<code>wxLIST_MASK_FORMAT</code>	The m_format field is valid.

The **m_stateMask** and **m_state** members take flags from the following:

<code>wxLIST_STATE_DONTCARE</code>	Don't care what the state is.
<code>wxLIST_STATE_DROPHILITED</code>	The item is highlighted to receive a drop event.
<code>wxLIST_STATE_FOCUSED</code>	The item has the focus.
<code>wxLIST_STATE_SELECTED</code>	The item is selected.
<code>wxLIST_STATE_CUT</code>	The item is in the cut state.

long SetItem(long index, int col, const wxString& label, int imageld = -1)

Sets a string field at a particular column.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

SetItem(item)	Sets information about the given <code>wxListItem</code> .
SetStringItem(index, col, label, imageld)	Sets a string or image at a given location.

4.107.37 wxListCtrl::SetItemData

bool SetItemData(long item, long data)

Associates application-defined data with this item.

4.107.38 wxListCtrl::SetItemImage

bool SetItemImage(long item, int image, int selImage)

Sets the unselected and selected images associated with the item. The images are indices into the image list associated with the list control.

4.107.39 wxListCtrl::SetItemPosition**bool SetItemPosition(long *item*, const wxPoint& *pos*)**

Sets the position of the item, in icon or small icon view.

4.107.40 wxListCtrl::SetItemState**bool SetItemState(long *item*, long *state*, long *stateMask*)**

Sets the item state. For a list of state flags, see *wxListCtrl::SetItem* (p. 391).

The **stateMask** indicates which state flags are valid.

4.107.41 wxListCtrl::SetItemText**void SetItemText(long *item*, const wxString& *text*)**

Sets the item text for this item.

4.107.42 wxListCtrl::SetSingleStyle**void SetSingleStyle(long *style*, const bool *add* = TRUE)**

Adds or removes a single window style.

4.107.43 wxListCtrl::SetTextColour**void SetTextColour(const wxColour& *col*)**

Sets the text colour of the list control.

4.107.44 wxListCtrl::SetWindowStyleFlag**void SetWindowStyleFlag(long *style*)**

Sets the whole window style.

4.107.45 wxListCtrl::SortItems**bool SortItems(wxListCtrlCompare *fn*, long *data*)**

Sorts the items in the list control.

fn is a function which takes 3 long arguments: item1, item2, data.

item1 is the long data associated with a first item (NOT the index).

item2 is the long data associated with a second item (NOT the index).

data is the same value as passed to SortItems.

The return value is a negative number if the first item should precede the second item, a positive number if the second item should precede the first, or zero if the two items are equivalent.

data is arbitrary data to be passed to the sort function.

4.108 wxListEvent

A list event holds information about events associated with wxListCtrl objects.

Derived from

wxCommandEvent (p. 103)

wxEvt (p. 221)

wxObject (p. 471)

Include files

<wx/listctrl.h>

Event table macros

To process input from a list control, use these event handler macros to direct input to member functions that take a wxListEvent argument.

EVT_LIST_BEGIN_DRAG(id, func)	Begin dragging with the left mouse button.
EVT_LIST_BEGIN_RDRAG(id, func)	Begin dragging with the right mouse button.
EVT_LIST_BEGIN_LABEL_EDIT(id, func)	Begin editing a label.
EVT_LIST_END_LABEL_EDIT(id, func)	Finish editing a label.
EVT_LIST_DELETE_ITEM(id, func)	Delete an item.
EVT_LIST_DELETE_ALL_ITEMS(id, func)	Delete all items.
EVT_LIST_GET_INFO(id, func)	Request information from the application, usually the item text.
EVT_LIST_SET_INFO(id, func)	Information is being supplied (not implemented).
EVT_LIST_ITEM_SELECTED(id, func)	The item has been selected.
EVT_LIST_ITEM_DESELECTED(id, func)	The item has been deselected.
EVT_LIST_KEY_DOWN(id, func)	A key has been pressed.
EVT_LIST_INSERT_ITEM(id, func)	An item has been inserted.
EVT_LIST_COL_CLICK(id, func)	A column (m_col) has been left-clicked.

See also

wxListCtrl (p. 381)

4.108.1 wxListEvent::wxListEvent

wxListEvent(WXTYPE *commandType* = 0, int *id* = 0)

Constructor.

4.108.2 wxListEvent::m_code

int m_code

Key code if the event is a keypress event.

4.108.3 wxListEvent::m_itemIndex

long m_itemIndex

The item index.

4.108.4 wxListEvent::m_oldItemIndex

long m_oldItemIndex

The old item index.

4.108.5 wxListEvent::m_col

int m_col

The column position.

4.108.6 wxListEvent::m_cancelled

bool m_cancelled

TRUE if this event is an end edit event and the user cancelled the edit.

4.108.7 wxListEvent::m_pointDrag

wxPoint m_pointDrag

The position of the mouse pointer if the event is a drag event.

4.108.8 wxListEvent::m_item**wxListItem m_item**

An item object, used by some events. See also *wxListCtrl::SetItem* (p. 391).

4.109 wxLocale

wxLocale class encapsulates all language-dependent settings and is a generalization of the C locale concept.

In wxWindows this class manages message catalogs which contain the translations of the strings used to the current language.

Derived from

No base class

See also

118n overview (p. 978)

Include files

<wx/intl.h>

4.109.1 wxLocale::wxLocale**wxLocale()**

This is the default constructor and it does nothing to initialize the object: *Init()* (p. 398) must be used to do that.

wxLocale(const char *szName, const char *szShort = NULL, const char *szLocale = NULL, bool bLoadDefault = TRUE)

The parameters have the following meaning:

- szName is the name of the locale and is only used in diagnostic messages
- szShort is the standard 2 letter locale abbreviation and is used as the directory prefix when looking for the message catalog files
- szLocale is the parameter for the call to setlocale()

- `bLoadDefault` may be set to `FALSE` to prevent loading of the message catalog for the given locale containing the translations of standard wxWindows messages. This parameter would be rarely used in normal circumstances.

The call of this function has several global side effects which you should understand: first of all, the application locale is changed - note that this will affect many of standard C library functions such as `printf()` or `strftime()`. Second, this `wxLocale` object becomes the new current global locale for the application and so all subsequent calls to `wxGetTranslation()` will try to translate the messages using the message catalogs for this locale.

4.109.2 `wxLocale::~~wxLocale`

`~wxLocale()`

The destructor, like the constructor, also has global side effects: the previously set locale is restored and so the changes described in *Init* (p. 398) documentation are rolled back.

4.109.3 `wxLocale::GetLocale`

`const char* GetLocale() const`

Returns the locale name as passed to the constructor or *Init*() (p. 398).

4.109.4 `wxLocale::AddCatalog`

`bool AddCatalog(const char *szDomain)`

Add a catalog for use with the current locale: it's searched for in standard places (current directory first, then the system one), but you may also prepend additional directories to the search path with *AddCatalogLookupPathPrefix()* (p. 397).

All loaded catalogs will be used for message lookup by `GetString()` for the current locale.

Returns `TRUE` if catalog was successfully loaded, `FALSE` otherwise (which might mean that the catalog is not found or that it isn't in the correct format).

4.109.5 `wxLocale::AddCatalogLookupPathPrefix`

`void AddCatalogLookupPathPrefix(const wxString& prefix)`

Add a prefix to the catalog lookup path: the message catalog files will be looked up under `prefix/<lang>/LC_MESSAGES`, `prefix/LC_MESSAGES` and `prefix` (in this order).

This only applies to subsequent invocations of `AddCatalog()`!

4.109.6 wxLocale::Init

bool Init(const char *szName, const char *szShort = NULL, const char *szLocale = NULL, bool bLoadDefault = TRUE)

The parameters have the following meaning:

- szName is the name of the locale and is only used in diagnostic messages
- szShort is the standard 2 letter locale abbreviation and is used as the directory prefix when looking for the message catalog files
- szLocale is the parameter for the call to setlocale()
- bLoadDefault may be set to FALSE to prevent loading of the message catalog for the given locale containing the translations of standard wxWindows messages. This parameter would be rarely used in normal circumstances.

The call of this function has several global side effects which you should understand: first of all, the application locale is changed - note that this will affect many of standard C library functions such as printf() or strftime(). Second, this wxLocale object becomes the new current global locale for the application and so all subsequent calls to wxGetTranslation() will try to translate the messages using the message catalogs for this locale.

Returns TRUE on success or FALSE if the given locale couldn't be set.

4.109.7 wxLocale::IsLoaded

bool IsLoaded(const char* domain) const

Check if the given catalog is loaded, and returns TRUE if it is.

According to GNU gettext tradition, each catalog normally corresponds to 'domain' which is more or less the application name.

See also: *AddCatalog* (p. 397)

4.109.8 wxLocale::GetName

const wxString& GetName() const

Returns the current short name for the locale (as given to the constructor or the Init() function).

4.109.9 wxLocale::GetString

**const char* GetString(const char *szOrigString, const char *szDomain = NULL)
const**

Retrieves the translation for a string in all loaded domains unless the `szDomain` parameter is specified (and then only this catalog/domain is searched).

Returns original string if translation is not available (in this case an error message is generated the first time a string is not found; use *wxLogNull* (p. 904) to suppress it).

Remarks

Domains are searched in the last to first order, i.e. catalogs added later override those added before.

4.110 wxLog

`wxLog` class defines the interface for the *log targets* used by `wxWindows` logging functions as explained in the *wxLog overview* (p. 904). The only situations when you need to directly use this class is when you want to derive your own log target because the existing ones don't satisfy your needs. Another case is if you wish to customize the behaviour of the standard logging classes (all of which respect the `wxLog` settings): for example, set which trace messages are logged and which are not or change (or even remove completely) the timestamp on the messages.

Otherwise, it is completely hidden behind the *wxLogXXX()* functions and you may not even know about its existence.

See *log overview* (p. 904) for the descriptions of `wxWindows` logging facilities.

Derived from

No base class

Include files

<wx/log.h>

4.110.1 Static functions

The functions in this section work with and manipulate the active log target. The *OnLog()* is called by the *wxLogXXX()* functions and invokes the *DoLog()* of the active log target if any. *Get/Set* methods are used to install/query the current active target and, finally, *DontCreateOnDemand()* disables the automatic creation of a standard log target if none actually exists. It is only useful when the application is terminating and shouldn't be used in other situations because it may easily lead to a loss of messages.

OnLog (p. 401)

GetActiveTarget (p. 401)

SetActiveTarget (p. 401)

DontCreateOnDemand (p. 401)

4.110.2 Message buffering

Some of wxLog implementations, most notably the standard wxLogGui class, buffer the messages (for example, to avoid showing the user a zillion of modal message boxes one after another - which would be really annoying). *Flush()* shows them all and clears the buffer contents. Although this function doesn't do anything if the buffer is already empty, *HasPendingMessages()* is also provided which allows to explicitly verify it.

Flush (p. 401)

HasPendingMessages (p. 402)

4.110.3 Customization

The functions below allow some limited customization of wxLog behaviour without writing a new log target class (which, aside of being a matter of several minutes, allows you to do anything you want).

The verbose messages are the trace messages which are not disabled in the release mode and are generated by *wxLogVerbose()*. They are not normally shown to the user because they present little interest, but may be activated, for example, in order to help the user find some program problem.

As for the (real) trace messages, they come in different kinds:

- for the messages about creating and deleting objects
- for tracing the windowing system messages/events
- for allocating and releasing the system resources
- for reference counting related messages
- for the OLE (or COM) method invocations (wxMSW only)
- the remaining bits are free for user-defined trace levels

The trace mask is a bit mask which tells which (if any) of these trace messages are going to be actually logged. For the trace message to appear somewhere, all the bits in the mask used in the call to *wxLogTrace()* function must be set in the current trace mask. For example,

```
wxLogTrace(wxTraceRefCount | wxTraceOle, "Active object ref count: %d",  
nRef);
```

will do something only if the current trace mask contains both wxTraceRefCount and wxTraceOle.

Finally, the *wxLog::DoLog()* function automatically prepends a time stamp to all the messages. The format of the time stamp may be changed: it can be any string with % specifiers fully described in the documentation of the standard *strftime()* function. For example, the default format is "[%d/%b/%y %H:%M:%S] " which gives something like "[17/Sep/98 22:10:16] " (without quotes) for the current date. Setting an empty string as the time format disables timestamping of the messages completely.

SetVerbose (p. 402)
GetVerbose (p. 402)
SetTimeStampFormat (p. 402)
GetTimeStampFormat (p. 402)
SetTraceMask (p. 402)
GetTraceMask (p. 402)

4.110.4 wxLog::OnLog

static void OnLog(wxLogLevel *level*, const char * *message*)

Forwards the message at specified level to the *DoLog()* function of the active log target if there is any, does nothing otherwise.

4.110.5 wxLog::GetActiveTarget

static wxLog * GetActiveTarget()

Returns the pointer to the active log target (may be NULL).

4.110.6 wxLog::SetActiveTarget

static wxLog * SetActiveTarget(wxLog * *logtarget*)

Sets the specified log target as the active one. Returns the pointer to the previous active log target (may be NULL).

4.110.7 wxLog::DontCreateOnDemand

static void DontCreateOnDemand()

Instructs wxLog to not create new log targets on the fly if there is none currently. (Almost) for internal use only.

4.110.8 wxLog::Flush

virtual void Flush()

Shows all the messages currently in buffer and clears it. If the buffer is already empty, nothing happens.

4.110.9 wxLog::HasPendingMessages**bool HasPendingMessages() const**

Returns true if there are any messages in the buffer (not yet shown to the user). (Almost) for internal use only.

4.110.10 wxLog::SetVerbose**void SetVerbose(bool verbose = TRUE)**

Activates or deactivates verbose mode in which the verbose messages are logged as the normal ones instead of being silently dropped.

4.110.11 wxLog::GetVerbose**bool GetVerbose() const**

Returns whether the verbose mode is currently active.

4.110.12 wxLog::SetTimeStampFormat**void SetTimeStampFormat(const char * format)**

Sets the timestamp format prepended by the default log targets to all messages. The string may contain any normal characters as well as %prefixed format specifiers, see *strftime()* manual for details. Passing an empty string to this function disables message timestamping.

4.110.13 wxLog::GetTimeStampFormat**const char * GetTimeStampFormat() const**

Returns the current timestamp format string.

4.110.14 wxLog::SetTraceMask**static void SetTraceMask(wxTraceMask mask)**

Sets the trace mask, see *Customization* (p. 400) section for details.

4.110.15 wxLog::GetTraceMask

Returns the current trace mask, see *Customization* (p. 400) section for details.

4.111 wxMask

This class encapsulates a monochrome mask bitmap, where the masked area is black and the unmasked area is white.

Derived from

wxObject (p. 471)

Include files

<wx/bitmap.h>

Remarks

A mask may be associated with a *wxBitmap* (p. 39). It is used in *wxDC::Blit* (p. 152) when the source device context is a *wxMemoryDC* (p. 415) with *wxBitmap* selected into it that contains a mask.

See also

wxBitmap (p. 39), *wxDC::Blit* (p. 152), *wxMemoryDC* (p. 415)

4.111.1 wxMask::wxMask

wxMask()

Default constructor.

wxMask(const wxBitmap& *bitmap*)

Constructs a mask from a monochrome bitmap.

wxMask(const wxBitmap& *bitmap*, const wxColour& *colour*)

Constructs a mask from a bitmap and a colour that indicates the background.

wxMask(const wxBitmap& *bitmap*, int *index*)

Constructs a mask from a bitmap and a palette index that indicates the background.

Parameters

bitmap

A valid bitmap.

colour

A colour specifying the transparency RGB values.

index

Index into a palette, specifying the transparency colour.

4.111.2 **wxMask::~~wxMask**

~wxMask()

Destroys the wxMask object and the underlying bitmap data.

4.111.3 **wxMask::Create**

bool Create(const wxBitmap& *bitmap*)

Constructs a mask from a monochrome bitmap.

bool Create(const wxBitmap& *bitmap*, const wxColour& *colour*)

Constructs a mask from a bitmap and a colour that indicates the background.

bool Create(const wxBitmap& *bitmap*, int *index*)

Constructs a mask from a bitmap and a palette index that indicates the background.

Parameters

bitmap

A valid bitmap.

colour

A colour specifying the transparency RGB values.

index

Index into a palette, specifying the transparency colour.

4.112 **wxMDIChildFrame**

An MDI child frame is a frame that can only exist on a *wxMDIClientWindow* (p. 407), which is itself a child of *wxMDIParentFrame* (p. 409).

Derived from

wxFrame (p. 275)

wxWindow (p. 798)

wxEvtHandler (p. 224)

wxObject (p. 471)

Include files

<wx/mdi.h>

Window styles

wxCAPTION	Puts a caption on the frame.
wxDEFAULT_FRAME_STYLE	Defined as wxMINIMIZE_BOX wxMAXIMIZE_BOX wxTHICK_FRAME wxSYSTEM_MENU wxCAPTION .
wxICONIZE	Display the frame iconized (minimized) (Windows only).
wxMAXIMIZE	Displays the frame maximized (Windows only).
wxMAXIMIZE_BOX	Displays a maximize box on the frame (Windows and Motif only).
wxMINIMIZE	Identical to wxICONIZE .
wxMINIMIZE_BOX	Displays a minimize box on the frame (Windows and Motif only).
wxRESIZE_BORDER	Displays a resizable border around the window (Motif only; for Windows, it is implicit in wxTHICK_FRAME).
wxSTAY_ON_TOP	Stay on top of other windows (Windows only).
wxSYSTEM_MENU	Displays a system menu (Windows and Motif only).
wxTHICK_FRAME	Displays a thick frame around the window (Windows and Motif only).

See also *window styles overview* (p. 959).

Remarks

Although internally an MDI child frame is a child of the MDI client window, in `wxWindows` you create it as a child of `wxMDIParentFrame` (p. 409). You can usually forget that the client window exists.

MDI child frames are clipped to the area of the MDI client window, and may be iconized on the client window.

You can associate a menubar with a child frame as usual, although an MDI child doesn't display its menubar under its own title bar. The MDI parent frame's menubar will be changed to reflect the currently active child frame. If there are currently no children, the parent frame's own menubar will be displayed.

See also

`wxMDIClientWindow` (p. 407), `wxMDIParentFrame` (p. 409), `wxFrame` (p. 275)

4.112.1 wxMDIChildFrame::wxMDIChildFrame

wxMDIChildFrame()

Default constructor.

wxMDIChildFrame(wxMDIParentFrame* *parent*, wxWindowID *id*, const wxString& *title*, const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE, const wxString& *name* = "frame")

Constructor, creating the window.

Parameters

parent

The window parent. This should not be NULL.

id

The window identifier. It may take a value of -1 to indicate a default value.

title

The caption to be displayed on the frame's title bar.

pos

The window position. A value of (-1, -1) indicates a default position, chosen by either the windowing system or wxWindows, depending on platform.

size

The window size. A value of (-1, -1) indicates a default size, chosen by either the windowing system or wxWindows, depending on platform.

style

The window style. See *wxMDIChildFrame* (p. 404).

name

The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

Remarks

None.

See also

wxMDIChildFrame::Create (p. 407)

4.112.2 wxMDIChildFrame::~~wxMDIChildFrame

~wxMDIChildFrame()

Destructor. Destroys all child windows and menu bar if present.

4.112.3 **wxMDIChildFrame::Activate**

void Activate()

Activates this MDI child frame.

[See also](#)

wxMDIChildFrame::Maximize (p. 407), *wxMDIChildFrame::Restore* (p. 407)

4.112.4 **wxMDIChildFrame::Create**

bool Create(wxWindow* parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT_FRAME_STYLE, const wxString& name = "frame")

Used in two-step frame construction. See *wxMDIChildFrame::wxMDIChildFrame* (p. 405) for further details.

4.112.5 **wxMDIChildFrame::Maximize**

void Maximize()

Maximizes this MDI child frame.

[See also](#)

wxMDIChildFrame::Activate (p. 407), *wxMDIChildFrame::Restore* (p. 407)

4.112.6 **wxMDIChildFrame::Restore**

void Restore()

Restores this MDI child frame (unmaximizes).

[See also](#)

wxMDIChildFrame::Activate (p. 407), *wxMDIChildFrame::Maximize* (p. 407)

4.113 **wxMDIClientWindow**

An MDI client window is a child of *wxMDIParentFrame* (p. 409), and manages zero or more *wxMDIChildFrame* (p. 404) objects.

[Derived from](#)

wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/mdi.h>

Remarks

The client window is the area where MDI child windows exist. It doesn't have to cover the whole parent frame; other windows such as toolbars and a help window might coexist with it. There can be scrollbars on a client window, which are controlled by the parent window style.

The **wxMDIClientWindow** class is usually adequate without further derivation, and it is created automatically when the MDI parent frame is created. If the application needs to derive a new class, the function *wxMDIParentFrame::OnCreateClient* (p. 414) must be overridden in order to give an opportunity to use a different class of client window.

Under Windows 95, the client window will automatically have a sunken border style when the active child is not maximized, and no border style when a child is maximized.

See also

wxMDIChildFrame (p. 404), *wxMDIParentFrame* (p. 409), *wxFrame* (p. 275)

4.113.1 **wxMDIClientWindow::wxMDIClientWindow**

wxMDIClientWindow()

Default constructor.

wxMDIClientWindow(wxMDIParentFrame* parent, long style = 0)

Constructor, creating the window.

Parameters

parent
The window parent.

style
The window style. Currently unused.

Remarks

The second style of constructor is called within *wxMDIParentFrame::OnCreateClient* (p.

414).

See also

wxMDIParentFrame::wxMDIParentFrame (p. 410), *wxMDIParentFrame::OnCreateClient* (p. 414)

4.113.2 **wxMDIClientWindow::~~wxMDIClientWindow**

~wxMDIClientWindow()

Destructor.

4.113.3 **wxMDIClientWindow::CreateClient**

bool CreateClient(wxMDIParentFrame* parent, long style = 0)

Used in two-step frame construction. See *wxMDIClientWindow::wxMDIClientWindow* (p. 408) for further details.

4.114 **wxMDIParentFrame**

An MDI (Multiple Document Interface) parent frame is a window which can contain MDI child frames in its own 'desktop'. It is a convenient way to avoid window clutter, and is used in many popular Windows applications, such as Microsoft Word(TM).

Derived from

wxFrame (p. 275)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/mdi.h>

Remarks

There may be multiple MDI parent frames in a single application, but this probably only makes sense within programming development environments.

Child frames may be either *wxMDIChildFrame* (p. 404), or *wxFrame* (p. 275).

An MDI parent frame always has a *wxMDIClientWindow* (p. 407) associated with it, which is the parent for MDI client frames. This client window may be resized to accomodate non-MDI windows, as seen in Microsoft Visual C++ (TM) and Microsoft Publisher (TM), where a documentation window is placed to one side of the workspace.

MDI remains popular despite dire warnings from Microsoft itself that MDI is an obsolete user interface style.

The implementation is native in Windows, and simulated under Motif. Under Motif, the child window frames will often have a different appearance from other frames because the window decorations are simulated.

Window styles

wxCAPTION	Puts a caption on the frame.
wxDEFAULT_FRAME_STYLE	Defined as wxMINIMIZE_BOX wxMAXIMIZE_BOX wxTHICK_FRAME wxSYSTEM_MENU wxCAPTION .
wxHSCROLL	Displays a horizontal scrollbar in the <i>client window</i> , allowing the user to view child frames that are off the current view.
wxICONIZE	Display the frame iconized (minimized) (Windows only).
wxMAXIMIZE	Displays the frame maximized (Windows only).
wxMAXIMIZE_BOX	Displays a maximize box on the frame (Windows and Motif only).
wxMINIMIZE	Identical to wxICONIZE .
wxMINIMIZE_BOX	Displays a minimize box on the frame (Windows and Motif only).
wxRESIZE_BORDER	Displays a resizable border around the window (Motif only; for Windows, it is implicit in wxTHICK_FRAME).
wxSTAY_ON_TOP	Stay on top of other windows (Windows only).
wxSYSTEM_MENU	Displays a system menu (Windows and Motif only).
wxTHICK_FRAME	Displays a thick frame around the window (Windows and Motif only).
wxVSCROLL	Displays a vertical scrollbar in the <i>client window</i> , allowing the user to view child frames that are off the current view.

See also *window styles overview* (p. 959).

See also

wxMDIChildFrame (p. 404), *wxMDIClientWindow* (p. 407), *wxFrame* (p. 275), *wxDialog* (p. 178)

4.114.1 wxMDIParentFrame::wxMDIParentFrame

wxMDIParentFrame()

Default constructor.

wxMDIParentFrame(wxWindow* parent, wxWindowID id, const wxString& title,

```
const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long  
style = wxDEFAULT_FRAME_STYLE | wxVSCROLL | wxHSCROLL, const wxString&  
name = "frame")
```

Constructor, creating the window.

Parameters

parent

The window parent. This should be NULL.

id

The window identifier. It may take a value of -1 to indicate a default value.

title

The caption to be displayed on the frame's title bar.

pos

The window position. A value of (-1, -1) indicates a default position, chosen by either the windowing system or wxWindows, depending on platform.

size

The window size. A value of (-1, -1) indicates a default size, chosen by either the windowing system or wxWindows, depending on platform.

style

The window style. See *wxMDIParentFrame* (p. 409).

name

The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

Remarks

During the construction of the frame, the client window will be created. To use a different class from *wxMDIClientWindow* (p. 407), override *wxMDIParentFrame::OnCreateClient* (p. 414).

Under Windows 95, the client window will automatically have a sunken border style when the active child is not maximized, and no border style when a child is maximized.

See also

wxMDIParentFrame::Create (p. 412), *wxMDIParentFrame::OnCreateClient* (p. 414)

4.114.2 **wxMDIParentFrame::~~wxMDIParentFrame**

~wxMDIParentFrame()

Destructor. Destroys all child windows and menu bar if present.

4.114.3 wxMDIParentFrame::ActivateNext

void ActivateNext()

Activates the MDI child following the currently active one.

[See also](#)

wxMDIParentFrame::ActivatePrevious (p. 412)

4.114.4 wxMDIParentFrame::ActivatePrevious

void ActivatePrevious()

Activates the MDI child preceding the currently active one.

[See also](#)

wxMDIParentFrame::ActivateNext (p. 412)

4.114.5 wxMDIParentFrame::ArrangeIcons

void ArrangeIcons()

Arranges any iconized (minimized) MDI child windows.

[See also](#)

wxMDIParentFrame::Cascade (p. 412), *wxMDIParentFrame::Tile* (p. 415)

4.114.6 wxMDIParentFrame::Cascade

void Cascade()

Arranges the MDI child windows in a cascade.

[See also](#)

wxMDIParentFrame::Tile (p. 415), *wxMDIParentFrame::ArrangeIcons* (p. 412)

4.114.7 wxMDIParentFrame::Create

bool Create(wxWindow* parent, wxWindowID id, const wxString& title, const

wxPoint& pos = wxDefaultPosition, **const wxSize&** size = wxDefaultSize, **long** style = wxDEFAULT_FRAME_STYLE | wxVSCROLL | wxHSCROLL, **const wxString&** name = "frame")

Used in two-step frame construction. See *wxMDIParentFrame::wxMDIParentFrame* (p. 410) for further details.

4.114.8 wxMDIParentFrame::GetClientSize

virtual void GetClientSize(int* width, int* height) const

This gets the size of the frame 'client area' in pixels.

Parameters

width

Receives the client width in pixels.

height

Receives the client height in pixels.

Remarks

The client area is the area which may be drawn on by the programmer, excluding title bar, border, status bar, and toolbar if present.

If you wish to manage your own toolbar (or perhaps you have more than one), provide an **OnSize** event handler. Call **GetClientSize** to find how much space there is for your windows and don't forget to set the size and position of the MDI client window as well as your toolbar and other windows (but not the status bar).

If you have set a toolbar with *wxMDIParentFrame::SetToolBar* (p. 415), the client size returned will have subtracted the toolbar height. However, the available positions for the client window and other windows of the frame do not start at zero - you must add the toolbar height.

The position and size of the status bar and toolbar (if known to the frame) are always managed by **wxMDIParentFrame**, regardless of what behaviour is defined in your **OnSize** event handler. However, the client window position and size are always set in **OnSize**, so if you override this event handler, make sure you deal with the client window.

You do not have to manage the size and position of MDI child windows, since they are managed automatically by the client window.

See also

wxMDIParentFrame::GetToolBar (p. 414), *wxMDIParentFrame::SetToolBar* (p. 415), *wxWindow* (p. 828), *wxMDIClientWindow* (p. 407)

wxPython note:

The wxPython version of this method takes no arguments and returns a tuple containing width and height.

4.114.9 wxMDIParentFrame::GetActiveChild

wxMDIChildFrame* GetActiveChild() const

Returns a pointer to the active MDI child, if there is one.

4.114.10 wxMDIParentFrame::GetClientWindow

wxMDIClientWindow* GetClientWindow() const

Returns a pointer to the client window.

[See also](#)

wxMDIParentFrame::OnCreateClient (p. 414)

4.114.11 wxMDIParentFrame::GetToolBar

virtual wxWindow* GetToolBar() const

Returns the window being used as the toolbar for this frame.

[See also](#)

wxMDIParentFrame::SetToolBar (p. 415)

4.114.12 wxMDIParentFrame::OnCreateClient

virtual wxMDIClientWindow* OnCreateClient()

Override this to return a different kind of client window.

Remarks

You might wish to derive from *wxMDIClientWindow* (p. 407) in order to implement different erase behaviour, for example, such as painting a bitmap on the background.

Note that it is probably impossible to have a client window that scrolls as well as painting a bitmap or pattern, since in **OnScroll**, the scrollbar positions always return zero. (Solutions to: julian.smart@ukonline.co.uk).

[See also](#)

wxMDIParentFrame::GetClientWindow (p. 414), *wxMDIClientWindow* (p. 407)

4.114.13 **wxMDIParentFrame::SetToolBar**

virtual void SetToolBar(wxWindow* *toolbar*)

Sets the window to be used as a toolbar for this MDI parent window. It saves the application having to manage the positioning of the toolbar MDI client window.

Parameters

toolbar

Toolbar to manage.

Remarks

When the frame is resized, the toolbar is resized to be the width of the frame client area, and the toolbar height is kept the same.

The parent of the toolbar must be this frame.

If you wish to manage your own toolbar (or perhaps you have more than one), don't call this function, and instead manage your subwindows and the MDI client window by providing an **OnSize** event handler. Call *wxMDIParentFrame::GetClientSize* (p. 413) to find how much space there is for your windows.

Note that SDI (normal) frames and MDI child windows must always have their toolbars managed by the application.

See also

wxMDIParentFrame::GetToolBar (p. 414), *wxMDIParentFrame::GetClientSize* (p. 413)

4.114.14 **wxMDIParentFrame::Tile**

void Tile()

Tiles the MDI child windows.

See also

wxMDIParentFrame::Cascade (p. 412), *wxMDIParentFrame::ArrangeIcons* (p. 412)

4.115 **wxMemoryDC**

A memory device context provides a means to draw graphics onto a bitmap.

Derived from

wxDC (p. 151)
wXObject (p. 471)

Include files

<wx/dcmemory.h>

Remarks

A bitmap must be selected into the new memory DC before it may be used for anything. Typical usage is as follows:

```
// Create a memory DC
wxMemoryDC temp_dc;
temp_dc.SelectObject(test_bitmap);

// We can now draw into the memory DC...
// Copy from this DC to another DC.
old_dc.Blit(250, 50, BITMAP_WIDTH, BITMAP_HEIGHT, temp_dc, 0, 0);
```

Note that the memory DC *must* be deleted (or the bitmap selected out of it) before a bitmap can be reselected into another memory DC.

See also

wxBitmap (p. 39), *wxDC* (p. 151)

4.115.1 wxMemoryDC::wxMemoryDC

wxMemoryDC()

Constructs a new memory device context.

Use the *Ok* member to test whether the constructor was successful in creating a useable device context. Don't forget to select a bitmap into the DC before drawing on it.

4.115.2 wxMemoryDC::SelectObject

SelectObject(const wxBitmap& *bitmap*)

Selects the given bitmap into the device context, to use as the memory bitmap. Selecting the bitmap into a memory DC allows you to draw into the DC (and therefore the bitmap) and also to use **Blit** to copy the bitmap to a window. For this purpose, you may find *wxDC::DrawIcon* (p. 155) easier to use instead.

If the argument is *wxNullBitmap* (or some other uninitialised *wxBitmap*) the current bitmap is selected out of the device context, and the original bitmap restored, allowing

the current bitmap to be destroyed safely.

4.116 wxMemoryInputStream

Derived from

wxInputStream (p. 346)

Include files

<wx/mstream.h>

See also

wxStreamBuffer (p. 648)

Remark

You can create a similar stream by this way:

```
wxStreamBuffer *sb = new wxStreamBuffer(wxStreamBuffer::read);  
wxInputStream *input = new wxInputStream(sb);  
  
sb->SetBufferIO(data, data+_end);
```

4.116.1 wxMemoryInputStream::wxMemoryInputStream

wxMemoryInputStream(const char * data, size_t len)

Initializes a new read-only memory stream which will use the specified buffer *data* of length *len*.

4.116.2 wxMemoryInputStream::~~wxMemoryInputStream

~wxFileInputStream()

Destructor.

4.117 wxMemoryOutputStream

Derived from

wxOutputStream (p. 476)

Include files

<wx/mstream.h>

See also

wxStreamBuffer (p. 648)

Remark

You can create a similar stream by this way:

```
wxStreamBuffer *sb = new wxStreamBuffer(wxStreamBuffer::write);
wxOutputStream *input = new wxOutputStream(sb);

// If there are data
sb->SetBufferIO(data, data\_end);
// Else
sb->Fixed(FALSE);
```

This way is also useful to create read/write memory stream:

```
wxStreamBuffer *sb = new wxStreamBuffer(wxStreamBuffer::read\_write);
wxOutputStream *output = new wxOutputStream(sb);
wxInputStream *input = new wxInputStream(sb);

// If there are data
sb->SetBufferIO(data, data\_end);
// Else
sb->Fixed(FALSE);
```

4.117.1 **wxMemoryOutputStream::wxMemoryOutputStream**

wxMemoryOutputStream(char * data = NULL, size_t length = 0)

If *data* is NULL, then it will initialize a new empty buffer which will grow when it needs.

Warning

If the buffer is created, it will be destroyed at the destruction of the stream.

4.117.2 **wxMemoryOutputStream::~~wxMemoryOutputStream**

~wxMemoryOutputStream()

Destructor.

4.118 **wxMenu**

A menu is a popup (or pull down) list of items, one of which may be selected before the menu goes away (clicking elsewhere dismisses the menu). Menus may be used to construct either menu bars or popup menus.

A menu item has an integer ID associated with it which can be used to identify the selection, or to change the menu item in some way.

Derived from

wxEvtHandler (p. 224)

wxObject (p. 471)

Include files

<wx/menu.h>

Event handling

If the menu is part of a menubar, then *wxMenuBar* (p. 426) event processing is used.

With a popup menu, there is a variety of ways to handle a menu selection event (*wxEVT_COMMAND_MENU_SELECTED*).

1. Define a callback of type *wxFunction*, which you pass to the *wxMenu* constructor. The callback takes a reference to the menu, and a reference to a *wxCommandEvent* (p. 103).
2. Derive a new class from *wxMenu* and define event table entries using the *EVT_MENU* macro.
3. Set a new event handler for *wxMenu*, using an object whose class has *EVT_MENU* entries.
4. Provide *EVT_MENU* handlers in the window which pops up the menu, or in an ancestor of this window.

See also

wxMenuBar (p. 426), *wxWindow::PopupMenu* (p. 829), *Event handling overview* (p. 939)

4.118.1 **wxMenu::wxMenu**

wxMenu(const wxString& title = "", const wxFunction func = NULL)

Constructs a *wxMenu* object.

Parameters

title

A title for the popup menu: the empty string denotes no title.

func

A callback function if the menu is used as a popup using *wxWindow::PopupMenu* (p. 829).

wxPython note:

The wxPython version of the `wxMenu` constructor doesn't accept the callback argument because of reference counting issues. There is a specialized `wxMenu` constructor called `wxPyMenu` which does and can be used for `PopupMenu`s when callbacks are needed. You must retain a reference to the menu while using it otherwise your callback function will get dereferenced when the menu does.

4.118.2 wxMenu::~~wxMenu**~wxMenu()**

Destructor, destroying the menu.

Note: under Motif, a popup menu must have a valid parent (the window it was last popped up on) when being destroyed. Therefore, make sure you delete or re-use the popup menu *before* destroying the parent window. Re-use in this context means popping up the menu on a different window from last time, which causes an implicit destruction and recreation of internal data structures.

4.118.3 wxMenu::Append

void Append(int id, const wxString& item, const wxString& helpString = "", const bool checkable = FALSE)

Adds a string item to the end of the menu.

void Append(int id, const wxString& item, wxMenu *subMenu, const wxString& helpString = "")

Adds a pull-right submenu to the end of the menu.

void Append(wxMenuItem* menuItem)

Adds a menu item object. You can specify various extra properties of a menu item this way, such as bitmaps and fonts.

Parameters

id

The menu command identifier.

item

The string to appear on the menu item.

menu

Pull-right submenu.

checkable

If TRUE, this item is checkable.

helpString

An optional help string associated with the item. By default, *wxFrame::OnMenuHighlight* (p. 283) displays this string in the status line.

menuItem

A menuitem object. It will be owned by the *wxMenu* object after this function is called, so do not delete it yourself.

Remarks

This command can be used after the menu has been shown, as well as on initial creation of a menu or menubar.

See also

wxMenu::AppendSeparator (p. 421), *wxMenu::SetLabel* (p. 425), *wxMenu::GetHelpString* (p. 423), *wxMenu::SetHelpString* (p. 425), *wxMenuItem* (p. 433)

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

Append(id, string, helpStr="", checkable=FALSE)

AppendMenu(id, string, aMenu, helpStr="")

AppendItem(aMenuItem)

4.118.4 wxMenu::AppendSeparator

void AppendSeparator()

Adds a separator to the end of the menu.

See also

wxMenu::Append (p. 420)

4.118.5 wxMenu::Break

void Break()

Inserts a break in a menu, causing the next appended item to appear in a new column.

4.118.6 wxMenu::Check

void Check(int *id*, const bool *check*)

Checks or unchecks the menu item.

Parameters

id

The menu item identifier.

check

If TRUE, the item will be checked, otherwise it will be unchecked.

See also

wxMenu::IsChecked (p. 424)

4.118.7 wxMenu::Enable

void Enable(int *id*, const bool *enable*)

Enables or disables (greys out) a menu item.

Parameters

id

The menu item identifier.

enable

TRUE to enable the menu item, FALSE to disable it.

See also

wxMenu::IsEnabled (p. 425)

4.118.8 wxMenu::FindItem

int FindItem(const wxString& *itemString*) const

Finds the menu item id for a menu item string.

Parameters

itemString

Menu item string to find.

Return value

Menu item identifier, or -1 if none is found.

Remarks

Any special menu codes are stripped out of source and target strings before matching.

See also

wxMenu::FindItemForId (p. 423)

4.118.9 **wxMenu::FindItemForId**

wxMenuItem* FindItemForId(int id) const

Finds the menu item object associated with the given menu item identifier.

Parameters

id

Menu item identifier.

Return value

Returns the menu item object, or NULL if it is not found.

See also

wxMenu::FindItem (p. 422)

4.118.10 **wxMenu::GetHelpString**

wxString GetHelpString(int id) const

Returns the help string associated with a menu item.

Parameters

id

The menu item identifier.

Return value

The help string, or the empty string if there is no help string or the item was not found.

See also

wxMenu::SetHelpString (p. 425), *wxMenu::Append* (p. 420)

4.118.11 wxMenu::GetLabel**wxString GetLabel(int *id*) const**

Returns a menu item label.

Parameters*id*

The menu item identifier.

Return value

The item label, or the empty string if the item was not found.

See also

wxMenu::SetLabel (p. 425)

4.118.12 wxMenu::GetTitle**wxString GetTitle() const**

Returns the title of the menu.

Remarks

This is relevant only to popup menus.

See also

wxMenu::SetTitle (p. 426)

4.118.13 wxMenu::IsChecked**bool IsChecked(int *id*) const**

Determines whether a menu item is checked.

Parameters*id*

The menu item identifier.

Return value

TRUE if the menu item is checked, FALSE otherwise.

See also

wxMenu::Check (p. 422)

4.118.14 wxMenu::IsEnabled

bool IsEnabled(int *id*) const

Determines whether a menu item is enabled.

Parameters

id

The menu item identifier.

Return value

TRUE if the menu item is enabled, FALSE otherwise.

See also

wxMenu::Enable (p. 422)

4.118.15 wxMenu::SetHelpString

void SetHelpString(int *id*, const wxString& *helpString*)

Sets an item's help string.

Parameters

id

The menu item identifier.

helpString

The help string to set.

See also

wxMenu::GetHelpString (p. 423)

4.118.16 wxMenu::SetLabel

void SetLabel(int *id*, const wxString& *label*)

Sets the label of a menu item.

Parameters

id

The menu item identifier.

label

The menu item label to set.

See also

wxMenu::Append (p. 420), *wxMenu::GetLabel* (p. 424)

4.118.17 **wxMenu::SetTitle**

void SetTitle(const wxString& title)

Sets the title of the menu.

Parameters

title

The title to set.

Remarks

This is relevant only to popup menus.

See also

wxMenu::SetTitle (p. 426)

4.118.18 **wxMenu::UpdateUI**

void UpdateUI(wxEvtHandler* source = NULL) const

Sends events to *source* (or owning window if NULL) to update the menu UI. This is called just before the menu is popped up with *wxWindow::PopupMenu* (p. 829), but the application may call it at other times if required.

See also

wxUpdateUIEvent (p. 775)

4.119 **wxMenuBar**

A menu bar is a series of menus accessible from the top of a frame.

Derived from

wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/menu.h>

Event handling

To respond to a menu selection, provide a handler for EVT_MENU, in the frame that contains the menu bar.

See also

wxMenu (p. 418), *Event handling overview* (p. 939)

4.119.1 wxMenuBar::wxMenuBar

void wxMenuBar()

Default constructor.

void wxMenuBar(int *n*, wxMenu* *menus*[], const wxString *titles*[])

Construct a menu bar from arrays of menus and titles.

Parameters

n

The number of menus.

menus

An array of menus. Do not use this array again - it now belongs to the menu bar.

titles

An array of title strings. Deallocate this array after creating the menu bar.

wxPython note:

Only the default constructor is supported in wxPython. Use wxMenuBar.Append instead.

4.119.2 wxMenuBar::~~wxMenuBar

void ~wxMenuBar()

Destructor, destroying the menu bar and removing it from the parent frame (if any).

4.119.3 **wxMenuBar::Append**

void Append(**wxMenu** **menu*, **const wxString&** *title*)

Adds the item to the end of the menu bar.

Parameters

menu

The menu to add. Do not deallocate this menu after calling **Append**.

title

The title of the menu.

4.119.4 **wxMenuBar::Check**

void Check(**int** *id*, **const bool** *check*)

Checks or unchecks a menu item.

Parameters

id

The menu item identifier.

check

If TRUE, checks the menu item, otherwise the item is unchecked.

Remarks

Only use this when the menu bar has been associated with a frame; otherwise, use the **wxMenu** equivalent call.

4.119.5 **wxMenuBar::Enable**

void Enable(**int** *id*, **const bool** *enable*)

Enables or disables (greys out) a menu item.

Parameters

id

The menu item identifier.

enable

TRUE to enable the item, FALSE to disable it.

Remarks

Only use this when the menu bar has been associated with a frame; otherwise, use the `wxMenu` equivalent call.

4.119.6 `wxMenuBar::EnableTop`

void EnableTop(int *pos*, const bool *enable*)

Enables or disables a whole menu.

Parameters

pos

The position of the menu, starting from zero.

enable

TRUE to enable the menu, FALSE to disable it.

Remarks

Only use this when the menu bar has been associated with a frame.

4.119.7 `wxMenuBar::FindMenuItem`

int FindMenuItem(const wxString& *menuString*, const wxString& *itemString*) const

Finds the menu item id for a menu name/menu item string pair.

Parameters

menuString

Menu title to find.

itemString

Item to find.

Return value

The menu item identifier, or -1 if none was found.

Remarks

Any special menu codes are stripped out of source and target strings before matching.

4.119.8 wxMenuBar::FindItemById**wxMenuItem * FindItemById(int id) const**

Finds the menu item object associated with the given menu item identifier,

Parameters*id*

Menu item identifier.

Return value

The found menu item object, or NULL if one was not found.

4.119.9 wxMenuBar::GetHelpString**wxString GetHelpString(int id) const**

Gets the help string associated with the menu item identifier.

Parameters*id*

The menu item identifier.

Return value

The help string, or the empty string if there was no help string or the menu item was not found.

See also

wxMenuBar::SetHelpString (p. 432)

4.119.10 wxMenuBar::GetLabel**wxString GetLabel(int id) const**

Gets the label associated with a menu item.

Parameters*id*

The menu item identifier.

Return value

The menu item label, or the empty string if the item was not found.

Remarks

Use only after the menubar has been associated with a frame.

4.119.11 **wxMenuBar::GetLabelTop**

wxString GetLabelTop(int pos) const

Returns the label of a top-level menu.

Parameters

pos

Position of the menu on the menu bar, starting from zero.

Return value

The menu label, or the empty string if the menu was not found.

Remarks

Use only after the menubar has been associated with a frame.

See also

wxMenuBar::SetLabelTop (p. 433)

4.119.12 **wxMenuBar::GetMenu**

wxMenu* GetMenu(int menuIndex) const

Returns the menu at *menuIndex* (zero-based).

4.119.13 **wxMenuBar::GetMenuCount**

int GetMenuCount() const

Returns the number of menus in this menubar.

4.119.14 **wxMenuBar::IsChecked**

bool IsChecked(int id) const

Determines whether an item is checked.

Parameters

id
The menu item identifier.

Return value

TRUE if the item was found and is checked, FALSE otherwise.

4.119.15 **wxMenuBar::IsEnabled**

bool IsEnabled(int *id*) const

Determines whether an item is enabled.

Parameters

id
The menu item identifier.

Return value

TRUE if the item was found and is enabled, FALSE otherwise.

4.119.16 **wxMenuBar::SetHelpString**

void SetHelpString(int *id*, const wxString& *helpString*)

Sets the help string associated with a menu item.

Parameters

id
Menu item identifier.

helpString
Help string to associate with the menu item.

See also

wxMenuBar::GetHelpString (p. 430)

4.119.17 **wxMenuBar::SetLabel**

void SetLabel(int *id*, const wxString& *label*)

Sets the label of a menu item.

Parameters

id

Menu item identifier.

label

Menu item label.

Remarks

Use only after the menubar has been associated with a frame.

See also

wxMenuBar::GetLabel (p. 430)

4.119.18 **wxMenuBar::SetLabelTop**

void SetLabelTop(int *pos*, const wxString& *label*)

Sets the label of a top-level menu.

Parameters

pos

The position of a menu on the menu bar, starting from zero.

label

The menu label.

Remarks

Use only after the menubar has been associated with a frame.

See also

wxMenuBar::GetLabelTop (p. 431)

4.120 **wxMenuItem**

A menu item represents an item in a popup menu. Note that the majority of this class is only implemented under Windows so far, but everything except fonts, colours and bitmaps can be achieved via *wxMenu* on all platforms.

Derived from

wxOwnerDrawn (Windows only)

wxObject (p. 471)

Include files

<wx/menuitem.h>

See also

wxMenuBar (p. 426), *wxMenu* (p. 418)

4.120.1 **wxMenuItem::wxMenuItem**

wxMenuItem(*wxMenu** parentMenu = NULL, **int** id = ID_SEPARATOR, **const wxString&** text = "", **const wxString&** helpString = "", **bool** checkable = FALSE, *wxMenu** subMenu = NULL,)

Constructs a wxMenuItem object.

Parameters

parentMenu

Menu that the menu item belongs to.

id

Identifier for this menu item, or ID_SEPARATOR to indicate a separator.

text

Text for the menu item, as shown on the menu.

helpString

Optional help string that will be shown on the status bar.

checkable

TRUE if this menu item is checkable.

subMenu

If non-NULL, indicates that the menu item is a submenu.

4.120.2 **wxMenuItem::~~wxMenuItem**

~wxMenuItem()

Destructor.

4.120.3 **wxMenuItem::Check**

void Check(bool *check*)

Checks or unchecks the menu item.

4.120.4 wxMenuItem::DeleteSubMenu

void DeleteSubMenu()

Deletes the submenu, if any.

4.120.5 wxMenuItem::Enable

void Enable(bool *enable*)

Enables or disables the menu item.

4.120.6 wxMenuItem::GetBackgroundColour

wxColour& GetBackgroundColour() const

Returns the background colour associated with the menu item (Windows only).

4.120.7 wxMenuItem::GetBitmap

wxBitmap& GetBitmap(bool *checked* = TRUE) **const**

Returns the checked or unchecked bitmap (Windows only).

4.120.8 wxMenuItem::GetFont

wxFont& GetFont() const

Returns the font associated with the menu item (Windows only).

4.120.9 wxMenuItem::GetHelp

wxString GetHelp() const

Returns the help string associated with the menu item.

4.120.10 wxMenuItem::GetId

int GetId() const

Returns the menu item identifier.

4.120.11 wxMenuItem::GetMarginWidth

int GetMarginWidth() const

Gets the width of the menu item checkmark bitmap (Windows only).

4.120.12 wxMenuItem::GetName

wxString GetName() const

Returns the text associated with the menu item.

4.120.13 wxMenuItem::GetSubMenu

wxMenu* GetSubMenu() const

Returns the submenu associated with the menu item, or NULL if there isn't one.

4.120.14 wxMenuItem::GetTextColour

wxColour& GetTextColour() const

Returns the text colour associated with the menu item (Windows only).

4.120.15 wxMenuItem::IsCheckable

bool IsCheckable() const

Returns TRUE if the item is checkable.

4.120.16 wxMenuItem::IsChecked

bool IsChecked() const

Returns TRUE if the item is checked.

4.120.17 wxMenuItem::IsEnabled

bool IsEnabled() const

Returns TRUE if the item is enabled.

4.120.18 wxMenuItem::IsSeparator

bool IsSeparator() const

Returns TRUE if the item is a separator.

4.120.19 wxMenuItem::SetBackgroundColour

void SetBackgroundColour(const wxColour& *colour*) const

Sets the background colour associated with the menu item (Windows only).

4.120.20 wxMenuItem::SetBitmaps

void SetBitmaps(const wxBitmap& *checked*, const wxBitmap& *unchecked* = wxNullBitmap) const

Sets the checked/unchecked bitmaps for the menu item (Windows only). The first bitmap is also used as the single bitmap for uncheckable menu items.

4.120.21 wxMenuItem::SetFont

void SetFont(const wxFont& *font*) const

Sets the font associated with the menu item (Windows only).

4.120.22 wxMenuItem::SetHelp

void SetHelp(const wxString& *helpString*) const

Sets the help string.

4.120.23 wxMenuItem::SetMarginWidth

void SetMarginWidth(int *width*) const

Sets the width of the menu item checkmark bitmap (Windows only).

4.120.24 wxMenuItem::SetName

void SetName(const wxString& *text*) const

Sets the text associated with the menu item.

4.120.25 **wxMenuItem::SetTextColour**

void SetTextColour(const wxColour& colour) const

Sets the text colour associated with the menu item (Windows only).

4.121 **wxMenuEvent**

This class is used for a variety of menu-related events. Note that these do not include menu command events.

Derived from

wxEvent (p. 221)

wxObject (p. 471)

Include files

<wx/event.h>

Event table macros

To process a menu event, use these event handler macros to direct input to member functions that take a *wxMenuEvent* argument.

EVT_MENU_CHAR(func)	Process a <i>wxEVT_MENU_CHAR</i> event (a keypress when a menu is showing). Windows only; not yet implemented.
EVT_MENU_INIT(func)	Process a <i>wxEVT_MENU_INIT</i> event (the menu is about to pop up). Windows only; not yet implemented.
EVT_MENU_HIGHLIGHT(func)	Process a <i>wxEVT_MENU_HIGHLIGHT</i> event (a menu item is being highlighted). Windows only; not yet implemented.
EVT_POPUP_MENU(func)	Process a <i>wxEVT_POPUP_MENU</i> event (a menu item is being highlighted). Windows only; not yet implemented.
EVT_CONTEXT_MENU(func)	Process a <i>wxEVT_CONTEXT_MENU</i> event (F1 has been pressed with a particular menu item highlighted). Windows only; not yet implemented.

See also

wxWindow::OnMenuHighlight (p. 824), *Event handling overview* (p. 939)

4.121.1 wxMenuEvent::wxMenuEvent**wxMenuEvent**(WXTYPE *id* = 0, int *id* = 0, wxDC* *dc* = NULL)

Constructor.

4.121.2 wxMenuEvent::m_menuId

int m_menuId

The relevant menu identifier.

4.121.3 wxMenuEvent::GetMenuId

int GetMenuId() const

Returns the menu identifier associated with the event.

4.122 wxMessageDialog

This class represents a dialog that shows a single or multi-line message, with a choice of OK, Yes, No and Cancel buttons.

Derived from

wxDialog (p. 178)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/msgdlg.h>

See also*wxMessageDialog* overview (p. 918)**4.122.1 wxMessageDialog::wxMessageDialog****wxMessageDialog**(wxWindow* *parent*, const wxString& *message*, const wxString& *caption* = "Message box", long *style* = wxOK | wxCANCEL | wxCENTRE, const

wxPoint& *pos* = *wxDefaultPosition*)

Constructor. Use *wxMessageDialog::ShowModal* (p. 440) to show the dialog.

Parameters

parent

Parent window.

message

Message to show on the dialog.

caption

The dialog caption.

style

A dialog style (bitlist) containing flags chosen from the following:

wxOK	Show an OK button.
wxCANCEL	Show a Cancel button.
wxYES_NO	Show Yes and No buttons.
wxCENTRE	Centre the message. Not Windows.
wxICON_EXCLAMATION	Shows an exclamation mark icon. Windows only.
wxICON_HAND	Shows a hand icon. Windows only.
wxICON_QUESTION	Shows a question mark icon. Windows only.
wxICON_INFORMATION	Shows an information (i) icon. Windows only.

pos

Dialog position. Not Windows.

4.122.2 **wxMessageDialog::~wxMessageDialog**

~wxMessageDialog()

Destructor.

4.122.3 **wxMessageDialog::ShowModal**

int ShowModal()

Shows the dialog, returning one of *wxID_OK*, *wxID_CANCEL*, *wxID_YES*, *wxID_NO*.

4.123 **wxMetafile**

A **wxMetafile** represents the MS Windows metafile object, so metafile operations have no effect in X. In *wxWindows*, only sufficient functionality has been provided for copying a graphic to the clipboard; this may be extended in a future version. Presently, the only

way of creating a metafile is to use a `wxMetafileDC`.

Derived from

wxObject (p. 471)

Include files

<wx/metafile.h>

See also

wxMetafileDC (p. 442)

4.123.1 `wxMetafile::wxMetafile`

`wxMetafile(const wxString& filename = "")`

Constructor. If a filename is given, the Windows disk metafile is read in. Check whether this was performed successfully by using the *wxMetafile::Ok* (p. 441) member.

4.123.2 `wxMetafile::~~wxMetafile`

`~wxMetafile()`

Destructor.

4.123.3 `wxMetafile::Ok`

`bool Ok()`

Returns TRUE if the metafile is valid.

4.123.4 `wxMetafile::Play`

`bool Play(wxDC *dc)`

Plays the metafile into the given device context, returning TRUE if successful.

4.123.5 `wxMetafile::SetClipboard`

`bool SetClipboard(int width = 0, int height = 0)`

Passes the metafile data to the clipboard. The metafile can no longer be used for anything, but the `wxMetafile` object must still be destroyed by the application.

Below is a example of metafile, metafile device context and clipboard use from the `hello.cpp` example. Note the way the metafile dimensions are passed to the clipboard, making use of the device context's ability to keep track of the maximum extent of drawing commands.

```
wxMetafileDC dc;
if (dc.Ok())
{
    Draw(dc, FALSE);
    wxMetafile *mf = dc.Close();
    if (mf)
    {
        bool success = mf->SetClipboard((int)(dc.MaxX() + 10),
(int)(dc.MaxY() + 10));
        delete mf;
    }
}
```

4.124

wxMetafileDC

This is a type of device context that allows a metafile object to be created (Windows only), and has most of the characteristics of a normal **wxDC**. The `wxMetafileDC::Close` (p. 443) member must be called after drawing into the device context, to return a metafile. The only purpose for this at present is to allow the metafile to be copied to the clipboard (see *wxMetafile* (p. 440)).

Adding metafile capability to an application should be easy if you already write to a `wxDC`; simply pass the `wxMetafileDC` to your drawing function instead. You may wish to conditionally compile this code so it is not compiled under X (although no harm will result if you leave it in).

Note that a metafile saved to disk is in standard Windows metafile format, and cannot be imported into most applications. To make it importable, call the function `::wxMakeMetafilePlaceable` (p. 856) after closing your disk-based metafile device context.

Derived from

wxDC (p. 151)
wxObject (p. 471)

Include files

<wx/metafile.h>

See also

wxMetafile (p. 440), *wxDC* (p. 151)

4.124.1 **wxMetafileDC::wxMetafileDC**

wxMetafileDC(const wxString& *filename* = "")

Constructor. If no filename is passed, the metafile is created in memory.

4.124.2 **wxMetafileDC::~~wxMetafileDC**

~wxMetafileDC()

Destructor.

4.124.3 **wxMetafileDC::Close**

wxMetafile * Close()

This must be called after the device context is finished with. A metafile is returned, and ownership of it passes to the calling application (so it should be destroyed explicitly).

4.125 **wxMimeTypeManager**

This class allows the application to retrieve the information about all known MIME types from a system-specific location and the filename extensions to the MIME types and vice versa. After initialization the functions *wxMimeTypeManager::GetFileTypeFromMimeType* (p. 445) and *wxMimeTypeManager::GetFileTypeFromExtension* (p. 445) may be called: they will return a *wxFileType* (p. 258) object which may be further queried for file description, icon and other attributes.

Windows: MIME type information is stored in the registry and no additional initialization is needed.

Unix: MIME type information is stored in the files mailcap and mime.types (system-wide) and .mailcap and .mime.types in the current user's home directory: all of these files are searched for and loaded if found by default. However, additional functions *wxMimeTypeManager::ReadMailcap* (p. 445) and *wxMimeTypeManager::ReadMimeType* (p. 446) are provided to load additional files.

NB: Currently, *wxMimeTypeManager* is limited to reading MIME type information but it will support modifying it as well in the future versions.

Derived from

No base class.

Include files

<wx/mimetype.h>

See also

wxFileType (p. 258)

4.125.1 Helper functions

All of these functions are static (i.e. don't need a *wxMimeTypesManager* object to call them) and provide some useful operations for string representations of MIME types. Their usage is recommended instead of directly working with MIME types using *wxString* functions.

IsOfType (p. 445)

4.125.2 Constructor and destructor

NB: You won't normally need to use more than one *wxMimeTypesManager* object in a program.

wxMimeTypesManager (p. 445)

~wxMimeTypesManager (p. 445)

4.125.3 Query database

These functions are the heart of this class: they allow to find a *file type* (p. 258) object from either file extension or MIME type. If the function is successful, it returns a pointer to the *wxFileType* object which **must** be deleted by the caller, otherwise NULL will be returned.

GetFileTypeFromMimeType (p. 445)

GetFileTypeFromExtension (p. 445)

4.125.4 Initialization functions

Unix: These functions may be used to load additional files (except for the default ones which are loaded automatically) containing MIME information in either mailcap(5) or mime.types(5) format.

ReadMailcap (p. 445)

ReadMimeTypes (p. 446)

4.125.5 wxMimeTypesManager::wxMimeTypesManager**wxMimeTypesManager()**

Constructor puts the object in the "working" state, no additional initialization are needed - but *ReadXXX* (p. 444) may be used to load additional mailcap/mime.types files.

4.125.6 wxMimeTypesManager::~~wxMimeTypesManager**~wxMimeTypesManager()**

Destructor is not virtual, so this class should not be derived from.

4.125.7 wxMimeTypesManager::GetFileTypeFromExtension**wxFileType* GetFileTypeFromExtension(const wxString& *extension*)**

Gather information about the files with given extension and return the corresponding *wxFileType* (p. 258) object or NULL if the extension is unknown.

4.125.8 wxMimeTypesManager::GetFileTypeFromMimeType**wxFileType* GetFileTypeFromMimeType(const wxString& *mimeType*)**

Gather information about the files with given MIME type and return the corresponding *wxFileType* (p. 258) object or NULL if the MIME type is unknown.

4.125.9 wxMimeTypesManager::IsOfType**bool IsOfType(const wxString& *mimeType*, const wxString& *wildcard*)**

This function returns TRUE if either the given *mimeType* is exactly the same as *wildcard* or if it has the same category and the subtype of *wildcard* is '*'. Note that the '*' wildcard is not allowed in *mimeType* itself.

The comparison done by this function is case insensitive so it is not necessary to convert the strings to the same case before calling it.

4.125.10 wxMimeTypesManager::ReadMailcap**void ReadMailcap(const wxString& *filename*)**

Load additional file containing information about MIME types and associated information in mailcap format. See `metamail(1)` and `mailcap(5)` for more information.

4.125.11 `wxMimeTypesManager::ReadMimeTypes`

void ReadMimeTypes(const wxString& filename)

Load additional file containing information about MIME types and associated information in mime.types file format. See `metamail(1)` and `mailcap(5)` for more information.

4.126 `wxMiniFrame`

A miniframe is a frame with a small title bar. It is suitable for floating toolbars that must not take up too much screen area.

Derived from

`wxFrame` (p. 275)
`wxWindow` (p. 798)
`wxEvtHandler` (p. 224)
`wxObject` (p. 471)

Include files

<wx/minifram.h>

Window styles

wxICONIZE	Display the frame iconized (minimized) (Windows only).
wxCAPTION	Puts a caption on the frame.
wxDEFAULT_FRAME_STYLE	Defined as wxMINIMIZE_BOX wxMAXIMIZE_BOX wxTHICK_FRAME wxSYSTEM_MENU wxCAPTION .
wxMINIMIZE	Identical to wxICONIZE .
wxMINIMIZE_BOX	Displays a minimize box on the frame (Windows and Motif only).
wxMAXIMIZE	Displays the frame maximized (Windows only).
wxMAXIMIZE_BOX	Displays a maximize box on the frame (Windows and Motif only).
wxSTAY_ON_TOP	Stay on top of other windows (Windows only).
wxSYSTEM_MENU	Displays a system menu (Windows and Motif only).
wxTHICK_FRAME	Displays a thick frame around the window (Windows and Motif only).
wxTINY_CAPTION_HORIZ	Displays a small horizontal caption. Use instead of wxCAPTION .
wxTINY_CAPTION_VERT	Under Windows, displays a small vertical caption. Use instead of wxCAPTION .
wxRESIZE_BORDER	Displays a resizable border around the window (Motif only; for Windows, it is implicit in wxTHICK_FRAME).

See also *window styles overview* (p. 959). Note that all the window styles above are ignored under GTK and the mini frame cannot be resized by the user.

Remarks

This class has miniframe functionality under Windows and GTK, i.e. the presence of mini frame will not be noted in the task bar and focus behaviour is different. On other platforms, it behaves like a normal frame.

See also

wxMDIParentFrame (p. 409), *wxMDIChildFrame* (p. 404), *wxFrame* (p. 275), *wxDialog* (p. 178)

4.126.1 wxMiniFrame::wxMiniFrame

wxMiniFrame()

Default constructor.

wxMiniFrame(**wxWindow*** *parent*, **wxWindowID** *id*, **const wxString&** *title*, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = *wxDEFAULT_FRAME_STYLE*, **const wxString&** *name* = "frame")

Constructor, creating the window.

Parameters

parent

The window parent. This may be NULL. If it is non-NULL, the frame will always be displayed on top of the parent window on Windows.

id

The window identifier. It may take a value of -1 to indicate a default value.

title

The caption to be displayed on the frame's title bar.

pos

The window position. A value of (-1, -1) indicates a default position, chosen by either the windowing system or wxWindows, depending on platform.

size

The window size. A value of (-1, -1) indicates a default size, chosen by either the windowing system or wxWindows, depending on platform.

style

The window style. See *wxMiniFrame* (p. 446).

name

The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

Remarks

The frame behaves like a normal frame on non-Windows platforms.

See also

wxMiniFrame::Create (p. 448)

4.126.2 wxMiniFrame::~~wxMiniFrame

void ~wxMiniFrame()

Destructor. Destroys all child windows and menu bar if present.

4.126.3 wxMiniFrame::Create

bool Create(wxWindow* parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT_FRAME_STYLE, const wxString& name = "frame")

Used in two-step frame construction. See *wxMiniFrame::wxMiniFrame* (p. 447) for further details.

4.127 wxModule

The module system is a very simple mechanism to allow applications (and parts of wxWindows itself) to define initialization and cleanup functions that are automatically called on wxWindows startup and exit.

To define a new kind of module, derive a class from wxModule, override the OnInit and OnExit functions, and add the DECLARE_DYNAMIC_CLASS and IMPLEMENT_DYNAMIC_CLASS to header and implementation files (which can be the same file). On initialization, wxWindows will find all classes derived from wxModule, create an instance of each, and call each OnInit function. On exit, wxWindows will call the OnExit function for each module instance.

Note that your module class does not have to be in a header file.

For example:

```
// A module to allow DDE initialization/cleanup
// without calling these functions from app.cpp or from
```



```
// the user's application.

class wxDDEModule: public wxModule
{
    DECLARE_DYNAMIC_CLASS(wxDDEModule)
public:
    wxDDEModule() {}
    bool OnInit() { wxDDEInitialize(); return TRUE; };
    void OnExit() { wxDDECleanUp(); };
};

IMPLEMENT_DYNAMIC_CLASS(wxDDEModule, wxModule)
```

Derived from

wxObject (p. 471)

Include files

<wx/module.h>

4.127.1 **wxModule::wxModule**

wxModule()

Constructs a wxModule object.

4.127.2 **wxModule::~~wxModule**

~wxModule()

Destructor.

4.127.3 **wxModule::CleanupModules**

static void CleanupModules()

Calls Exit for each module instance. Called by wxWindows on exit, so there is no need for an application to call it.

4.127.4 **wxModule::Exit**

void Exit()

Calls OnExit. This function is called by wxWindows and should not need to be called by an application.

4.127.5 wxModule::Init**bool Init()**

Calls OnInit. This function is called by wxWindows and should not need to be called by an application.

4.127.6 wxModule::InitializeModules**static bool InitializeModules()**

Calls Init for each module instance. Called by wxWindows on startup, so there is no need for an application to call it.

4.127.7 wxModule::OnExit**virtual void OnExit()**

Provide this function with appropriate cleanup for your module.

4.127.8 wxModule::OnInit**virtual bool OnInit()**

Provide this function with appropriate initialization for your module. If the function returns FALSE, wxWindows will exit immediately.

4.127.9 wxModule::RegisterModule**static void RegisterModule(wxModule* module)**

Registers this module with wxWindows. Called by wxWindows on startup, so there is no need for an application to call it.

4.127.10 wxModule::RegisterModules**static bool RegisterModules()**

Creates instances of and registers all modules. Called by wxWindows on startup, so there is no need for an application to call it.

4.128 wxMouseEvent

This event class contains information about mouse events. See *wxWindow::OnMouseEvent* (p. 824).

Derived from

wxEvent (p. 221)

Include files

<wx/event.h>

Event table macros

To process a mouse event, use these event handler macros to direct input to member functions that take a *wxMouseEvent* argument.

EVT_LEFT_DOWN(func)	Process a <i>wxEVT_LEFT_DOWN</i> event.
EVT_LEFT_UP(func)	Process a <i>wxEVT_LEFT_UP</i> event.
EVT_LEFT_DCLICK(func)	Process a <i>wxEVT_LEFT_DCLICK</i> event.
EVT_MIDDLE_DOWN(func)	Process a <i>wxEVT_MIDDLE_DOWN</i> event.
EVT_MIDDLE_UP(func)	Process a <i>wxEVT_MIDDLE_UP</i> event.
EVT_MIDDLE_DCLICK(func)	Process a <i>wxEVT_MIDDLE_DCLICK</i> event.
EVT_RIGHT_DOWN(func)	Process a <i>wxEVT_RIGHT_DOWN</i> event.
EVT_RIGHT_UP(func)	Process a <i>wxEVT_RIGHT_UP</i> event.
EVT_RIGHT_DCLICK(func)	Process a <i>wxEVT_RIGHT_DCLICK</i> event.
EVT_MOTION(func)	Process a <i>wxEVT_MOTION</i> event.
EVT_ENTER_WINDOW(func)	Process a <i>wxEVT_ENTER_WINDOW</i> event.
EVT_LEAVE_WINDOW(func)	Process a <i>wxEVT_LEAVE_WINDOW</i> event.
EVT_MOUSE_EVENTS(func)	Process all mouse events.

4.128.1 **wxMouseEvent::m_altDown**

bool m_altDown

TRUE if the Alt key is pressed down.

4.128.2 **wxMouseEvent::m_controlDown**

bool m_controlDown

TRUE if control key is pressed down.

4.128.3 **wxMouseEvent::m_leftDown**

bool m_leftDown

TRUE if the left mouse button is currently pressed down.

4.128.4 wxMouseEvent::m_middleDown**bool m_middleDown**

TRUE if the middle mouse button is currently pressed down.

4.128.5 wxMouseEvent::m_rightDown**bool m_rightDown**

TRUE if the right mouse button is currently pressed down.

4.128.6 wxMouseEvent::m_leftDown**bool m_leftDown**

TRUE if the left mouse button is currently pressed down.

4.128.7 wxMouseEvent::m_metaDown**bool m_metaDown**

TRUE if the Meta key is pressed down.

4.128.8 wxMouseEvent::m_shiftDown**bool m_shiftDown**

TRUE if shift is pressed down.

4.128.9 wxMouseEvent::m_x**float m_x**

X-coordinate of the event.

4.128.10 wxMouseEvent::m_y**float m_y**

Y-coordinate of the event.

4.128.11 **wxMouseEvent::wxMouseEvent**

wxMouseEvent(WXTYPE *mouseEventType* = 0, int *id* = 0)

Constructor. Valid event types are:

- **wxEVT_ENTER_WINDOW**
- **wxEVT_LEAVE_WINDOW**
- **wxEVT_LEFT_DOWN**
- **wxEVT_LEFT_UP**
- **wxEVT_LEFT_DCLICK**
- **wxEVT_MIDDLE_DOWN**
- **wxEVT_MIDDLE_UP**
- **wxEVT_MIDDLE_DCLICK**
- **wxEVT_RIGHT_DOWN**
- **wxEVT_RIGHT_UP**
- **wxEVT_RIGHT_DCLICK**
- **wxEVT_MOTION**

4.128.12 **wxMouseEvent::AltDown**

bool AltDown()

Returns TRUE if the Alt key was down at the time of the event.

4.128.13 **wxMouseEvent::Button**

bool Button(int *button*)

Returns TRUE if the identified mouse button is changing state. Valid values of *button* are 1, 2 or 3 for left, middle and right buttons respectively.

Not all mice have middle buttons so a portable application should avoid this one.

4.128.14 **wxMouseEvent::ButtonDClick**

bool ButtonDClick(int *but* = -1)

If the argument is omitted, this returns TRUE if the event was a mouse double click event. Otherwise the argument specifies which double click event was generated (1, 2 or 3 for left, middle and right buttons respectively).

4.128.15 wxMouseEvent::ButtonDown**bool ButtonDown**(int *but* = -1)

If the argument is omitted, this returns TRUE if the event was a mouse button down event. Otherwise the argument specifies which button-down event was generated (1, 2 or 3 for left, middle and right buttons respectively).

4.128.16 wxMouseEvent::ButtonUp**bool ButtonUp**(int *but* = -1)

If the argument is omitted, this returns TRUE if the event was a mouse button up event. Otherwise the argument specifies which button-up event was generated (1, 2 or 3 for left, middle and right buttons respectively).

4.128.17 wxMouseEvent::ControlDown**bool ControlDown**()

Returns TRUE if the control key was down at the time of the event.

4.128.18 wxMouseEvent::Dragging**bool Dragging**()

Returns TRUE if this was a dragging event (motion while a button is depressed).

4.128.19 wxMouseEvent::Entering**bool Entering**()

Returns TRUE if the mouse was entering the window (MS Windows and Motif).

See also *wxMouseEvent::Leaving* (p. 455).

4.128.20 wxMouseEvent::GetX**float GetX**()

Returns X coordinate of the mouse event position.

4.128.21 wxMouseEvent::GetY

float GetY()

Returns Y coordinate of the mouse event position.

4.128.22 wxMouseEvent::IsButton**bool IsButton()**

Returns TRUE if the event was a mouse button event (not necessarily a button down event - that may be tested using *ButtonDown*).

4.128.23 wxMouseEvent::Leaving**bool Leaving()**

Returns TRUE if the mouse was leaving the window (MS Windows and Motif).

See also *wxMouseEvent::Entering* (p. 454).

4.128.24 wxMouseEvent::LeftDClick**bool LeftDClick()**

Returns TRUE if the event was a left double click.

4.128.25 wxMouseEvent::LeftDown**bool LeftDown()**

Returns TRUE if the left mouse button changed to down.

4.128.26 wxMouseEvent::LeftIsDown**bool LeftIsDown()**

Returns TRUE if the left mouse button is currently down, independent of the current event type.

4.128.27 wxMouseEvent::LeftUp**bool LeftUp()**

Returns TRUE if the left mouse button changed to up.

4.128.28 wxMouseEvent::MetaDown**bool MetaDown()**

Returns TRUE if the Meta key was down at the time of the event.

4.128.29 wxMouseEvent::MiddleDClick**bool MiddleDClick()**

Returns TRUE if the event was a middle double click.

4.128.30 wxMouseEvent::MiddleDown**bool MiddleDown()**

Returns TRUE if the middle mouse button changed to down.

4.128.31 wxMouseEvent::MiddleIsDown**bool MiddleIsDown()**

Returns TRUE if the middle mouse button is currently down, independent of the current event type.

4.128.32 wxMouseEvent::MiddleUp**bool MiddleUp()**

Returns TRUE if the middle mouse button changed to up.

4.128.33 wxMouseEvent::Moving**bool Moving()**

Returns TRUE if this was a motion event (no buttons depressed).

4.128.34 wxMouseEvent::Position**void Position(float *x, float *y)**

Sets *x and *y to the position at which the event occurred. If the window is a window, the position is converted to logical units (according to the current mapping mode) with

scrolling taken into account. To get back to device units (for example to calculate where on the screen to place a dialog box associated with a window mouse event), use **wxDC::LogicalToDeviceX** and **wxDC::LogicalToDeviceY**.

For example, the following code calculates screen pixel coordinates from the frame position, window view start (assuming the window is the only subwindow on the frame and therefore at the top left of it), and the logical event position. A menu is popped up at the position where the mouse click occurred. (Note that the application should also check that the dialog box will be visible on the screen, since the click could have occurred near the screen edge!)

```
float event_x, event_y;
event.Position(&event_x, &event_y);
frame->GetPosition(&x, &y);
window->ViewStart(&x1, &y1);
int mouse_x = (int)(window->GetDC()->LogicalToDeviceX(event_x + x -
x1);
int mouse_y = (int)(window->GetDC()->LogicalToDeviceY(event_y + y -
y1);

char *choice = wxGetSingleChoice("Menu", "Pick a node action",
                                no_choices, choices, frame, mouse_x,
mouse_y);
```

4.128.35 **wxMouseEvent::RightDClick**

bool RightDClick()

Returns TRUE if the event was a right double click.

4.128.36 **wxMouseEvent::RightDown**

bool RightDown()

Returns TRUE if the right mouse button changed to down.

4.128.37 **wxMouseEvent::RightIsDown**

bool RightIsDown()

Returns TRUE if the right mouse button is currently down, independent of the current event type.

4.128.38 **wxMouseEvent::RightUp**

bool RightUp()

Returns TRUE if the right mouse button changed to up.

4.128.39 wxMouseEvent::ShiftDown**bool ShiftDown()**

Returns TRUE if the shift key was down at the time of the event.

4.129 wxMoveEvent

A move event holds information about move change events.

Derived from

wxEvent (p. 221)

wxObject (p. 471)

Include files

<wx/event.h>

Event table macros

To process a move event, use this event handler macro to direct input to a member function that takes a *wxMoveEvent* argument.

EVT_MOVE(func)	Process a <i>wxEVT_MOVE</i> event, which is generated when a window is moved.
-----------------------	-------------------------------------------------------------------------------

See also

wxWindow::OnMove (p. 825), *wxPoint* (p. 503), *Event handling overview* (p. 939)

4.129.1 wxMoveEvent::wxMoveEvent**wxMoveEvent(const wxPoint& pt, int id = 0)**

Constructor.

4.129.2 wxMoveEvent::GetPosition**wxPoint GetPosition() const**

Returns the position of the window generating the move change event.

4.130 wxMultipleChoiceDialog

This class represents a dialog that shows a list of strings, and allows the user to select one or more.

NOTE: this class is not yet implemented.

Derived from

wxDialog (p. 178)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/choicdlg.h>

See also

wxMultipleChoiceDialog overview (p. 918)

4.131 wxMutex

A mutex object is a synchronization object whose state is set to signaled when it is not owned by any thread, and nonsignaled when it is owned. Its name comes from its usefulness in coordinating mutually-exclusive access to a shared resource. Only one thread at a time can own a mutex object.

For example, when several thread use the data stored in the linked list, modifications to the list should be only allowed to one thread at a time because during a new node addition the list integrity is temporarily broken (this is also called *program invariant*).

Example

```
// this variable has an "s_" prefix because it is static: seeing an
"s_" in
// a multithreaded program is in general a good sign that you should
use a
// mutex (or a critical section)
static wxMutex *s_mutexProtectingTheGlobalData;

// we store some numbers in this global array which is presumably
used by
// several threads simultaneously
wxArrayInt s_data;

void MyThread::AddNewNode(int num)
{
    // ensure that no other thread accesses the list
```

```
s_mutexProtectingTheGlobalList->Lock();

s_data.Add(num);

s_mutexProtectingTheGlobalList->Unlock();
}

// return TRUE the given number is greater than all array elements
bool MyThread::IsGreater(int num)
{
    // before using the list we must acquire the mutex
    wxMutexLocker lock(s_mutexProtectingTheGlobalData);

    size_t count = s_data.Count();
    for ( size_t n = 0; n < count; n++ )
    {
        if ( s_data[n] > num )
            return FALSE;
    }

    return TRUE;
}
```

Notice how `wxMutexLocker` was used in the second function to ensure that the mutex is unlocked in any case: whether the function returns `TRUE` or `FALSE` (because the destructor of the local object *lock* is always called). Using this class instead of directly using `wxMutex` is, in general safer and is even more so if your program uses C++ exceptions.

Derived from

None.

Include files

`<wx/thread.h>`

See also

wxThread (p. 736), *wxCondition* (p. 110), *wxMutexLocker* (p. 462), *wxCriticalSection* (p. 126)

4.131.1 wxMutex::wxMutex

wxMutex()

Default constructor.

4.131.2 wxMutex::~~wxMutex**~wxMutex()**

Destroys the wxMutex object.

4.131.3 wxMutex::IsLocked**bool IsLocked() const**

Returns TRUE if the mutex is locked, FALSE otherwise.

4.131.4 wxMutex::Lock**wxMutexError Lock()**

Locks the mutex object.

Return value

One of:

wxMUTEX_NO_ERROR
wxMUTEX_DEAD_LOCK
wxMUTEX_BUSY

There was no error.
A deadlock situation was detected.
The mutex is already locked by another thread.

4.131.5 wxMutex::TryLock**wxMutexError TryLock()**

Tries to lock the mutex object. If it can't, returns immediately with an error.

Return value

One of:

wxMUTEX_NO_ERROR
wxMUTEX_DEAD_LOCK
wxMUTEX_BUSY

There was no error.
A deadlock situation was detected.
The mutex is already locked by another thread.

4.131.6 wxMutex::Unlock**wxMutexError Unlock()**

Unlocks the mutex object.

Return value

One of:

wxMUTEX_NO_ERROR
wxMUTEX_DEAD_LOCK
wxMUTEX_BUSY
wxMUTEX_UNLOCKED

There was no error.
A deadlock situation was detected.
The mutex is already locked by another thread.
The calling thread tries to unlock an unlocked mutex.

4.132 wxMutexLocker

This is a small helper class to be used with *wxMutex* (p. 459) objects. A *wxMutexLocker* acquires a mutex lock in the constructor and releases (or unlocks) the mutex in the destructor making it much more difficult to forget to release a mutex (which, in general, will promptly lead to the serious problems). See *wxMutex* (p. 459) for an example of *wxMutexLocker* usage.

Derived from

None.

Include files

<wx/thread.h>

See also

wxMutex (p. 459), *wxCriticalSectionLocker* (p. 127)

4.132.1 wxMutexLocker::wxMutexLocker

wxMutexLocker(wxMutex *mutex)

Constructs a *wxMutexLocker* object associated with *mutex* which must be non NULL and locks it. Call *IsLocked* (p. 463) to check if the mutex was successfully locked.

4.132.2 wxMutexLocker::~~wxMutexLocker

~wxMutexLocker()

Destuctor releases the mutex if it was successfully acquired in the ctor.

4.132.3 **wxMutexLocker::IsOk**

bool IsOk() const

Returns TRUE if mutex was acquired in the constructor, FALSE otherwise.

4.133 **wxNodeBase**

A node structure used in linked lists (see *wxList* (p. 367)) and derived classes. You should never use *wxNodeBase* class directly because it works with untyped (void *) data and this is unsafe. Use *wxNode*-derived classes which are defined by `WX_DECLARE_LIST` and `WX_DEFINE_LIST` macros instead as described in *wxList* (p. 367) documentation (see example there). *wxNode* is defined for compatibility as *wxNodeBase* containing "wxObject *" pointer, but usage of this class is deprecated.

Derived from

None.

Include files

<wx/list.h>

See also

wxList (p. 367), *wxHashTable* (p. 310)

4.133.1 **wxNodeBase::GetData**

void * Data()

Retrieves the client data pointer associated with the node.

4.133.2 **wxNodeBase::GetNext**

wxNodeBase * Next()

Retrieves the next node (NULL if at end of list).

4.133.3 **wxNodeBase::Previous**

wxNodeBase * GetPrevious()

Retrieves the previous node (NULL if at start of list).

4.133.4 wxNodeBase::SetData

void SetData(void *data)

Sets the data associated with the node (usually the pointer will have been set when the node was created).

4.133.5 wxNodeBase::IndexOf

int IndexOf()

Returns the zero-based index of this node within the list. The return value will be NOT_FOUND if the node has not been added to a list yet.

4.134 wxNotebook

This class represents a notebook control, which manages multiple windows with associated tabs.

To use the class, create a wxNotebook object and call *AddPage* (p. 465) or *InsertPage* (p. 468), passing a window to be used as the page. Do not explicitly delete the window for a page that is currently managed by wxNotebook.

wxNotebookPage is a typedef for wxWindow.

Derived from

wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/notebook.h>

Event handling

To process input from a notebook control, use the following event handler macros to direct input to member functions that take a *wxNotebookEvent* (p. 470) argument.

EVT_NOTEBOOK_PAGE_CHANGED(id, func) The page selection was changed.
Processes a
wxEVT_COMMAND_NOTEBOOK_PAGE_CHANGED event.
EVT_NOTEBOOK_PAGE_CHANGING(id, func) The page selection is about to be changed. Processes a

`wxEVT_COMMAND_NOTEBOOK_PAGE_CHANGING` event.

See also

`wxNotebookEvent` (p. 470), `wxImageList` (p. 339), `wxTabCtrl` (p. 695)

4.134.1 `wxNotebook::wxNotebook`

`wxNotebook()`

Default constructor.

`wxNotebook(wxWindow* parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& size, long style = 0, const wxString& name = "notebook")`

Constructs a notebook control.

Parameters

parent

The parent window. Must be non-NULL.

id

The window identifier.

pos

The window position.

size

The window size.

style

The window style. Its value is a bit list of zero or more of `wxTC_MULTILINE`, `wxTC_RIGHTJUSTIFY`, `wxTC_FIXEDWIDTH` and `wxTC_OWNERDRAW`.

4.134.2 `wxNotebook::~~wxNotebook`

`~wxNotebook()`

Destroys the `wxNotebook` object.

4.134.3 `wxNotebook::AddPage`

bool AddPage(wxNotebookPage* page, const wxString& text, bool select = FALSE, int imageld = -1)

Adds a new page.

Parameters

page

Specifies the new page.

text

Specifies the text for the new page.

select

Specifies whether the page should be selected.

imageld

Specifies the optional image index for the new page.

Return value

TRUE if successful, FALSE otherwise.

Remarks

Do not delete the page, it will be deleted by the notebook.

See also

wxNotebook::InsertPage (p. 468)

4.134.4 wxNotebook::AdvanceSelection

void AdvanceSelection(bool forward = TRUE)

Cycles through the tabs.

4.134.5 wxNotebook::Create

bool Create(wxWindow* parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& size, long style = 0, const wxString& name = "notebook")

Creates a notebook control. See *wxNotebook::wxNotebook* (p. 465) for a description of the parameters.

4.134.6 wxNotebook::DeleteAllPages**bool DeleteAllPages()**

Deletes all pages.

4.134.7 wxNotebook::DeletePage**bool DeletePage(int *page*)**

Deletes the specified page, and the associated window.

4.134.8 wxNotebook::GetImageList**wxImageList* GetImageList() const**

Returns the associated image list.

[See also](#)

wxImageList (p. 339), *wxNotebook::SetImageList* (p. 469)

4.134.9 wxNotebook::GetPage**wxNotebookPage* GetPage(int *page*)**

Returns the window at the given page position.

4.134.10 wxNotebook::GetPageCount**int GetPageCount() const**

Returns the number of pages in the notebook control.

4.134.11 wxNotebook::GetPageImage**int GetPageImage() const**

Returns the image index for the given page.

4.134.12 wxNotebook::GetPageText**wxString GetPageText() const**

Returns the string for the given page.

4.134.13 wxNotebook::GetRowCount

int GetRowCount() const

Returns the number of rows in the notebook control.

4.134.14 wxNotebook::GetSelection

int GetSelection() const

Returns the currently selected page, or -1 if none was selected.

4.134.15 wxNotebook::InsertPage

**bool InsertPage(int index, wxNotebookPage* page, const wxString& text, bool
select = FALSE, int imageld = -1)**

Inserts a new page at the specified position.

Parameters

index

Specifies the position for the new page.

page

Specifies the new page.

text

Specifies the text for the new page.

select

Specifies whether the page should be selected.

imageld

Specifies the optional image index for the new page.

Return value

TRUE if successful, FALSE otherwise.

Remarks

Do not delete the page, it will be deleted by the notebook.

See also

wxNotebook::AddPage (p. 465)

4.134.16 wxNotebook::OnSelChange

void OnSelChange(wxNotebookEvent& *event*)

An event handler function, called when the page selection is changed.

[See also](#)

wxNotebookEvent (p. 470)

4.134.17 wxNotebook::RemovePage

bool RemovePage(int *page*)

Deletes the specified page, without deleting the associated window.

4.134.18 wxNotebook::SetImageList

void SetImageList(wxImageList* *imageList*)

Sets the image list for the page control.

[See also](#)

wxImageList (p. 339)

4.134.19 wxNotebook::SetPadding

void SetPadding(const wxSize& *padding*)

Sets the amount of space around each page's icon and label, in pixels.

4.134.20 wxNotebook::SetPageSize

void SetPageSize(const wxSize& *size*)

Sets the width and height of the pages.

4.134.21 wxNotebook::SetPageImage

bool SetPageImage(int *page*, int *image*)

Sets the image index for the given page. *image* is an index into the image list which was set with *wxNotebook::SetImageList* (p. 469).

4.134.22 **wxNotebook::SetPageText**

bool SetPageText(int *page*, const wxString& *text*)

Sets the text for the given page.

4.134.23 **wxNotebook::SetSelection**

int SetSelection(int *page*)

Sets the selection for the given page, returning the previous selection.

See also

wxNotebook::GetSelection (p. 468)

4.135 **wxNotebookEvent**

This class represents the events generated by a notebook control.

Derived from

wxCommandEvent (p. 103)

wxEvent (p. 221)

wxEvtHandler (p. 224)

wxObject (p. 471)

Include files

<wx/notebook.h>

Event table macros

To process a notebook event, use these event handler macros to direct input to member functions that take a *wxNotebookEvent* argument.

EVT_NOTEBOOK_PAGE_CHANGED(*id*, *func*) The page selection was changed.
Processes a
wxEVT_COMMAND_NOTEBOOK_PAGE_CHANGED event.

EVT_NOTEBOOK_PAGE_CHANGING(*id*, *func*) The page selection is about to be changed. Processes a
wxEVT_COMMAND_NOTEBOOK_PAGE_CHANGING event.

See also

wxNotebook (p. 464), *wxTabCtrl* (p. 695), *wxTabEvent* (p. 700)

4.135.1 wxNotebookEvent::wxNotebookEvent

wxNotebookEvent(**wxEventType** *eventType* = *wxEVT_NULL*, **int** *id* = 0, **int** *sel* = -1, **int** *oldSel* = -1)

Constructor.

4.135.2 wxNotebookEvent::GetOldSelection

int **GetOldSelection**() **const**

Returns the page that was selected before the change, -1 if none was selected.

4.135.3 wxNotebookEvent::GetSelection

int **GetSelection**() **const**

Returns the currently selected page, or -1 if none was selected.

4.135.4 wxNotebookEvent::SetOldSelection

void **SetOldSelection**(**int** *page*)

Sets the id of the page selected before the change.

4.135.5 wxNotebookEvent::SetSelection

void **SetSelection**(**int** *page*)

Sets the selection member variable.

See also

wxNotebookEvent::GetSelection (p. 471)

4.136 wxObject

This is the root class of all wxWindows classes. It declares a virtual destructor which ensures that destructors get called for all derived class objects where necessary.

wxObject is the hub of a dynamic object creation scheme, enabling a program to create instances of a class only knowing its string class name, and to query the class hierarchy.

The class contains optional debugging versions of **new** and **delete**, which can help trace memory allocation and deallocation problems.

wxObject can be used to implement reference counted objects, such as wxPen, wxBitmap and others.

See also

wxClassInfo (p. 79), *Debugging overview* (p. 927), *wxObjectRefData* (p. 475)

4.136.1 wxObject::wxObject

wxObject()

Default constructor.

4.136.2 wxObject::~~wxObject

wxObject()

Destructor. Performs dereferencing, for those objects that use reference counting.

4.136.3 wxObject::m_refData

wxObjectRefData* m_refData

Pointer to an object which is the object's reference-counted data.

See also

wxObject::Ref (p. 474), *wxObject::UnRef* (p. 474), *wxObject::SetRefData* (p. 474), *wxObject::GetRefData* (p. 473), *wxObjectRefData* (p. 475)

4.136.4 wxObject::Dump

void Dump(ostream& stream)

A virtual function that should be redefined by derived classes to allow dumping of

memory states.

Parameters

stream

Stream on which to output dump information.

Remarks

Currently wxWindows does not define Dump for derived classes, but programmers may wish to use it for their own applications. Be sure to call the Dump member of the class's base class to allow all information to be dumped.

The implementation of this function just writes the class name of the object. If DEBUG is undefined or zero, the implementation is empty.

4.136.5 **wxObject::GetClassInfo**

wxClassInfo * GetClassInfo()

This virtual function is redefined for every class that requires run-time type information, when using DECLARE_CLASS macros.

4.136.6 **wxObject::GetRefData**

wxObjectRefData* GetRefData() const

Returns the **m_refData** pointer.

See also

wxObject::Ref (p. 474), *wxObject::UnRef* (p. 474), *wxObject::m_refData* (p. 472), *wxObject::SetRefData* (p. 474), *wxObjectRefData* (p. 475)

4.136.7 **wxObject::IsKindOf**

bool IsKindOf(wxClassInfo *info)

Determines whether this class is a subclass of (or the same class as) the given class.

Parameters

info

A pointer to a class information object, which may be obtained by using the CLASSINFO macro.

Return value

TRUE if the class represented by *info* is the same class as this one or is derived from it.

Example

```
bool tmp = obj->IsKindOf(CLASSINFO(wxFrame));
```

4.136.8 wxObject::Ref

void Ref(const wxObject& clone)

Makes this object refer to the data in *clone*.

Parameters

clone

The object to 'clone'.

Remarks

First this function calls *wxObject::UnRef* (p. 474) on itself to decrement (and perhaps free) the data it is currently referring to.

It then sets its own *m_refData* to point to that of *clone*, and increments the reference count inside the data.

See also

wxObject::UnRef (p. 474), *wxObject::m_refData* (p. 472), *wxObject::SetRefData* (p. 474), *wxObject::GetRefData* (p. 473), *wxObjectRefData* (p. 475)

4.136.9 wxObject::SetRefData

void SetRefData(wxObjectRefData* data)

Sets the *m_refData* pointer.

See also

wxObject::Ref (p. 474), *wxObject::UnRef* (p. 474), *wxObject::m_refData* (p. 472), *wxObject::GetRefData* (p. 473), *wxObjectRefData* (p. 475)

4.136.10 wxObject::UnRef

void UnRef()

Decrements the reference count in the associated data, and if it is zero, deletes the data. The *m_refData* member is set to NULL.

See also

wxObject::Ref (p. 474), *wxObject::m_refData* (p. 472), *wxObject::SetRefData* (p. 474), *wxObject::GetRefData* (p. 473), *wxObjectRefData* (p. 475)

4.136.11 **wxObject::operator new**

void * new(size_t *size*, const wxString& *filename* = NULL, int *lineNum* = 0)

The *new* operator is defined for debugging versions of the library only, when the identifier `DEBUG` is defined and is more than zero. It takes over memory allocation, allowing `wxDebugContext` operations.

4.136.12 **wxObject::operator delete**

void delete(void *buf*)

The *delete* operator is defined for debugging versions of the library only, when the identifier `DEBUG` is defined and is more than zero. It takes over memory deallocation, allowing `wxDebugContext` operations.

4.137 **wxObjectRefData**

This class is used to store reference-counted data. Derive classes from this to store your own data. When retrieving information from a **wxObject**'s reference data, you will need to cast to your own derived class.

Friends

wxObject (p. 471)

See also

wxObject (p. 471)

4.137.1 **wxObjectRefData::m_count**

int m_count

Reference count. When this goes to zero during a *wxObject::UnRef* (p. 474), an object can delete the **wxObjectRefData** object.

4.137.2 **wxObjectRefData::wxObjectRefData**

wxObjectRefData()

Default constructor. Initialises the **m_count** member to 1.

4.137.3 wxObjectRefData::~~wxObjectRefData**wxObjectRefData()**

Destructor.

4.138 wxOutputStream**Derived from**

wxStreamBase (p. 646)

Include files

<wx/stream.h>

See also

wxStreamBuffer (p. 648)

4.138.1 wxOutputStream::wxOutputStream**wxOutputStream()**

Creates a dummy *wxOutputStream* object.

wxOutputStream(wxStreamBuffer* sbuf)

Creates an input stream using the specified stream buffer *sbuf*. This stream buffer can point to another stream.

4.138.2 wxOutputStream::~~wxOutputStream**~wxOutputStream()**

Destructor.

4.138.3 wxOutputStream::OutputStreamBuffer**wxStreamBuffer * OutputStreamBuffer()**

Returns the stream buffer associated with the output stream.

4.138.4 wxOutputStream::LastWrite

size_t LastWrite() const

4.138.5 wxOutputStream::PutC

void PutC(char c)

Puts the specified character in the output queue and increments the stream position.

4.138.6 wxOutputStream::SeekO

off_t SeekO(off_t pos, wxSeekMode mode)

Changes the stream current position.

4.138.7 wxOutputStream::TellO

off_t TellO() const

Returns the current stream position.

4.138.8 wxOutputStream::Write

wxOutputStream& Write(const void *buffer, size_t size)

Writes the specified amount of bytes using the data of *buffer*. *WARNING!* The buffer absolutely needs to have at least the specified size.

This function returns a reference on the current object, so the user can test any states of the stream right away.

wxOutputStream& Write(wxInputStream& stream_in)

Reads data from the specified input stream and stores them in the current stream. The data is read until an error is raised by one of the two streams.

4.139 wxPageSetupData

This class holds a variety of information related to *wxPageSetupDialog* (p. 482).

Derived from

wxObject (p. 471)

Include files

<wx/cmndata.h>

See also

wxPageSetupDialog (p. 482)

4.139.1 wxPageSetupData::wxPageSetupData

wxPageSetupData()

Constructor.

4.139.2 wxPageSetupData::~wxPageSetupData

~wxPageSetupData()

Destructor.

4.139.3 wxPageSetupData::EnableHelp

void EnableHelp(bool *flag*)

Enables or disables the 'Help' button (Windows only).

4.139.4 wxPageSetupData::EnableMargins

void EnableMargins(bool *flag*)

Enables or disables the margin controls (Windows only).

4.139.5 wxPageSetupData::EnableOrientation

void EnableOrientation(bool *flag*)

Enables or disables the orientation control (Windows only).

4.139.6 wxPageSetupData::EnablePaper

void EnablePaper(bool *flag*)

Enables or disables the paper size control (Windows only).

4.139.7 wxPageSetupData::EnablePrinter

void EnablePrinter(bool *flag*)

Enables or disables the **Printer** button, which invokes a printer setup dialog.

4.139.8 wxPageSetupData::GetPaperSize

wxPoint GetPaperSize()

Returns the paper size in millimetres.

4.139.9 wxPageSetupData::GetMarginTopLeft

wxPoint GetMarginTopLeft()

Returns the left (x) and top (y) margins.

4.139.10 wxPageSetupData::GetMarginBottomRight

wxPoint GetMarginBottomRight()

Returns the right (x) and bottom (y) margins.

4.139.11 wxPageSetupData::GetMinMarginTopLeft

wxPoint GetMinMarginTopLeft()

Returns the left (x) and top (y) minimum margins the user can enter (Windows only).

4.139.12 wxPageSetupData::GetMinMarginBottomRight

wxPoint GetMinMarginBottomRight()

Returns the right (x) and bottom (y) minimum margins the user can enter (Windows only).

4.139.13 wxPageSetupData::GetOrientation

int GetOrientation()

Returns the orientation, which can be wxPORTRAIT or wxLANDSCAPE.

4.139.14 wxPageSetupData::GetDefaultMinMargins**bool GetDefaultMinMargins()**

Returns TRUE if the page setup dialog will take its minimum margin values from the currently selected printer properties. Windows only.

4.139.15 wxPageSetupData::GetEnableMargins**bool GetEnableMargins()**

Returns TRUE if the margin controls are enabled (Windows only).

4.139.16 wxPageSetupData::GetEnableOrientation**bool GetEnableOrientation()**

Returns TRUE if the orientation control is enabled (Windows only).

4.139.17 wxPageSetupData::GetEnablePaper**bool GetEnablePaper()**

Returns TRUE if the paper size control is enabled (Windows only).

4.139.18 wxPageSetupData::GetEnablePrinter**bool GetEnablePrinter()**

Returns TRUE if the printer setup button is enabled.

4.139.19 wxPageSetupData::GetEnableHelp**bool GetEnableHelp()**

Returns TRUE if the printer setup button is enabled.

4.139.20 wxPageSetupData::GetDefaultInfo

bool GetDefaultInfo()

Returns TRUE if the dialog will simply return default printer information (such as orientation) instead of showing a dialog. Windows only.

4.139.21 wxPageSetupData::SetPaperSize**void SetPaperSize(const wxPoint& size)**

Sets the paper size in millimetres.

4.139.22 wxPageSetupData::SetMarginTopLeft**void GetMarginTopLeft(const wxPoint& pt)**

Sets the left (x) and top (y) margins.

4.139.23 wxPageSetupData::SetMarginBottomRight**void SetMarginBottomRight(const wxPoint& pt)**

Sets the right (x) and bottom (y) margins.

4.139.24 wxPageSetupData::SetMinMarginTopLeft**void SetMinMarginTopLeft(const wxPoint& pt)**

Sets the left (x) and top (y) minimum margins the user can enter (Windows only).

4.139.25 wxPageSetupData::SetMinMarginBottomRight**void SetMinMarginBottomRight(const wxPoint& pt)**

Sets the right (x) and bottom (y) minimum margins the user can enter (Windows only).

4.139.26 wxPageSetupData::SetOrientation**void SetOrientation(int orientation)**

Sets the orientation, which can be wxPORTRAIT or wxLANDSCAPE.

4.139.27 wxPageSetupData::SetDefaultMinMargins

void SetDefaultMinMargins(bool flag)

Pass TRUE if the page setup dialog will take its minimum margin values from the currently selected printer properties. Windows only.

4.139.28 wxPageSetupData::SetDefaultInfo**void SetDefaultInfo(bool flag)**

Pass TRUE if the dialog will simply return default printer information (such as orientation) instead of showing a dialog. Windows only.

4.140 wxPageSetupDialog

This class represents the page setup common dialog. The page setup dialog is standard from Windows 95 on, replacing the print setup dialog (which is retained in Windows and wxWindows for backward compatibility). On Windows 95 and NT 4.0 and above, the page setup dialog is native to the windowing system, otherwise it is emulated.

The page setup dialog contains controls for paper size (A4, A5 etc.), orientation (landscape or portrait), and controls for setting left, top, right and bottom margin sizes in millimetres. The page setup dialog does not set any global information (the exception being orientation for PostScript printing) so you need to query the *wxPageSetupData* (p. 477) object associated with the dialog.

Note that the OK and Cancel buttons do not destroy the dialog; this must be done by the application.

Derived from

wxDialog (p. 178)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/printdlg.h>

See also

wxPrintDialog (p. 512), *wxPageSetupData* (p. 477)

4.140.1 wxPageSetupDialog::wxPageSetupDialog

wxPageSetupDialog(wxWindow* parent, wxPageSetupData* data = NULL)

Constructor. Pass a parent window, and optionally a pointer to a block of page setup data, which will be copied to the print dialog's internal data.

4.140.2 **wxPageSetupDialog::~wxPageSetupDialog**

~wxPageSetupDialog()

Destructor.

4.140.3 **wxPageSetupDialog::GetPageSetupData**

wxPageSetupData& GetPageSetupData()

Returns the *page setup data* (p. 477) associated with the dialog.

4.140.4 **wxPageSetupDialog::ShowModal**

int ShowModal()

Shows the dialog, returning `wxID_OK` if the user pressed OK, and `wxID_CANCEL` otherwise.

4.141 **wxPaintDC**

A `wxPaintDC` must be constructed if an application wishes to paint on the client area of a window from within an **OnPaint** event. This should normally be constructed as a temporary stack object; don't store a `wxPaintDC` object. If you have an `OnPaint` handler, you *must* create a `wxPaintDC` object within it even if you don't actually use it.

Using `wxPaintDC` within `OnPaint` is important because it automatically sets the clipping area to the damaged area of the window. Attempts to draw outside this area do not appear.

To draw on a window from outside **OnPaint**, construct a `wxClientDC` (p. 81) object.

To draw on the whole window including decorations, construct a `wxWindowDC` (p. 843) object (Windows only).

Derived from

`wxDC` (p. 151)

Include files

`<wx/dcclient.h>`

See also

wxDC (p. 151), *wxMemoryDC* (p. 415), *wxPaintDC* (p. 483), *wxWindowDC* (p. 843), *wxScreenDC* (p. 578)

4.141.1 wxPaintDC::wxPaintDC

wxPaintDC(*wxWindow** window)

Constructor. Pass a pointer to the window on which you wish to paint.

4.142 wxPaintEvent

A paint event is sent when a window's contents needs to be repainted.

Derived from

wxEvent (p. 221)
wxObject (p. 471)

Include files

<wx/event.h>

Event table macros

To process a paint event, use this event handler macro to direct input to a member function that takes a *wxPaintEvent* argument.

EVT_PAINT(func) Process a *wxEVT_PAINT* event.

See also

wxWindow::OnPaint (p. 825), *Event handling overview* (p. 939)

4.142.1 wxPaintEvent::wxPaintEvent

wxPaintEvent(int id = 0)

Constructor.

4.143 wxPalette

A palette is a table that maps pixel values to RGB colours. It allows the colours of a low-depth bitmap, for example, to be mapped to the available colours in a display.

Derived from

wxGDIObject (p. 295)

wxObject (p. 471)

Include files

<wx/palette.h>

Predefined objects

Objects:

wxNullPalette

See also

wxDC::SetPalette (p. 163), *wxBitmap* (p. 39)

4.143.1 **wxPalette::wxPalette**

wxPalette()

Default constructor.

wxPalette(const wxPalette& *palette*)

Copy constructor. This uses reference counting so is a cheap operation.

**wxPalette(int *n*, const unsigned char* *red*,
const unsigned char* *green*, const unsigned char* *blue*)**

Creates a palette from arrays of size *n*, one for each red, blue or green component.

Parameters

palette

A pointer or reference to the palette to copy.

n

The number of indices in the palette.

red

An array of red values.

green

An array of green values.

blue

An array of blue values.

See also

wxPalette::Create (p. 486)

4.143.2 **wxPalette::~~wxPalette**

~wxPalette()

Destructor.

4.143.3 **wxPalette::Create**

bool Create(int *n*, const unsigned char* *red*, const unsigned char* *green*, const unsigned char* *blue*)

Creates a palette from arrays of size *n*, one for each red, blue or green component.

Parameters

n

The number of indices in the palette.

red

An array of red values.

green

An array of green values.

blue

An array of blue values.

Return value

TRUE if the creation was successful, FALSE otherwise.

See also

wxPalette::wxPalette (p. 485)

4.143.4 **wxPalette::GetPixel**

int GetPixel(const unsigned char *red*, const unsigned char *green*, const unsigned char *blue*) const

Returns a pixel value (index into the palette) for the given RGB values.

Parameters

red
Red value.

green
Green value.

blue
Blue value.

Return value

The nearest palette index.

See also

wxPalette::GetRGB (p. 487)

4.143.5 wxPalette::GetRGB

bool GetPixel(int *pixel*, const unsigned char* *red*, const unsigned char* *green*, const unsigned char* *blue*) const

Returns RGB values for a given palette index.

Parameters

pixel
The palette index.

red
Receives the red value.

green
Receives the green value.

blue
Receives the blue value.

Return value

TRUE if the operation was successful.

See also

wxPalette::GetPixel (p. 486)

4.143.6 **wxPalette::Ok**

bool Ok() const

Returns TRUE if palette data is present.

4.143.7 **wxPalette::operator =**

wxPalette& operator =(const wxPalette& palette)

Assignment operator, using reference counting. Returns a reference to 'this'.

4.143.8 **wxPalette::operator ==**

bool operator ==(const wxPalette& palette)

Equality operator. Two palettes are equal if they contain pointers to the same underlying palette data. It does not compare each attribute, so two independently-created palettes using the same parameters will fail the test.

4.143.9 **wxPalette::operator !=**

bool operator !=(const wxPalette& palette)

Inequality operator. Two palettes are not equal if they contain pointers to different underlying palette data. It does not compare each attribute.

4.144 **wxPanel**

A panel is a window on which controls are placed. It is usually placed within a frame. It contains minimal extra functionality over and above its parent class `wxWindow`; its main purpose is to be similar in appearance and functionality to a dialog, but with the flexibility of having any window as a parent.

Note: if not all characters are being intercepted by your `OnKeyDown` or `OnChar` handler, it may be because you are using the `wxTAB_TRAVERSAL` style, which grabs some keypresses for use by child controls.

Derived from

wxWindow (p. 798)

wxEvtHandler (p. 224)

wxObject (p. 471)

Include files

<wx/panel.h>

Window styles

There are no specific styles for this window.

See also *window styles overview* (p. 959).

Remarks

By default, a panel has the same colouring as a dialog.

A panel may be loaded from a wxWindows resource file (extension `wxr`).

See also

wxDialog (p. 178)

4.144.1 wxPanel::wxPanel

wxPanel()

Default constructor.

wxPanel(wxWindow* *parent*, wxWindowID *id*, const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size* = wxDefaultSize, long *style* = wxTAB_TRAVERSAL, const wxString& *name* = "panel")

Constructor.

Parameters

parent

The parent window.

id

An identifier for the panel. A value of -1 is taken to mean a default.

pos

The panel position. A value of (-1, -1) indicates a default position, chosen by either the windowing system or wxWindows, depending on platform.

size

The panel size. A value of (-1, -1) indicates a default size, chosen by either the windowing system or wxWindows, depending on platform.

style

The window style. See *wxPanel* (p. 488).

name

Used to associate a name with the window, allowing the application user to set Motif resource values for individual dialog boxes.

[See also](#)

wxPanel::Create (p. 490)

4.144.2 **wxPanel::~~wxPanel**

~wxPanel()

Destructor. Deletes any child windows before deleting the physical window.

4.144.3 **wxPanel::Create**

bool Create(*wxWindow* parent*, *wxWindowID id*, **const wxPoint& pos** = *wxDefaultPosition*, **const wxSize& size** = *wxDefaultSize*, **long style** = *wxTAB_TRAVERSAL*, **const wxString& name** = "panel")

Used for two-step panel construction. See *wxPanel::wxPanel* (p. 489) for details.

4.144.4 **wxPanel::InitDialog**

void InitDialog()

Sends an *wxWindow::OnInitDialog* (p. 823) event, which in turn transfers data to the dialog via validators.

[See also](#)

wxWindow::OnInitDialog (p. 823)

4.144.5 **wxPanel::OnSysColourChanged**

void OnSysColourChanged(*wxSysColourChangedEvent& event*)

The default handler for *wxEVT_SYS_COLOUR_CHANGED*.

[Parameters](#)

event

The colour change event.

Remarks

Changes the panel's colour to conform to the current settings (Windows only). Add an event table entry for your panel class if you wish the behaviour to be different (such as keeping a user-defined background colour). If you do override this function, call *wxWindow::OnSysColourChanged* (p. 828) to propagate the notification to child windows and controls.

See also

wxSysColourChangedEvent (p. 678)

4.145 wxPanelTabView

The *wxPanelTabView* is responsible for input and output on a *wxPanel*.

Derived from

wxTabView (p. 687)

wxObject (p. 471)

Include files

<wx/tab.h>

See also

wxTabView overview (p. 963), *wxTabView* (p. 687)

4.145.1 wxPanelTabView::wxPanelTabView

void wxPanelTabView(wxPanel *panel, long style = wxTAB_STYLE_DRAW_BOX | wxTAB_STYLE_COLOUR_INTERIOR)

Constructor. *panel* should be a *wxTabbedPanel* or *wxTabbedDialog*: the type will be checked by the view at run time.

style may be a bit list of the following:

wxTAB_STYLE_DRAW_BOX	Draw a box around the view area. Most commonly used for dialogs.
wxTAB_STYLE_COLOUR_INTERIOR	Draw tab backgrounds in the specified colour. Omitting this style will ensure that the tab background matches the dialog background.

4.145.2 wxPanelTabView::~~wxPanelTabView**void ~wxPanelTabView()**

Destructor. This destructor deletes all the panels associated with the view. If you do not wish this to happen, call `ClearWindows` with argument `FALSE` before the view is likely to be destroyed. This will clear the list of windows, without deleting them.

4.145.3 wxPanelTabView::AddTabWindow**void AddTabPanel(int id, wxWindow *window)**

Adds a window to the view. The window is associated with the tab identifier, and will be shown or hidden as the tab is selected or deselected.

4.145.4 wxPanelTabView::ClearWindows**void ClearWindows(bool deleteWindows = TRUE)**

Removes the child windows from the view. If *deleteWindows* is `TRUE`, the windows will be deleted.

4.145.5 wxPanelTabView::GetCurrentWindow**wxPanel * GetCurrentWindow()**

Returns the child window currently being displayed on the tabbed panel or dialog box.

4.145.6 wxPanelTabView::GetTabWindow**wxWindow * GetTabWindow(int id)**

Returns the window associated with the tab identifier.

4.145.7 wxPanelTabView::ShowWindowForTab**void ShowWindowForTab(int id)**

Shows the child window corresponding to the tab identifier, and hides the previously shown window.

4.146 wxPathList

The path list is a convenient way of storing a number of directories, and when presented

with a filename without a directory, searching for an existing file in those directories. Storing the filename only in an application's files and using a locally-defined list of directories makes the application and its files more portable.

Use the *wxFileNameFromPath* global function to extract the filename from the path.

Derived from

wxList (p. 367)

wxObject (p. 471)

Include files

<wx/filefn.h>

See also

wxList (p. 367)

4.146.1 **wxPathList::wxPathList**

wxPathList()

Constructor.

4.146.2 **wxPathList::AddEnvList**

void AddEnvList(const wxString& *env_variable*)

Finds the value of the given environment variable, and adds all paths to the path list. Useful for finding files in the PATH variable, for example.

4.146.3 **wxPathList::Add**

void Add(const wxString& *path*)

Adds the given directory to the path list, but does not check if the path was already on the list (use *wxPathList::Member* for this).

4.146.4 **wxPathList::EnsureFileAccessible**

void EnsureFileAccessible(const wxString& *filename*)

Given a full filename (with path), ensures that files in the same path can be accessed

using the pathlist. It does this by stripping the filename and adding the path to the list if not already there.

4.146.5 **wxPathList::FindAbsolutePath**

wxString FindAbsolutePath(const wxString& file)

Searches for a full path for an existing file by appending *file* to successive members of the path list. If the file exists, a temporary pointer to the absolute path is returned.

4.146.6 **wxPathList::FindValidPath**

wxString FindValidPath(const wxString& file)

Searches for a full path for an existing file by appending *file* to successive members of the path list. If the file exists, a temporary pointer to the full path is returned. This path may be relative to the current working directory.

4.146.7 **wxPathList::Member**

bool Member(const wxString& file)

TRUE if the path is in the path list (ignoring case).

4.147 **wxPen**

A pen is a drawing tool for drawing outlines. It is used for drawing lines and painting the outline of rectangles, ellipses, etc. It has a colour, a width and a style.

Derived from

wxGDIObject (p. 295)

wxObject (p. 471)

Include files

<wx/pen.h>

Predefined objects

Objects:

wxNullPen

Pointers:

wxRED_PEN

wxCYAN_PEN
wxGREEN_PEN
wxBLACK_PEN
wxWHITE_PEN
wxTRANSPARENT_PEN
wxBLACK_DASHED_PEN
wxGREY_PEN
wxMEDIUM_GREY_PEN
wxLIGHT_GREY_PEN

Remarks

On a monochrome display, wxWindows shows all non-white pens as black.

Do not initialize objects on the stack before the program commences, since other required structures may not have been set up yet. Instead, define global pointers to objects and create them in *OnInit* or when required.

An application may wish to dynamically create pens with different characteristics, and there is the consequent danger that a large number of duplicate pens will be created. Therefore an application may wish to get a pointer to a pen by using the global list of pens **wxThePenList**, and calling the member function **FindOrCreatePen**. See the entry for *wxPenList* (p. 501).

wxPen uses a reference counting system, so assignments between brushes are very cheap. You can therefore use actual wxPen objects instead of pointers without efficiency problems. Once one wxPen object changes its data it will create its own pen data internally so that other pens, which previously shared the data using the reference counting, are not affected.

See also

wxPenList (p. 501), *wxDC* (p. 151), *wxDC::SetPen* (p. 165)

4.147.1 wxPen::wxPen

wxPen()

Default constructor. The pen will be uninitialised, and *wxPen::Ok* (p. 499) will return FALSE.

wxPen(const wxColour& colour, int width, int style)

Constructs a pen from a colour object, pen width and style.

wxPen(const wxString& colourName, int width, int style)

Constructs a pen from a colour name, pen width and style.

wxPen(const wxBitmap& *stipple*, int *width*)

Constructs a stippled pen from a stipple bitmap and a width.

wxPen(const wxPen& *pen*)

Copy constructor. This uses reference counting so is a cheap operation.

Parameters

colour

A colour object.

colourName

A colour name.

width

Pen width. Under Windows, the pen width cannot be greater than 1 if the style is wxDOT, wxLONG_DASH, wxSHORT_DASH, wxDOT_DASH, or wxUSER_DASH.

stipple

A stipple bitmap.

pen

A pointer or reference to a pen to copy.

width

Pen width. Under Windows, the pen width cannot be greater than 1 if the style is wxDOT, wxLONG_DASH, wxSHORT_DASH, wxDOT_DASH, or wxUSER_DASH.

style

The style may be one of the following:

wxSOLID

Solid style.

wxTRANSPARENT

No pen is used.

wxDOT

Dotted style.

wxLONG_DASH

Long dashed style.

wxSHORT_DASH

Short dashed style.

wxDOT_DASH

Dot and dash style.

wxSTIPPLE

Use the stipple bitmap.

wxUSER_DASH

Use the user dashes: see *wxPen::SetDashes* (p. 499).

wxBDIAGONAL_HATCH

Backward diagonal hatch.

wxCROSSDIAG_HATCH

Cross-diagonal hatch.

wxFDIAGONAL_HATCH

Forward diagonal hatch.

wxCROSS_HATCH

Cross hatch.

wxHORIZONTAL_HATCH

Horizontal hatch.

wxVERTICAL_HATCH

Vertical hatch.

Remarks

If the named colour form is used, an appropriate **wxColour** structure is found in the colour database.

style may be one of **wxSOLID**, **wxDOT**, **wxLONG_DASH**, **wxSHORT_DASH** and **wxDOT_DASH**.

See also

wxPen::SetStyle (p. 500), *wxPen::SetColour* (p. 499), *wxPen::SetWidth* (p. 500), *wxPen::SetStipple* (p. 500)

4.147.2 wxPen::~~wxPen

~wxPen()

Destructor.

Remarks

The destructor may not delete the underlying pen object of the native windowing system, since **wxBrush** uses a reference counting system for efficiency.

Although all remaining pens are deleted when the application exits, the application should try to clean up all pens itself. This is because **wxWindows** cannot know if a pointer to the pen object is stored in an application data structure, and there is a risk of double deletion.

4.147.3 wxPen::GetCap

int GetCap() const

Returns the pen cap style, which may be one of **wxCAP_ROUND**, **wxCAP_PROJECTING** and **wxCAP_BUTT**. The default is **wxCAP_ROUND**.

See also

wxPen::SetCap (p. 499)

4.147.4 wxPen::GetColour

wxColour& GetColour() const

Returns a reference to the pen colour.

See also

wxPen::SetColour (p. 499)

4.147.5 **wxPen::GetDashes**

int GetDashes(wxDash dashes) const**

Gets an array of dashes (defined as char in X, DWORD under Windows). *dashes* is a pointer to the internal array. Do not deallocate or store this pointer. The function returns the number of dashes associated with this pen.

[See also](#)

wxPen::SetDashes (p. 499)

4.147.6 **wxPen::GetJoin**

int GetJoin() const

Returns the pen join style, which may be one of **wxJOIN_BEVEL**, **wxJOIN_ROUND** and **wxJOIN_MITER**. The default is **wxJOIN_ROUND**.

[See also](#)

wxPen::SetJoin (p. 500)

4.147.7 **wxPen::GetStipple**

wxBitmap* GetStipple() const

Gets a pointer to the stipple bitmap.

[See also](#)

wxPen::SetStipple (p. 500)

4.147.8 **wxPen::GetStyle**

int GetStyle() const

Returns the pen style.

[See also](#)

wxPen::wxPen (p. 495), *wxPen::SetStyle* (p. 500)

4.147.9 wxPen::GetWidth**int GetWidth() const**

Returns the pen width.

[See also](#)

wxPen::SetWidth (p. 500)

4.147.10 wxPen::Ok**bool Ok() const**

Returns TRUE if the pen is initialised.

4.147.11 wxPen::SetCap**void SetCap(int capStyle)**

Sets the pen cap style, which may be one of **wxCAP_ROUND**, **wxCAP_PROJECTING** and **wxCAP_BUTT**. The default is **wxCAP_ROUND**.

[See also](#)

wxPen::GetCap (p. 497)

4.147.12 wxPen::SetColour**void SetColour(wxColour& colour)****void SetColour(const wxString& colourName)****void SetColour(int red, int green, int blue)**

The pen's colour is changed to the given colour.

[See also](#)

wxPen::GetColour (p. 497)

4.147.13 wxPen::SetDashes**void SetDashes(int n, wxDash* dashes)**

Associates an array of pointers to dashes (defined as char in X, DWORD under

Windows) with the pen. The array is not deallocated by `wxPen`, but neither must it be deallocated by the calling application until the pen is deleted or this function is called with a `NULL` array.

[See also](#)

`wxPen::GetDashes` (p. 498)

4.147.14 `wxPen::SetJoin`

`void SetJoin(int join_style)`

Sets the pen join style, which may be one of `wxJOIN_BEVEL`, `wxJOIN_ROUND` and `wxJOIN_MITER`. The default is `wxJOIN_ROUND`.

[See also](#)

`wxPen::GetJoin` (p. 498)

4.147.15 `wxPen::SetStipple`

`void SetStipple(wxBitmap* stipple)`

Sets the bitmap for stippling.

[See also](#)

`wxPen::GetStipple` (p. 498)

4.147.16 `wxPen::SetStyle`

`void SetStyle(int style)`

Set the pen style.

[See also](#)

`wxPen::wxPen` (p. 495)

4.147.17 `wxPen::SetWidth`

`void SetWidth(int width)`

Sets the pen width.

[See also](#)

wxPen::GetWidth (p. 499)

4.147.18 wxPen::operator =

wxPen& operator =(const wxPen& pen)

Assignment operator, using reference counting. Returns a reference to 'this'.

4.147.19 wxPen::operator ==

bool operator ==(const wxPen& pen)

Equality operator. Two pens are equal if they contain pointers to the same underlying pen data. It does not compare each attribute, so two independently-created pens using the same parameters will fail the test.

4.147.20 wxPen::operator !=

bool operator !=(const wxPen& pen)

Inequality operator. Two pens are not equal if they contain pointers to different underlying pen data. It does not compare each attribute.

4.148 wxPenList

There is only one instance of this class: **wxThePenList**. Use this object to search for a previously created pen of the desired type and create it if not already found. In some windowing systems, the pen may be a scarce resource, so it can pay to reuse old resources if possible. When an application finishes, all pens will be deleted and their resources freed, eliminating the possibility of 'memory leaks'. However, it is best not to rely on this automatic cleanup because it can lead to double deletion in some circumstances.

There are two mechanisms in recent versions of wxWindows which make the pen list less useful than it once was. Under Windows, scarce resources are cleaned up internally if they are not being used. Also, a referencing counting mechanism applied to all GDI objects means that some sharing of underlying resources is possible. You don't have to keep track of pointers, working out when it is safe delete a pen, because the referencing counting does it for you. For example, you can set a pen in a device context, and then immediately delete the pen you passed, because the pen is 'copied'.

So you may find it easier to ignore the pen list, and instead create and copy pens as you see fit. If your Windows resource meter suggests your application is using too many resources, you can resort to using GDI lists to share objects explicitly.

The only compelling use for the pen list is for wxWindows to keep track of pens in order

to clean them up on exit. It is also kept for backward compatibility with earlier versions of `wxWindows`.

See also

`wxPen` (p. 494)

4.148.1 `wxPenList::wxPenList`

`void wxPenList()`

Constructor. The application should not construct its own pen list: use the object pointer **`wxThePenList`**.

4.148.2 `wxPenList::AddPen`

`void AddPen(wxPen* pen)`

Used internally by `wxWindows` to add a pen to the list.

4.148.3 `wxPenList::FindOrCreatePen`

`wxPen* FindOrCreatePen(const wxColour& colour, int width, int style)`

Finds a pen with the specified attributes and returns it, else creates a new pen, adds it to the pen list, and returns it.

`wxPen* FindOrCreatePen(const wxString& colourName, int width, int style)`

Finds a pen with the specified attributes and returns it, else creates a new pen, adds it to the pen list, and returns it.

Parameters

colour
Colour object.

colourName
Colour name, which should be in the *colour database* (p. 91).

width
Width of pen.

style
Pen style. See `wxPen::wxPen` (p. 495) for a list of styles.

4.148.4 wxPenList::RemovePen**void RemovePen**(wxPen* *pen*)

Used by wxWindows to remove a pen from the list.

4.149 wxPoint

A **wxPoint** is a useful data structure for graphics operations. It simply contains integer *x* and *y* members.

See also *wxRealPoint* (p. 546) for a floating point version.

Derived from

None

Include files

<wx/gdicmn.h>

See also

wxRealPoint (p. 546)

4.149.1 wxPoint::wxPoint**wxPoint**()**wxPoint**(int *x*, int *y*)

Create a point.

4.149.2 wxPoint::x**int** *x*

x member.

4.149.3 wxPoint::y**int** *y*

y member.

4.150 wxPostScriptDC

This defines the wxWindows Encapsulated PostScript device context, which can write PostScript files on any platform. See *wxDC* (p. 151) for descriptions of the member functions.

Derived from

wxDC (p. 151)
wXObject (p. 471)

Include files

<wx/dcps.h>

4.150.1 wxPostScriptDC::wxPostScriptDC

wxPostScriptDC(const **wxString&** *output*, **bool** *interactive* = *TRUE*,
wxWindow **parent*)

Constructor. *output* is an optional file for printing to, and if *interactive* is *TRUE* a dialog box will be displayed for adjusting various parameters. *parent* is the parent of the printer dialog box.

Use the *Ok* member to test whether the constructor was successful in creating a useable device context.

See *Printer settings* (p. 857) for functions to set and get PostScript printing settings.

4.150.2 wxPostScriptDC::GetStream

ostream * **GetStream**()

Returns the stream currently being used to write PostScript output. Use this to insert any PostScript code that is outside the scope of *wxPostScriptDC*.

4.151 wxPreviewCanvas

A preview canvas is the default canvas used by the print preview system to display the preview.

Derived from

wxScrolledWindow (p. 586)
wxWindow (p. 798)
wxevthandler (p. 224)

wxObject (p. 471)

Include files

<wx/print.h>

See also

wxPreviewFrame (p. 507), *wxPreviewControlBar* (p. 505), *wxPrintPreview* (p. 519)

4.151.1 **wxPreviewCanvas::wxPreviewCanvas**

wxPreviewCanvas(*wxPrintPreview** preview, *wxWindow** parent, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = 0, **const wxString&** name = "canvas")

Constructor.

4.151.2 **wxPreviewCanvas::~~wxPreviewCanvas**

~wxPreviewCanvas()

Destructor.

4.151.3 **wxPreviewCanvas::OnPaint**

void OnPaint(*wxPaintEvent&* event)

Calls *wxPrintPreview::PaintPage* (p. 522) to refresh the canvas.

4.152 **wxPreviewControlBar**

This is the default implementation of the preview control bar, a panel with buttons and a zoom control. You can derive a new class from this and override some or all member functions to change the behaviour and appearance; or you can leave it as it is.

Derived from

wxPanel (p. 488)

wxWindow (p. 798)

wxEvtHandler (p. 224)

wxObject (p. 471)

Include files

<wx/print.h>

[See also](#)

wxPreviewFrame (p. 507), *wxPreviewCanvas* (p. 504), *wxPrintPreview* (p. 519)

4.152.1 **wxPreviewControlBar::wxPreviewControlbar**

wxPreviewControlBar(*wxPrintPreview** preview, **long** buttons, **wxWindow*** parent, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = 0, **const wxString&** name = "panel")

Constructor.

The buttons parameter may be a combination of the following, using the bitwise 'or' operator.

<code>wxPREVIEW_PRINT</code>	Create a print button.
<code>wxPREVIEW_NEXT</code>	Create a next page button.
<code>wxPREVIEW_PREVIOUS</code>	Create a previous page button.
<code>wxPREVIEW_ZOOM</code>	Create a zoom control.
<code>wxPREVIEW_DEFAULT</code>	Equivalent to a combination of <code>wxPREVIEW_PREVIOUS</code> , <code>wxPREVIEW_NEXT</code> and <code>wxPREVIEW_ZOOM</code> .

4.152.2 **wxPreviewControlBar::~~wxPreviewControlBar**

~wxPreviewControlBar()

Destructor.

4.152.3 **wxPreviewControlBar::CreateButtons**

void CreateButtons()

Creates buttons, according to value of the button style flags.

4.152.4 **wxPreviewControlBar::GetPrintPreview**

wxPrintPreview * GetPrintPreview()

Gets the print preview object associated with the control bar.

4.152.5 wxPreviewControlBar::GetZoomControl**int GetZoomControl()**

Gets the current zoom setting in percent.

4.152.6 wxPreviewControlBar::SetZoomControl**void SetZoomControl(int percent)**

Sets the zoom control.

4.153 wxPreviewFrame

This class provides the default method of managing the print preview interface. Member functions may be overridden to replace functionality, or the class may be used without derivation.

Derived from

wxFrame (p. 275)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/print.h>

See also

wxPreviewCanvas (p. 504), *wxPreviewControlBar* (p. 505), *wxPrintPreview* (p. 519)

4.153.1 wxPreviewFrame::wxPreviewFrame

wxPreviewFrame(wxPrintPreview* preview, wxFrame* parent, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT_FRAME_STYLE, const wxString& name = "frame")

Constructor. Pass a print preview object plus other normal frame arguments.

4.153.2 wxPreviewFrame::~~wxPreviewFrame**~wxPreviewFrame()**

Destructor.

4.153.3 wxPreviewFrame::CreateControlBar

void CreateControlBar()

Creates a wxPreviewControlBar. Override this function to allow a user-defined preview control bar object to be created.

4.153.4 wxPreviewFrame::CreateCanvas

void CreateCanvas()

Creates a wxPreviewCanvas. Override this function to allow a user-defined preview canvas object to be created.

4.153.5 wxPreviewFrame::Initialize

void Initialize()

Creates the preview canvas and control bar, and calls wxWindow::MakeModal(TRUE) to disable other top-level windows in the application.

This function should be called by the application prior to showing the frame.

4.153.6 wxPreviewFrame::OnCloseWindow

void OnCloseWindow(wxCloseEvent& event)

Enables the other frames in the application, and deletes the print preview object, implicitly deleting any printout objects associated with the print preview object.

4.154 wxPrintData

This class holds a variety of information related to print dialogs.

Derived from

wxObject (p. 471)

Include files

<wx/cmndata.h>

See also

wxPrintDialog (p. 512), *wxPrintDialog Overview* (p. 917)

4.154.1 wxPrintData::wxPrintData

wxPrintData()

Constructor.

4.154.2 wxPrintData::~~wxPrintData

~wxPrintData()

Destructor.

4.154.3 wxPrintData::EnableHelp

void EnableHelp(bool *flag*)

Enables or disables the 'Help' button.

4.154.4 wxPrintData::EnablePageNumbers

void EnablePageNumbers(bool *flag*)

Enables or disables the 'Page numbers' controls.

4.154.5 wxPrintData::EnablePrintToFile

void EnablePrintToFile(bool *flag*)

Enables or disables the 'Print to file' checkbox.

4.154.6 wxPrintData::EnableSelection

void EnableSelection(bool *flag*)

Enables or disables the 'Selection' radio button.

4.154.7 wxPrintData::GetAllPages

bool GetAllPages()

Returns TRUE if the user requested that all pages be printed.

4.154.8 wxPrintData::GetCollate**bool GetCollate()**

Returns TRUE if the user requested that the document(s) be collated.

4.154.9 wxPrintData::GetFromPage**int GetFromPage()**

Returns the *from* page number, as entered by the user.

4.154.10 wxPrintData::GetMaxPage**int GetMaxPage()**

Returns the *maximum* page number.

4.154.11 wxPrintData::GetMinPage**int GetMinPage()**

Returns the *minimum* page number.

4.154.12 wxPrintData::GetNoCopies**int GetNoCopies()**

Returns the number of copies requested by the user.

4.154.13 wxPrintData::GetOrientation**int GetOrientation()**

Gets the orientation. This can be wxLANDSCAPE or wxPORTRAIT.

4.154.14 wxPrintData::GetToPage**int GetToPage()**

Returns the *to* page number, as entered by the user.

4.154.15 wxPrintData::SetCollate

void SetCollate(bool *flag*)

Sets the 'Collate' checkbox to TRUE or FALSE.

4.154.16 wxPrintData::SetFromPage

void SetFromPage(int *page*)

Sets the *from* page number.

4.154.17 wxPrintData::SetMaxPage

void SetMaxPage(int *page*)

Sets the *maximum* page number.

4.154.18 wxPrintData::SetMinPage

void SetMinPage(int *page*)

Sets the *minimum* page number.

4.154.19 wxPrintData::SetOrientation

void SetOrientation(int *orientation*)

Sets the orientation. This can be wxLANDSCAPE or wxPORTRAIT.

4.154.20 wxPrintData::SetNoCopies

void SetNoCopies(int *n*)

Sets the default number of copies to be printed out.

4.154.21 wxPrintData::SetPrintToFile

void SetPrintToFile(bool *flag*)

Sets the 'Print to file' checkbox to TRUE or FALSE.

4.154.22 **wxPrintData::SetSetupDialog**

void SetSetupDialog(bool *flag*)

Determines whether the dialog to be shown will be the Print dialog (pass FALSE) or Print Setup dialog (pass TRUE).

Note that the setup dialog is obsolete from Windows 95, though retained for backward compatibility.

4.154.23 **wxPrintData::SetToPage**

void SetToPage(int *page*)

Sets the *to* page number.

4.155 **wxPrintDialog**

This class represents the print and print setup common dialogs. You may obtain a *wxPrinterDC* (p. 515) device context from a successfully dismissed print dialog.

Derived from

wxDialog (p. 178)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/printdlg.h>

See also

wxPrintDialog Overview (p. 917)

4.155.1 **wxPrintDialog::wxPrintDialog**

wxPrintDialog(wxWindow* *parent*, wxPrintData* *data* = NULL)

Constructor. Pass a parent window, and optionally a pointer to a block of print data, which will be copied to the print dialog's print data.

See also

wxPrintData (p. 508)

4.155.2 wxPrintDialog::~wxPrintDialog**~wxPrintDialog()**

Destructor. If *wxPrintDialog::GetPrintDC* has *not* been called, the device context obtained by the dialog (if any) will be deleted.

4.155.3 wxPrintDialog::GetPrintData**wxPrintData& GetPrintData()**

Returns the *print data* (p. 508) associated with the print dialog.

4.155.4 wxPrintDialog::GetPrintDC**wxDC* GetPrintDC()**

Returns the device context created by the print dialog, if any. When this function has been called, the ownership of the device context is transferred to the application, so it must then be deleted explicitly.

4.155.5 wxPrintDialog::ShowModal**int ShowModal()**

Shows the dialog, returning *wxID_OK* if the user pressed OK, and *wxID_CANCEL* otherwise. After this function is called, a device context may be retrievable using *wxPrintDialog::GetPrintDC* (p. 513).

4.156 wxPrinter

This class represents the Windows or PostScript printer, and is the vehicle through which printing may be launched by an application. Printing can also be achieved through using of lower functions and classes, but this and associated classes provide a more convenient and general method of printing.

Derived from

wxObject (p. 471)

Include files

<wx/print.h>

See also

Printing framework overview (p. 950), *wxPrinterDC* (p. 515), *wxPrintDialog* (p. 512), *wxPrintout* (p. 516), *wxPrintPreview* (p. 519).

4.156.1 wxPrinter::wxPrinter

wxPrinter(wxPrintData* data = NULL)

Constructor. Pass an optional pointer to a block of print data, which will be copied to the printer object's print data.

See also

wxPrintData (p. 508)

4.156.2 wxPrinter::~~wxPrinter

~wxPrinter()

Destructor.

4.156.3 wxPrinter::Abort

bool Abort()

Returns TRUE if the user has aborted the print job.

4.156.4 wxPrinter::CreateAbortWindow

void CreateAbortWindow(wxWindow* parent, wxPrintout* printout)

Creates the default printing abort window, with a cancel button.

4.156.5 wxPrinter::GetPrintData

wxPrintData& GetPrintData()

Returns the *print data* (p. 508) associated with the printer object.

4.156.6 wxPrinter::Print**bool Print**(wxWindow *parent, wxPrintout *printout, bool prompt=TRUE)

Starts the printing process. Provide a parent window, a user-defined wxPrintout object which controls the printing of a document, and whether the print dialog should be invoked first.

Print could return FALSE if there was a problem initializing the printer device context (current printer not set, for example).

4.156.7 wxPrinter::PrintDialog**bool PrintDialog**(wxWindow *parent)

Invokes the print dialog.

4.156.8 wxPrinter::ReportError**void ReportError**(wxWindow *parent, wxPrintout *printout, const wxString& message)

Default error-reporting function.

4.156.9 wxPrinter::Setup**void Setup**(wxWindow *parent)

Invokes the print setup dialog. Note that the setup dialog is obsolete from Windows 95, though retained for backward compatibility.

4.157 wxPrinterDC

A printer device context is specific to Windows, and allows access to any printer with a Windows driver. See wxDC (p. 151) for further information on device contexts, and wxDC::GetSize (p. 160) for advice on achieving the correct scaling for the page.

Derived from

wxDC (p. 151)

wxObject (p. 151)

Include files

<wx/dcprint.h>

See also

wxDC (p. 151), *Printing framework overview* (p. 950)

4.157.1 **wxPrinterDC::wxPrinterDC**

wxPrinterDC(const wxString& *driver*, const wxString& *device*, const wxString& *output*, const bool *interactive* = TRUE, int *orientation* = wxPORTRAIT)

Constructor. With empty strings for the first three arguments, the default printer dialog is displayed. *device* indicates the type of printer and *output* is an optional file for printing to. The *driver* parameter is currently unused. Use the *Ok* member to test whether the constructor was successful in creating a useable device context.

4.158 **wxPrintout**

This class encapsulates the functionality of printing out an application document. A new class must be derived and members overridden to respond to calls such as *OnPrintPage* and *HasPage*. Instances of this class are passed to *wxPrinter::Print* or a *wxPrintPreview* object to initiate printing or previewing.

Derived from

wxObject (p. 471)

Include files

<wx/print.h>

See also

Printing framework overview (p. 950), *wxPrinterDC* (p. 515), *wxPrintDialog* (p. 512), *wxPrinter* (p. 513), *wxPrintPreview* (p. 519)

4.158.1 **wxPrintout::wxPrintout**

wxPrintout(const wxString& *title* = "Printout")

Constructor. Pass an optional title argument (currently unused).

4.158.2 **wxPrintout::~~wxPrintout**

~wxPrintout()

Destructor.

4.158.3 **wxPrintout::GetDC**

wxDC * GetDC()

Returns the device context associated with the printout (given to the printout at start of printing or previewing). This will be a wxPrinterDC if printing under Windows, a wxPostScriptDC if printing on other platforms, and a wxMemoryDC if previewing.

4.158.4 **wxPrintout::GetPageInfo**

void GetPageInfo(int *minPage, int *maxPage, int *pageFrom, int *pageTo)

Called by the framework to obtain information from the application about minimum and maximum page values that the user can select, and the required page range to be printed. By default this returns 1, 32000 for the page minimum and maximum values, and 1, 1 for the required page range.

If *minPage* is zero, the page number controls in the print dialog will be disabled.

4.158.5 **wxPrintout::GetPageSizeMM**

void GetPageSizeMM(int *w, int *h)

Returns the size of the printer page in millimetres.

4.158.6 **wxPrintout::GetPageSizePixels**

void GetPageSizePixels(int *w, int *h)

Returns the size of the printer page in pixels. These may not be the same as the values returned from *wxDC::GetSize* (p. 160) if the printout is being used for previewing, since in this case, a memory device context is used, using a bitmap size reflecting the current preview zoom. The application must take this discrepancy into account if previewing is to be supported.

4.158.7 **wxPrintout::GetPPIPrinter**

void GetPPIPrinter(int *w, int *h)

Returns the number of pixels per logical inch of the printer device context. Dividing the printer PPI by the screen PPI can give a suitable scaling factor for drawing text onto the printer. Remember to multiply this by a scaling factor to take the preview DC size into account.

4.158.8 wxPrintout::GetPPIScreen**void GetPPIScreen(int *w, int *h)**

Returns the number of pixels per logical inch of the screen device context. Dividing the printer PPI by the screen PPI can give a suitable scaling factor for drawing text onto the printer. Remember to multiply this by a scaling factor to take the preview DC size into account.

4.158.9 wxPrintout::HasPage**bool HasPage(int pageNum)**

Should be overridden to return TRUE if the document has this page, or FALSE if not. Returning FALSE signifies the end of the document. By default, HasPage behaves as if the document has only one page.

4.158.10 wxPrintout::IsPreview**bool IsPreview()**

Returns TRUE if the printout is currently being used for previewing.

4.158.11 wxPrintout::OnBeginDocument**bool OnBeginDocument(int startPage, int endPage)**

Called by the framework at the start of document printing. Return FALSE from this function cancels the print job. OnBeginDocument is called once for every copy printed.

The base `wxPrintout::OnBeginDocument` *must* be called (and the return value checked) from within the overridden function, since it calls `wxDC::StartDoc`.

4.158.12 wxPrintout::OnEndDocument**void OnEndDocument()**

Called by the framework at the end of document printing. OnEndDocument is called once for every copy printed.

The base `wxPrintout::OnEndDocument` *must* be called from within the overridden function, since it calls `wxDC::EndDoc`.

4.158.13 wxPrintout::OnBeginPrinting**void OnBeginPrinting()**

Called by the framework at the start of printing. OnBeginPrinting is called once for every print job (regardless of how many copies are being printed).

4.158.14 wxPrintout::OnEndPrinting**void OnEndPrinting()**

Called by the framework at the end of printing. OnEndPrinting is called once for every print job (regardless of how many copies are being printed).

4.158.15 wxPrintout::OnPreparePrinting**void OnPreparePrinting()**

Called once by the framework before any other demands are made of the wxPrintout object. This gives the object an opportunity to calculate the number of pages in the document, for example.

4.158.16 wxPrintout::OnPrintPage**bool OnPrintPage(int pageNum)**

Called by the framework when a page should be printed. Returning FALSE cancels the print job. The application can use wxPrintout::GetDC to obtain a device context to draw on.

4.159 wxPrintPreview

Printing framework overview (p. 950)

Objects of this class manage the print preview process. The object is passed a wxPrintout object, and the wxPrintPreview object itself is passed to a wxPreviewFrame object. Previewing is started by initializing and showing the preview frame. Unlike wxPrinter::Print, flow of control returns to the application immediately after the frame is shown.

Derived from

wxObject (p. 471)

Include files

<wx/print.h>

See also

Printing framework overview (p. 950), *wxPrinterDC* (p. 515), *wxPrintDialog* (p. 512), *wxPrintout* (p. 516), *wxPrinter* (p. 513), *wxPreviewCanvas* (p. 504), *wxPreviewControlBar* (p. 505), *wxPreviewFrame* (p. 507).

4.159.1 **wxPrintPreview::wxPrintPreview**

wxPrintPreview(*wxPrintout** printout, *wxPrintout** printoutForPrinting, *wxPrintData** data=NULL)

Constructor. Pass a printout object, an optional printout object to be used for actual printing, and the address of an optional block of printer data, which will be copied to the print preview object's print data.

If *printoutForPrinting* is non-NULL, a **Print...** button will be placed on the preview frame so that the user can print directly from the preview interface.

Do not explicitly delete the printout objects once this destructor has been called, since they will be deleted in the *wxPrintPreview* constructor. The same does not apply to the *data* argument.

Test the *Ok* member to check whether the *wxPrintPreview* object was created correctly. *Ok* could return *FALSE* if there was a problem initializing the printer device context (current printer not set, for example).

4.159.2 **wxPrintPreview::~~wxPrintPreview**

~wxPrinter()

Destructor. Deletes both print preview objects, so do not destroy these objects in your application.

4.159.3 **wxPrintPreview::DrawBlankPage**

bool DrawBlankPage(*wxWindow** window)

Draws a representation of the blank page into the preview window. Used internally.

4.159.4 **wxPrintPreview::GetCanvas**

*wxWindow** **GetCanvas**()

Gets the preview window used for displaying the print preview image.

4.159.5 wxPrintPreview::GetCurrentPage

int GetCurrentPage()

Gets the page currently being previewed.

4.159.6 wxPrintPreview::GetFrame

wxFrame * GetFrame()

Gets the frame used for displaying the print preview canvas and control bar.

4.159.7 wxPrintPreview::GetMaxPage

int GetMaxPage()

Returns the maximum page number.

4.159.8 wxPrintPreview::GetMinPage

int GetMinPage()

Returns the minimum page number.

4.159.9 wxPrintPreview::GetPrintData

wxPrintData& GetPrintData()

Returns a reference to the internal print data.

4.159.10 wxPrintPreview::GetPrintout

wxPrintout * GetPrintout()

Gets the preview printout object associated with the wxPrintPreview object.

4.159.11 wxPrintPreview::GetPrintoutForPrinting

wxPrintout * GetPrintoutForPrinting()

Gets the printout object to be used for printing from within the preview interface, or NULL

if none exists.

4.159.12 wxPrintPreview::Ok

bool Ok()

Returns TRUE if the wxPrintPreview is valid, FALSE otherwise. It could return FALSE if there was a problem initializing the printer device context (current printer not set, for example).

4.159.13 wxPrintPreview::PaintPage

bool PaintPage(wxWindow* window)

This refreshes the preview window with the preview image. It must be called from the preview window's OnPaint member.

The implementation simply blits the preview bitmap onto the canvas, creating a new preview bitmap if none exists.

4.159.14 wxPrintPreview::Print

bool Print(bool prompt)

Invokes the print process using the second wxPrintout object supplied in the wxPrintPreview constructor. Will normally be called by the **Print...** panel item on the preview frame's control bar.

4.159.15 wxPrintPreview::RenderPage

bool RenderPage(int pageNum)

Renders a page into a wxMemoryDC. Used internally by wxPrintPreview.

4.159.16 wxPrintPreview::SetCanvas

void SetCanvas(wxWindow* window)

Sets the window to be used for displaying the print preview image.

4.159.17 wxPrintPreview::SetCurrentPage

void SetCurrentPage(int pageNum)

Sets the current page to be previewed.

4.159.18 wxPrintPreview::SetFrame

void SetFrame(wxFrame **frame*)

Sets the frame to be used for displaying the print preview canvas and control bar.

4.159.19 wxPrintPreview::SetPrintout

void SetPrintout(wxPrintout **printout*)

Associates a printout object with the wxPrintPreview object.

4.159.20 wxPrintPreview::SetZoom

void SetZoom(int *percent*)

Sets the percentage preview zoom, and refreshes the preview canvas accordingly.

4.160 wxPrivateDataObject

wxPrivateDataObject is a specialization of wxDataObject for application-specific or standard format data. The format of the data contained in an instance of this class must be identified with a string literal corresponding to the mime-type of the data. Typically this would be "image/png" or "text/html" or "application/word".

Derived from

wxDataObject (p. 138)

Include files

<wx/dataobj.h>

See also

wxDataObject (p. 138)

4.160.1 wxPrivateDataObject::wxPrivateDataObject

wxPrivateDataObject()

4.160.2 wxPrivateDataObject::~~wxPrivateDataObject**~wxPrivateDataObject()****4.160.3 wxPrivateDataObject::SetId****virtual void SetId(const wxString& id)**

The string ID identifies the format of clipboard or DnD data. A word processor would e.g. add a wxTextDataObject and a wxPrivateDataObject to the clipboard - the latter with the Id "application/word".

4.160.4 wxPrivateDataObject::GetId**virtual wxString GetId() const**

Returns the ID of the clipboard or DnD data format.

4.160.5 wxPrivateDataObject::SetData**virtual void SetData(const char *data, size_t size)**

Set the data. The data object will make an internal copy.

4.160.6 wxPrivateDataObject::GetSize**virtual size_t GetDataSize() const**

Returns the data size.

4.160.7 wxPrivateDataObject::GetData**virtual char* GetData()**

Returns a pointer to the data.

4.160.8 wxPrivateDataObject::WriteData**virtual void WriteData(void*dest) const**

Write the data owned by this class to *dest*. By default, this calls *WriteData* (p. 525) with data set using *SetData* (p. 524). This can be overridden to provide data on-demand; in this case *WriteData(data,dest)* (p. 525) (see below) must be called from within the overriding *WriteData()* method.

4.160.9 wxPrivateDataObject::WriteData

void WriteData(const char* data, void*dest) const

Writes the data *data* to *dest*. This method must be called from *WriteData* (p. 524).

4.161 wxPrivateDropTarget

wxPrivateDropTarget is for...

Derived from

wxDropTarget (p. 218)

Include files

<wx/dnd.h>

See also

wxDropTarget (p. 218)

4.161.1 wxPrivateDropTarget::wxPrivateDropTarget

wxPrivateDropTarget()

4.161.2 wxPrivateDropTarget::SetId

void SetId(const wxString& id)

You have to override *OnDrop* to get at the data. The string ID identifies the format of clipboard or DnD data. A word processor would e.g. add a *wxTextDataObject* and a *wxPrivateDataObject* to the clipboard - the latter with the Id "WXWORD_FORMAT".

4.161.3 wxPrivateDropTarget::GetId

virtual wxString GetId() const

4.162 wxProcess

The objects of this class are used in conjunction with *wxExecute* (p. 865) function. When a *wxProcess* object is passed to *wxExecute()*, its *OnTerminate()* (p. 527) virtual method is called when the process terminates. This allows the program to be (asynchronously)

notified about the process termination and also retrieve its exit status which is unavailable from `wxExecute()` in the case of asynchronous execution.

Please note that if the process termination notification is processed by the parent, it is responsible for deleting the `wxProcess` object which sent it. However, if it is not processed, the object will delete itself and so the library users should only delete those objects whose notifications have been processed (and call *Detach()* (p. 526) for others).

Derived from

`wxEvtHandler` (p. 224)

Include files

`<wx/process.h>`

4.162.1 `wxProcess::wxProcess`

`wxProcess(wxEvtHandler * parent = NULL, int id = -1)`

Constructs a process object. *id* is only used in the case you want to use `wxWindows` events. It identifies this object, or another window that will receive the event.

If the *parent* parameter is different from `NULL`, it will receive a `wxEVT_END_PROCESS` notification event (you should insert `EVT_END_PROCESS` macro in the event table of the parent to handle it) with the given *id*.

Parameters

parent

The event handler parent.

id

id of an event.

4.162.2 `wxProcess::~~wxProcess`

`~wxProcess()`

Destroys the `wxProcess` object.

4.162.3 `wxProcess::Detach`

`void Detach()`

Normally, a `wxProcess` object is deleted by its parent when it receives the notification about the process termination. However, it might happen that the parent object is destroyed before the external process is terminated (e.g. a window from which this external process was launched is closed by the user) and in this case it **should not delete** the `wxProcess` object, but **should call `Detach()`** instead. After the `wxProcess` object is detached from its parent, no notification events will be sent to the parent and the object will delete itself upon reception of the process termination notification.

4.162.4 `wxProcess::OnTerminate`

`void OnTerminate(int pid, int status) const`

It is called when the process with the pid `pid` finishes. It raises a `wxWindows` event when it isn't overridden.

pid

The pid of the process which has just terminated.

status

The exit code of the process.

4.163 `wxProcessEvent`

A process event is sent when a process is terminated.

Derived from

wxEvent (p. 221)

wxObject (p. 471)

Include files

<wx/process.h>

Event table macros

To process a `wxProcessEvent`, use these event handler macros to direct input to a member function that takes a `wxProcessEvent` argument.

`EVT_END_PROCESS(id, func)`

Process a `wxEVT_END_PROCESS` event. *id* is the identifier of the process object (the id passed to the `wxProcess` constructor) or a window to receive the event.

See also

wxProcess (p. 525), *Event handling overview* (p. 939)

4.163.1 wxProcessEvent::wxProcessEvent**wxProcessEvent**(int *id* = 0, int *pid* = 0)

Constructor. Takes a wxProcessObject or window id, and a process id.

4.163.2 wxProcessEvent::m_pid**int m_pid**

Contains the process id.

4.163.3 wxProcessEvent::GetPid**int GetPid()** const

Returns the process id.

4.163.4 wxProcessEvent::SetPid**void SetPid**(int *pid*)

Sets the process id.

4.164 wxProtocol**Derived from**

wxSocketClient (p. 615)

Include files

<wx/protocol/protocol.h>

See also

wxSocketBase (p. 607), *wxURL* (p. 779)

4.164.1 wxProtocol::Reconnect**bool Reconnect()**

Tries to reestablish a previous opened connection (close and renegotiate connection).

Return value

TRUE, if the connection is established, else FALSE.

4.164.2 **wxProtocol::GetInputStream**

wxInputStream * GetInputStream(const wxString& path)

Creates a new input stream on the the specified path. You can use all but seek functionality of wxStream. Seek isn't available on all stream. For example, http or ftp streams doesn't deal with it. Other functions like StreamSize and Tell aren't available for the moment for this sort of stream. You will be notified when the EOF is reached by an error.

Return value

Returns the initialized stream. You will have to delete it yourself once you don't use it anymore. The destructor closes the network connection.

See also

wxInputStream (p. 346)

4.164.3 **wxProtocol::Abort**

bool Abort()

Abort the current stream.

Warning

It is advised to destroy the input stream instead of aborting the stream this way.

Return value

Returns TRUE, if successful, else FALSE.

4.164.4 **wxProtocol::GetError**

wxProtocolError GetError()

Returns the last occurred error.

wxPROTO_NOERR
wxPROTO_NETERR

No error.
A generic network error occurred.

wxPROTO_PROTERR	An error occurred during negotiation.
wxPROTO_CONNERR	The client failed to connect the server.
wxPROTO_INVVAL	Invalid value.
wxPROTO_NOHNDLR	.
wxPROTO_NOFILE	The remote file doesn't exist.
wxPROTO_ABRT	Last action aborted.
wxPROTO_RCNCT	An error occurred during reconnection.
wxPROTO_STREAM	Someone tried to send a command during a transfer.

4.164.5 **wxProtocol::GetContentType**

wxString GetContentType()

Returns the type of the content of the last opened stream. It is a mime-type.

4.164.6 **wxProtocol::SetUser**

void SetUser(const wxString& user)

Sets the authentication user. It is mainly useful when FTP is used.

4.164.7 **wxProtocol::SetPassword**

void SetPassword(const wxString& user)

Sets the authentication password. It is mainly useful when FTP is used.

4.165 **wxQueryCol**

Every ODBC data column is represented by an instance of this class.

Derived from

wxObject (p. 471)

Include files

<wx/odbc.h>

See also

wxQueryCol overview (p. 923), *wxDatabase* overview (p. 923)

4.165.1 wxQueryCol::wxQueryCol**void wxQueryCol()**

Constructor. Sets the attributes of the column to default values.

4.165.2 wxQueryCol::~~wxQueryCol**void ~wxQueryCol()**

Destructor. Deletes the wxQueryField list.

4.165.3 wxQueryCol::BindVar**void * BindVar(void *v, long sz)**

Binds a user-defined variable to a column. Whenever a column is bound to a variable, it will automatically copy the data of the current field into this buffer (to a maximum of sz bytes).

4.165.4 wxQueryCol::FillVar**void FillVar(int recnum)**

Fills the bound variable with the data of the field recnum. When no variable is bound to the column nothing will happen.

4.165.5 wxQueryCol::GetData**void * GetData(int field)**

Returns a pointer to the data of the field.

4.165.6 wxQueryCol::GetName**wxString GetName()**

Returns the name of a column.

4.165.7 wxQueryCol::GetType**short GetType()**

Returns the data type of a column.

4.165.8 wxQueryCol::GetSize**long GetSize(int *field*)**

Return the size of the data of the field *field*.

4.165.9 wxQueryCol::IsRowDirty**bool IsRowDirty(int *field*)**

Returns TRUE if the given field has been changed, but not saved.

4.165.10 wxQueryCol::IsNullable**bool IsNullable()**

Returns TRUE if a column may contain no data.

4.165.11 wxQueryCol::AppendField**void AppendField(void **buf*, long *len*)**

Appends a wxQueryField instance to the field list of the column. *len* bytes from *buf* will be copied into the field's buffer.

4.165.12 wxQueryCol::SetData**bool SetData(int *field*, void **buf*, long *len*)**

Sets the data of a field. This function finds the wxQueryField corresponding to *field* and calls wxQueryField::SetData with *buf* and *len* arguments.

4.165.13 wxQueryCol::SetName**void SetName(const wxString& *name*)**

Sets the name of a column. Only useful when creating new tables or appending columns.

4.165.14 wxQueryCol::SetNullable**void SetNullable(bool *nullable*)**

Determines whether a column may contain no data. Only useful when creating new tables or appending columns.

4.165.15 wxQueryCol::SetFieldDirty**void SetFieldDirty**(int *field*, bool *dirty* = *TRUE*)

Sets the dirty tag of a given field.

4.165.16 wxQueryCol::SetType

void SetType(short *type*) Sets the data type of a column. Only useful when creating new tables or appending columns.

4.166 wxQueryField

Represents the data item for one or several columns.

Derivation

wxObject (p. 471)

See also

wxQueryField overview (p. 923), *wxDatabase overview* (p. 923)

4.166.1 wxQueryField::wxQueryField**wxQueryField()**

Constructor. Sets type and size of the field to default values.

4.166.2 wxQueryField::~~wxQueryField**~wxQueryField()**

Destructor. Frees the associated memory depending on the field type.

4.166.3 wxQueryField::AllocData**bool AllocData()**

Allocates memory depending on the size and type of the field.

4.166.4 wxQueryField::ClearData

void ClearData()

Deletes the contents of the field buffer without deallocating the memory.

4.166.5 wxQueryField::GetData**void * GetData()**

Returns a pointer to the field buffer.

4.166.6 wxQueryField::GetSize**long GetSize()**

Returns the size of the field buffer.

4.166.7 wxQueryField::GetType**short GetType()**

Returns the type of the field (currently SQL_CHAR, SQL_VARCHAR or SQL_INTEGER).

4.166.8 wxQueryField::IsDirty**bool IsDirty()**

Returns TRUE if the data of a field has been changed, but not saved.

4.166.9 wxQueryField::SetData**bool SetData(void *data, long sz)**

Allocates memory of the size *sz* and copies the contents of *d* into the field buffer.

4.166.10 wxQueryField::SetDirty**void SetDirty(bool dirty = TRUE)**

Sets the dirty tag of a field.

4.166.11 wxQueryField::SetSize**void SetSize(long size)**

Resizes the field buffer. Stored data will be lost.

4.166.12 wxQueryField::SetType

void SetType(short type)

Sets the type of the field. Currently the types SQL_CHAR, SQL_VARCHAR and SQL_INTEGER are supported.

4.167 wxQueryLayoutInfoEvent

This event is sent when *wxLayoutAlgorithm* (p. 362) wishes to get the size, orientation and alignment of a window. More precisely, the event is sent by the *OnCalculateLayout* handler which is itself invoked by *wxLayoutAlgorithm*.

Derived from

wxEvent (p. 221)

wxObject (p. 471)

Include files

<wx/laywin.h>

Event table macros

EVT_QUERY_LAYOUT_INFO(func)	Process a <i>wxEVT_QUERY_LAYOUT_INFO</i> event, to get size, orientation and alignment from a window.
------------------------------------	-------------------------------------------------------------------------------------------------------

Data structures

```
enum wxLayoutOrientation {  
    wxLAYOUT_HORIZONTAL,  
    wxLAYOUT_VERTICAL  
};
```

```
enum wxLayoutAlignment {  
    wxLAYOUT_NONE,  
    wxLAYOUT_TOP,  
    wxLAYOUT_LEFT,  
    wxLAYOUT_RIGHT,  
    wxLAYOUT_BOTTOM,  
};
```

See also

wxCalculateLayoutEvent (p. 68), *wxSashLayoutWindow* (p. 570), *wxLayoutAlgorithm* (p. 362).

4.167.1 wxQueryLayoutInfoEvent::wxQueryLayoutInfoEvent**wxQueryLayoutInfoEvent(wxWindowID id = 0)**

Constructor.

4.167.2 wxQueryLayoutInfoEvent::GetAlignment**void GetAlignment() const**

Specifies the alignment of the window (which side of the remaining parent client area the window sticks to). One of wxLAYOUT_TOP, wxLAYOUT_LEFT, wxLAYOUT_RIGHT, wxLAYOUT_BOTTOM.

4.167.3 wxQueryLayoutInfoEvent::GetFlags**int GetFlags() const**

Returns the flags associated with this event. Not currently used.

4.167.4 wxQueryLayoutInfoEvent::GetOrientation**wxLayoutOrientation GetOrientation() const**

Returns the orientation that the event handler specified to the event object. May be one of wxLAYOUT_HORIZONTAL, wxLAYOUT_VERTICAL.

4.167.5 wxQueryLayoutInfoEvent::GetRequestedLength**int GetRequestedLength() const**

Returns the requested length of the window in the direction of the window orientation. This information is not yet used.

4.167.6 wxQueryLayoutInfoEvent::GetSize**wxSize GetSize() const**

Returns the size that the event handler specified to the event object as being the requested size of the window.

4.167.7 wxQueryLayoutInfoEvent::SetAlignment

void SetAlignment(wxLayoutAlignment *alignment*)

Call this to specify the alignment of the window (which side of the remaining parent client area the window sticks to). May be one of wxLAYOUT_TOP, wxLAYOUT_LEFT, wxLAYOUT_RIGHT, wxLAYOUT_BOTTOM.

4.167.8 wxQueryLayoutInfoEvent::SetFlags**void SetFlags(int *flags*)**

Sets the flags associated with this event. Not currently used.

4.167.9 wxQueryLayoutInfoEvent::SetOrientation**void SetOrientation(wxLayoutOrientation *orientation*)**

Call this to specify the orientation of the window. May be one of wxLAYOUT_HORIZONTAL, wxLAYOUT_VERTICAL.

4.167.10 wxQueryLayoutInfoEvent::SetRequestedLength**void SetRequestedLength(int *length*)**

Sets the requested length of the window in the direction of the window orientation. This information is not yet used.

4.167.11 wxQueryLayoutInfoEvent::SetSize**void SetSize(const wxSize& *size*)**

Call this to let the calling code know what the size of the window is.

4.168 wxRadioBox

A radio box item is used to select one of number of mutually exclusive choices. It is displayed as a vertical column or horizontal row of labelled buttons.

Derived from

wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/radiobox.h>

Window styles

wxRA_SPECIFY_ROWS	The major dimension parameter refers to the maximum number of rows.
wxRA_SPECIFY_COLS	The major dimension parameter refers to the maximum number of columns.

See also *window styles overview* (p. 959).

Event handling

EVT_RADIOBOX(id, func)	Process a <code>wxEVT_COMMAND_RADIOBOX_SELECTED</code> event, when a radiobutton is clicked.
-------------------------------	----------------------------------------------------------------------------------------------

See also

Event handling overview (p. 939), *wxRadioButton* (p. 543), *wxCheckBox* (p. 70)

4.168.1 wxRadioBox::wxRadioBox

wxRadioBox()

Default constructor.

wxRadioBox(*wxWindow** parent, *wxWindowID* id, **const wxString&** label, **const wxPoint&** point = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **int** n = 0, **const wxString** choices[] = *NULL*, **int** majorDimension = 0, **long** style = *wxRA_SPECIFY_COLS*, **const wxValidator&** validator = *wxDefaultValidator*, **const wxString&** name = *"radioBox"*)

Constructor, creating and showing a radiobox.

Parameters

parent

Parent window. Must not be `NULL`.

id

Window identifier. A value of -1 indicates a default value.

label

Label for the static box surrounding the radio buttons.

pos

Window position. If the position (-1, -1) is specified then a default position is chosen.

size

Window size. If the default size (-1, -1) is specified then a default size is chosen.

n

Number of choices with which to initialize the radiobox.

choices

An array of choices with which to initialize the radiobox.

majorDimension

Specifies the maximum number of rows (if style contains `wxRA_SPECIFY_ROWS`) or columns (if style contains `wxRA_SPECIFY_COLS`) for a two-dimensional radiobox.

style

Window style. See `wxRadioBox` (p. 537).

validator

Window validator.

name

Window name.

See also

`wxRadioBox::Create` (p. 539), `wxValidator` (p. 781)

wxPython note:

The `wxRadioBox` constructor in wxPython reduces the `nand choices` arguments are to a single argument, which is a list of strings.

4.168.2 wxRadioBox::~~wxRadioBox**~wxRadioBox()**

Destructor, destroying the radiobox item.

4.168.3 wxRadioBox::Create

```
bool Create(wxWindow* parent, wxWindowID id, const wxString& label, const wxPoint& point = wxDefaultPosition, const wxSize& size = wxDefaultSize, int n = 0, const wxString choices[] = NULL, int majorDimension = 0, long style = wxRA_SPECIFY_COLS, const wxValidator& validator = wxDefaultValidator, const wxString& name = "radioBox")
```

Creates the radiobox for two-step construction. See `wxRadioBox::wxRadioBox` (p. 538) for further details.

4.168.4 `wxRadioBox::Enable`

void Enable(const bool *enable*)

Enables or disables the entire radiobox.

void Enable(int *n*, const bool *enable*)

Enables or disables an individual button in the radiobox.

Parameters

enable

TRUE to enable, FALSE to disable.

n

The zero-based button to enable or disable.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

Enable(flag)

Enables or disables the entire radiobox.

EnableItem(n, flag)

Enables or disables an individual button in the radiobox.

4.168.5 `wxRadioBox::FindString`

int FindString(const wxString& *string*) const

Finds a button matching the given string, returning the position if found, or -1 if not found.

Parameters

string

The string to find.

4.168.6 `wxRadioBox::GetLabel`

wxString GetLabel() const

Returns the radiobox label.

wxString GetLabel(int *n*) const

Returns the label for the given button.

Parameters

n

The zero-based button index.

See also

wxRadioBox::SetLabel (p. 541)

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

GetLabel()

Returns the radiobox label.

GetItemLabel(*n*)

Returns the label for the given button.

4.168.7 wxRadioBox::GetSelection

int GetSelection() const

Returns the zero-based position of the selected button.

4.168.8 wxRadioBox::GetStringSelection

wxString GetStringSelection() const

Returns the selected string.

4.168.9 wxRadioBox::Number

int Number() const

Returns the number of buttons in the radiobox.

4.168.10 wxRadioBox::SetLabel

void SetLabel(const wxString& *label*)

Sets the radiobox label.

void SetLabel(int *n*, const wxString& *label*)

Sets a label for a radio button.

Parameters

label

The label to set.

n

The zero-based button index.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

SetLabel(string)

Sets the radiobox label.

SetItemLabel(*n*, string)

Sets a label for a radio button.

4.168.11 wxRadioBox::SetSelection

void SetSelection(int *n*)

Sets a button by passing the desired string position.

Parameters

n

The zero-based button position.

4.168.12 wxRadioBox::SetStringSelection

void SetStringSelection(const wxString& *string*)

Sets a button by passing the desired string.

Parameters

string

The label of the button to select.

4.168.13 wxRadioBox::Show

void Show(const bool *show*)

Shows or hides the entire radiobox.

void Show(int *item*, const bool *show*)

Shows or hides individual buttons.

Parameters

show

TRUE to show, FALSE to hide.

item

The zero-based position of the button to show or hide.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

Show(flag)

Shows or hides the entire radiobox.

ShowItem(n, flag)

Shows or hides individual buttons.

4.168.14 wxRadioBox::GetString

wxString GetString(int *n*) const

Returns the label for the button at the given position.

Parameters

n

The zero-based button position.

4.169 wxRadioButton

A radio button item is a button which usually denotes one of several mutually exclusive options. It has a text label next to a (usually) round button.

Derived from

wxControl (p. 125)

wxWindow (p. 798)

wxEvtHandler (p. 224)

wxObject (p. 471)

Include files

<wx/radiobut.h>

Window styles

There are no specific styles for this class.

See also *window styles overview* (p. 959).

Event handling

EVT_RADIOBUTTON(id, func)	Process a wxEVT_COMMAND_RADIOBUTTON_SELECTED event, when the radiobutton is clicked.
----------------------------------	-----------------------------------------------------------------------------------------

See also

Event handling overview (p. 939), *wxRadioBox* (p. 537), *wxCheckBox* (p. 70)

4.169.1 wxRadioButton::wxRadioButton

wxRadioButton()

Default constructor.

wxRadioButton(wxWindow* parent, wxWindowID id, const wxString& label, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0, const wxValidator& validator = wxDefaultValidator, const wxString& name = "radioButton")

Constructor, creating and showing a radio button.

Parameters

parent

Parent window. Must not be NULL.

id

Window identifier. A value of -1 indicates a default value.

label

Label for the radio button.

pos

Window position. If the position (-1, -1) is specified then a default position is chosen.

size

Window size. If the default size (-1, -1) is specified then a default size is chosen.

style

Window style. See *wxRadioButton* (p. 543).

validator

Window validator.

name

Window name.

See also

wxRadioButton::Create (p. 545), *wxValidator* (p. 781)

4.169.2 **wxRadioButton::~~wxRadioButton**

void ~wxRadioButton()

Destructor, destroying the radio button item.

4.169.3 **wxRadioButton::Create**

bool Create(*wxWindow** parent, *wxWindowID* id, **const wxString&** label, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = 0, **const wxValidator&** validator = *wxDefaultValidator*, **const wxString&** name = "radioButton")

Creates the choice for two-step construction. See *wxRadioButton::wxRadioButton* (p. 544) for further details.

4.169.4 **wxRadioButton::GetValue**

bool GetValue() **const**

Returns TRUE if the radio button is depressed, FALSE otherwise.

4.169.5 **wxRadioButton::SetValue**

void SetValue(**const bool** value)

Sets the radio button to selected or deselected status.

Parameters

value

TRUE to select, FALSE to deselect.

4.170 wxRealPoint

A **wxRealPoint** is a useful data structure for graphics operations. It contains floating point *x* and *y* members. See also *wxPoint* (p. 503) for an integer version.

Derived from

None

Include files

<wx/gdicmn.h>

See also

wxPoint (p. 503)

4.170.1 wxRealPoint::wxRealPoint

wxRealPoint()

wxRealPoint(double *x*, double *y*)

Create a point.

double *x*

double *y*

Members of the **wxRealPoint** object.

4.171 wxRect

A class for manipulating rectangles.

Derived from

None

Include files

<wx/gdicmn.h>

See also

wxPoint (p. 503), *wxSize* (p. 595)

4.171.1 wxRect::wxRect**wxRect()**

Default constructor.

wxRect(int x, int y, int width, int height)

Creates a wxRect object from x, y, width and height values.

wxRect(const wxPoint& topLeft, const wxPoint& bottomRight)

Creates a wxRect object from top-left and bottom-right points.

wxRect(const wxPoint& pos, const wxSize& size)

Creates a wxRect object from position and size values.

4.171.2 wxRect::x**int x**

x member.

4.171.3 wxRect::y**int y**

y member.

4.171.4 wxRect::width**int width**

Width member.

4.171.5 wxRect::height**int height**

Height member.

4.171.6 wxRect::GetBottom**int GetBottom() const**

Gets the bottom point of the rectangle.

4.171.7 wxRect::GetHeight**int GetHeight() const**

Gets the height member.

4.171.8 wxRect::GetLeft**int GetLeft() const**

Gets the left point of the rectangle (the same as *wxRect::GetX* (p. 549)).

4.171.9 wxRect::GetPosition**wxPoint GetPosition() const**

Gets the position.

4.171.10 wxRect::GetRight**int GetRight() const**

Gets the right point of the rectangle.

4.171.11 wxRect::GetSize**wxSize GetSize() const**

Gets the size.

4.171.12 wxRect::GetTop**int GetTop() const**

Gets the top point of the rectangle (the same as *wxRect::GetY* (p. 549)).

4.171.13 wxRect::GetWidth

int GetWidth() const

Gets the width member.

4.171.14 wxRect::GetX

int GetX() const

Gets the x member.

4.171.15 wxRect::GetY

int GetY() const

Gets the y member.

4.171.16 wxRect::SetHeight

void SetHeight(int *height*)

Sets the height.

4.171.17 wxRect::SetWidth

void SetWidth(int *width*)

Sets the width.

4.171.18 wxRect::SetX

void SetX(int *x*)

Sets the x position.

4.171.19 wxRect::SetY

void SetY(int *y*)

Sets the y position.

4.171.20 wxRect::operator =

void operator =(const wxRect& rect)

Assignment operator.

4.171.21 wxRect::operator ==

bool operator ==(const wxRect& rect)

Equality operator.

4.171.22 wxRect::operator !=

bool operator !=(const wxRect& rect)

Inequality operator.

4.172 wxRecordSet

Each `wxRecordSet` represents an ODBC database query. You can make multiple queries at a time by using multiple `wxRecordSets` with a `wxDatabase` or you can make your queries in sequential order using the same `wxRecordSet`.

Derived from

wxObject (p. 471)

Include files

<wx/odbc.h>

See also

wxRecordSet overview (p. 924), *wxDatabase overview* (p. 923)

4.172.1 wxRecordSet::wxRecordSet

wxRecordSet(wxDatabase *db, int type = wxOPEN_TYPE_DYNASET, int opt = wxOPTION_DEFAULT)

Constructor. *db* is a pointer to the `wxDatabase` instance you wish to use the `wxRecordSet` with. Currently there are two possible values of *type*:

- `wxOPEN_TYPE_DYNASET`: Loads only one record at a time into memory. The other data of the result set will be loaded dynamically when moving the cursor. This is the default type.

- `wxOPEN_TYPE_SNAPSHOT`: Loads all records of a result set at once. This will need much more memory, but will result in faster access to the ODBC data.

The *option* parameter is not used yet.

The constructor appends the `wxRecordSet` object to the parent database's list of `wxRecordSet` objects, for later destruction when the `wxDatabase` is destroyed.

4.172.2 `wxRecordSet::~~wxRecordSet`

`~wxRecordSet()`

Destructor. All data except that stored in user-defined variables will be lost. It also unlinks the `wxRecordSet` object from the parent database's list of `wxRecordSet` objects.

4.172.3 `wxRecordSet::AddNew`

`void AddNew()`

Not implemented.

4.172.4 `wxRecordSet::BeginQuery`

`bool BeginQuery(int openType, const wxString& sql = NULL, int options = wxOPTION_DEFAULT)`

Not implemented.

4.172.5 `wxRecordSet::BindVar`

`void * BindVar(int col, void *buf, long size)`

Binds a user-defined variable to the column *col*. Whenever the current field's data changes, it will be copied into *buf* (maximum *size* bytes).

`void * BindVar(const wxString& col, void *buf, long size)`

The same as above, but uses the column name as the identifier.

4.172.6 `wxRecordSet::CanAppend`

`bool CanAppend()`

Not implemented.

4.172.7 `wxRecordSet::Cancel`

void Cancel()

Not implemented.

4.172.8 wxRecordSet::CanRestart

bool CanRestart()

Not implemented.

4.172.9 wxRecordSet::CanScroll

bool CanScroll()

Not implemented.

4.172.10 wxRecordSet::CanTransact

bool CanTransact()

Not implemented.

4.172.11 wxRecordSet::CanUpdate

bool CanUpdate()

Not implemented.

4.172.12 wxRecordSet::ConstructDefaultSQL

bool ConstructDefaultSQL()

Not implemented.

4.172.13 wxRecordSet::Delete

bool Delete()

Deletes the current record. Not implemented.

4.172.14 wxRecordSet::Edit

void Edit()

Not implemented.

4.172.15 wxRecordSet::EndQuery

bool EndQuery()

Not implemented.

4.172.16 wxRecordSet::ExecuteSQL

bool ExecuteSQL(const wxString& sql)

Directly executes a SQL statement. The data will be presented as a normal result set. Note that the recordset must have been created as a snapshot, not dynaset. Dynasets will be implemented in the near future.

Examples of common SQL statements are given in *A selection of SQL commands* (p. 925).

4.172.17 wxRecordSet::FillVars

void FillVars(int recnum)

Fills in the user-defined variables of the columns. You can set these variables with `wxQueryCol::BindVar`. This function will be automatically called after every successful database operation.

4.172.18 wxRecordSet::GetColName

wxString GetColName(int col)

Returns the name of the column at position *col*. Returns NULL if *col* does not exist.

4.172.19 wxRecordSet::GetColType

short GetColType(int col)

Returns the data type of the column at position *col*. Returns `SQL_TYPE_NULL` if *col* does not exist.

short GetColType(const wxString& name)

The same as above, but uses the column name as the identifier.

See *ODBC SQL data types* (p. 924) for a list of possible data types.

4.172.20 **wxRecordSet::GetColumns**

bool GetColumns(const wxString& table = NULL)

Returns the columns of the table with the specified name. If no name is given the class member *tablename* will be used. If both names are NULL nothing will happen. The data will be presented as a normal result set, organized as follows:

0 (VARCHAR)	TABLE_QUALIFIER
1 (VARCHAR)	TABLE_OWNER
2 (VARCHAR)	TABLE_NAME
3 (VARCHAR)	COLUMN_NAME
4 (SMALLINT)	DATA_TYPE
5 (VARCHAR)	TYPE_NAME
6 (INTEGER)	PRECISION
7 (INTEGER)	LENGTH
8 (SMALLINT)	SCALE
9 (SMALLINT)	RADIX
10 (SMALLINT)	NULLABLE
11 (VARCHAR)	REMARKS

4.172.21 **wxRecordSet::GetCurrentRecord**

long GetCurrentRecord()

Not implemented.

4.172.22 **wxRecordSet::GetDatabase**

wxDatabase * GetDatabase()

Returns the wxDatabase object bound to a wxRecordSet.

4.172.23 **wxRecordSet::GetDataSources**

bool GetDataSources()

Gets the currently-defined data sources via the ODBC manager. The data will be presented as a normal result set. See the documentation for the ODBC function *SQLDataSources* for how the data is organized.

Example:

```
wxDatabase Database;
```

```

wxRecordSet *Record = new wxRecordSet(&Database);

if (!Record->GetDataSources()) {
    char buf[300];
    sprintf(buf, "%s %s\n", Database.GetErrorClass(),
Database.GetErrorMessage());
    frame->output->SetValue(buf);
}
else {
    do {
        frame->DataSource->Append((char*)Record->GetFieldDataPtr(0,
SQL_CHAR));
    } while (Record->MoveNext());
}

```

4.172.24 wxRecordSet::GetDefaultConnect

wxString GetDefaultConnect()

Not implemented.

4.172.25 wxRecordSet::GetDefaultSQL

wxString GetDefaultSQL()

Not implemented.

4.172.26 wxRecordSet::GetErrorCode

wxRETCODE GetErrorCode()

Returns the error code of the last ODBC action. This will be one of:

SQL_ERROR	General error.
SQL_INVALID_HANDLE	An invalid handle was passed to an ODBC function.
SQL_NEED_DATA	ODBC expected some data.
SQL_NO_DATA_FOUND	No data was found by this ODBC call.
SQL_SUCCESS	The call was successful.
SQL_SUCCESS_WITH_INFO	The call was successful, but further information can be obtained from the ODBC manager.

4.172.27 wxRecordSet::GetFieldData

bool GetFieldData(int col, int dataType, void *dataPtr)

Copies the current data of the column at position *col* into the buffer *dataPtr*. To be sure

to get the right type of data, the user has to pass the correct data type. The function returns FALSE if *col* does not exist or the wrong data type was given.

bool GetFieldData(const wxString& name, int dataType, void *dataPtr)

The same as above, but uses the column name as the identifier.

See *ODBC SQL data types* (p. 924) for a list of possible data types.

4.172.28 wxRecordSet::GetFieldDataPtr

void * GetFieldDataPtr(int col, int dataType)

Returns the current data pointer of the column at position *col*. To be sure to get the right type of data, the user has to pass the data type. Returns NULL if *col* does not exist or if *dataType* is incorrect.

void * GetFieldDataPtr(const wxString& name, int dataType)

The same as above, but uses the column name as the identifier.

See *ODBC SQL data types* (p. 924) for a list of possible data types.

4.172.29 wxRecordSet::GetFilter

wxString GetFilter()

Returns the current filter.

4.172.30 wxRecordSet::GetForeignKeys

bool GetPrimaryKeys(const wxString& ptable = NULL, const wxString& ftable = NULL)

Returns a list of foreign keys in the specified table (columns in the specified table that refer to primary keys in other tables), or a list of foreign keys in other tables that refer to the primary key in the specified table.

If *ptable* contains a table name, this function returns a result set containing the primary key of the specified table.

If *ftable* contains a table name, this functions returns a result set of containing all of the foreign keys in the specified table and the primary keys (in other tables) to which they refer.

If both *ptable* and *ftable* contain table names, this function returns the foreign keys in the table specified in *ftable* that refer to the primary key of the table specified in *ptable*. This

should be one key at most.

GetForeignKeys returns results as a standard result set. If the foreign keys associated with a primary key are requested, the result set is ordered by FKTABLE_QUALIFIER, FKTABLE_OWNER, FKTABLE_NAME, and KEY_SEQ. If the primary keys associated with a foreign key are requested, the result set is ordered by PKTABLE_QUALIFIER, PKTABLE_OWNER, PKTABLE_NAME, and KEY_SEQ. The following table lists the columns in the result set.

0 (VARCHAR)	PKTABLE_QUALIFIER
1 (VARCHAR)	PKTABLE_OWNER
2 (VARCHAR)	PKTABLE_NAME
3 (VARCHAR)	PKCOLUMN_NAME
4 (VARCHAR)	FKTABLE_QUALIFIER
5 (VARCHAR)	FKTABLE_OWNER
6 (VARCHAR)	FKTABLE_NAME
7 (VARCHAR)	FKCOLUMN_NAME
8 (SMALLINT)	KEY_SEQ
9 (SMALLINT)	UPDATE_RULE
10 (SMALLINT)	DELETE_RULE
11 (VARCHAR)	FK_NAME
12 (VARCHAR)	PK_NAME

4.172.31 wxRecordSet::GetNumberCols

long GetNumberCols()

Returns the number of columns in the result set.

4.172.32 wxRecordSet::GetNumberFields

int GetNumberFields()

Not implemented.

4.172.33 wxRecordSet::GetNumberParams

int GetNumberParams()

Not implemented.

4.172.34 wxRecordSet::GetNumberRecords

long GetNumberRecords()

Returns the number of records in the result set.

4.172.35 wxRecordSet::GetPrimaryKeys

bool GetPrimaryKeys(const wxString& table = NULL)

Returns the column names that comprise the primary key of the table with the specified name. If no name is given the class member *tablename* will be used. If both names are NULL nothing will happen. The data will be presented as a normal result set, organized as follows:

0 (VARCHAR)	TABLE_QUALIFIER
1 (VARCHAR)	TABLE_OWNER
2 (VARCHAR)	TABLE_NAME
3 (VARCHAR)	COLUMN_NAME
4 (SMALLINT)	KEY_SEQ
5 (VARCHAR)	PK_NAME

4.172.36 wxRecordSet::GetOptions

int GetOptions()

Returns the options of the wxRecordSet. Options are not supported yet.

4.172.37 wxRecordSet::GetResultSet

bool GetResultSet()

Copies the data presented by ODBC into wxRecordSet. Depending on the wxRecordSet type all or only one record(s) will be copied. Usually this function will be called automatically after each successful database operation.

4.172.38 wxRecordSet::GetSortString

wxString GetSortString()

Not implemented.

4.172.39 wxRecordSet::GetSQL

wxString GetSQL()

Not implemented.

4.172.40 wxRecordSet::GetTableName

wxString GetTableName()

Returns the name of the current table.

4.172.41 wxRecordSet::GetTables

bool GetTables()

Gets the tables of a database. The data will be presented as a normal result set, organized as follows:

0 (VARCHAR)	TABLE_QUALIFIER
1 (VARCHAR)	TABLE_OWNER
2 (VARCHAR)	TABLE_NAME
3 (VARCHAR)	TABLE_TYPE (TABLE, VIEW, SYSTEM TABLE, GLOBAL TEMPORARY, LOCAL TEMPORARY, ALIAS, SYNONYM, or database-specific type)
4 (VARCHAR)	REMARKS

4.172.42 wxRecordSet::GetType**int GetType()**

Returns the type of the wxRecordSet: wxOPEN_TYPE_DYNASET or wxOPEN_TYPE_SNAPSHOT. See the wxRecordSet description for details.

4.172.43 wxRecordSet::GoTo**bool GoTo(long n)**

Moves the cursor to the record with the number n, where the first record has the number 0.

4.172.44 wxRecordSet::IsBOF**bool IsBOF()**

Returns TRUE if the user tried to move the cursor before the first record in the set.

4.172.45 wxRecordSet::IsFieldDirty**bool IsFieldDirty(int field)**

Returns TRUE if the given field has been changed but not saved yet.

bool IsFieldDirty(const wxString& name)

Same as above, but uses the column name as the identifier.

4.172.46 wxRecordSet::IsFieldNull

bool IsFieldNull(int *field*)

Returns TRUE if the given field has no data.

bool IsFieldNull(const wxString& *name*)

Same as above, but uses the column name as the identifier.

4.172.47 wxRecordSet::IsColNullable

bool IsColNullable(int *col*)

Returns TRUE if the given column may contain no data.

bool IsColNullable(const wxString& *name*)

Same as above, but uses the column name as the identifier.

4.172.48 wxRecordSet::IsEOF

bool IsEOF()

Returns TRUE if the user tried to move the cursor behind the last record in the set.

4.172.49 wxRecordSet::IsDeleted

bool IsDeleted()

Not implemented.

4.172.50 wxRecordSet::IsOpen

bool IsOpen()

Returns TRUE if the parent database is open.

4.172.51 wxRecordSet::Move

bool Move(long *rows*)

Moves the cursor a given number of rows. Negative values are allowed.

4.172.52 wxRecordSet::MoveFirst

bool MoveFirst()

Moves the cursor to the first record.

4.172.53 wxRecordSet::MoveLast

bool MoveLast()

Moves the cursor to the last record.

4.172.54 wxRecordSet::MoveNext

bool MoveNext()

Moves the cursor to the next record.

4.172.55 wxRecordSet::MovePrev

bool MovePrev()

Moves the cursor to the previous record.

4.172.56 wxRecordSet::Query

bool Query(const wxString& columns, const wxString& table, const wxString& filter = NULL)

Start a query. An SQL string of the following type will automatically be generated and executed: "SELECT columns FROM table WHERE filter".

4.172.57 wxRecordSet::RecordCountFinal

bool RecordCountFinal()

Not implemented.

4.172.58 wxRecordSet::Requery

bool Requery()

Re-executes the last query. Not implemented.

4.172.59 wxRecordSet::SetFieldDirty

void SetFieldDirty(int field, bool dirty = TRUE)

Sets the dirty tag of the field field. Not implemented.

void SetFieldDirty(const wxString& name, bool dirty = TRUE)

Same as above, but uses the column name as the identifier.

4.172.60 wxRecordSet::SetDefaultSQL

void SetDefaultSQL(const wxString& s)

Not implemented.

4.172.61 wxRecordSet::SetFieldNull

void SetFieldNull(void **p*, bool *isNull* = *TRUE*)

Not implemented.

4.172.62 wxRecordSet::SetOptions

void SetOptions(int *opt*)

Sets the options of the wxRecordSet. Not implemented.

4.172.63 wxRecordSet::SetTableName

void SetTableName(const wxString& *tablename*)

Specify the name of the table you want to use.

4.172.64 wxRecordSet::SetType

void SetType(int *type*)

Sets the type of the wxRecordSet. See the wxRecordSet class description for details.

4.172.65 wxRecordSet::Update

bool Update()

Writes back the current record. Not implemented.

4.173 wxRegion

A wxRegion represents a simple or complex region on a device context or window. It uses reference counting, so copying and assignment operations are fast.

Derived from

wxGDIObject (p. 295)

wxObject (p. 471)

Include files

<wx/region.h>

See also

wxRegionIterator (p. 566)

4.173.1 wxRegion::wxRegion**wxRegion()**

Default constructor.

wxRegion(long x, long y, long width, long height)

Constructs a rectangular region with the given position and size.

wxRegion(const wxPoint& topLeft, const wxPoint& bottomRight)

Constructs a rectangular region from the top left point and the bottom right point.

wxRegion(const wxRect& rect)

Constructs a rectangular region a wxRect object.

wxRegion(const wxRegion& region)

Constructs a region by copying another region.

4.173.2 wxRegion::~~wxRegion**~wxRegion()**

Destructor.

4.173.3 wxRegion::Clear**void Clear()**

Clears the current region.

4.173.4 wxRegion::Contains**wxRegionContain Contains(long& x, long& y) const**

Returns a value indicating whether the given point is contained within the region.

wxRegionContain Contains(const wxPoint& pt) const

Returns a value indicating whether the given point is contained within the region.

wxRegionContain Contains(long& x, long& y, long& width, long& height) const

Returns a value indicating whether the given rectangle is contained within the region.

wxRegionContain Contains(const wxRect& rect) const

Returns a value indicating whether the given rectangle is contained within the region.

Return value

The return value is one of wxOutRegion, wxPartRegion and wxInRegion.

On Windows, only wxOutRegion and wxInRegion are returned; a value wxInRegion then indicates that all or some part of the region is contained in this region.

4.173.5 wxRegion::GetBox

void GetBox(long& x, long& y, long& width, long& height) const

Returns the outer bounds of the region.

wxRect GetBox() const

Returns the outer bounds of the region.

4.173.6 wxRegion::Intersect

bool Intersect(long x, long y, long width, long height)

Finds the intersection of this region and another, rectangular region, specified using position and size.

bool Intersect(const wxRect& rect)

Finds the intersection of this region and another, rectangular region.

bool Intersect(const wxRegion& region)

Finds the intersection of this region and another region.

Return value

TRUE if successful, FALSE otherwise.

Remarks

Creates the intersection of the two regions, that is, the parts which are in both regions. The result is stored in this region.

4.173.7 wxRegion::IsEmpty**bool IsEmpty() const****bool IsEmpty()**

Returns TRUE if the region is empty, FALSE otherwise.

bool Subtract(const wxRect& rect)

Subtracts a rectangular region from this region.

bool Subtract(const wxRegion& region)

Subtracts a region from this region.

Return value

TRUE if successful, FALSE otherwise.

Remarks

This operation combines the parts of 'this' region that are not part of the second region. The result is stored in this region.

4.173.8 wxRegion::Union**bool Union(long x, long y, long width, long height)**

Finds the union of this region and another, rectangular region, specified using position and size.

bool Union(const wxRect& rect)

Finds the union of this region and another, rectangular region.

bool Union(const wxRegion& region)

Finds the union of this region and another region.

Return value

TRUE if successful, FALSE otherwise.

Remarks

This operation creates a region that combines all of this region and the second region. The result is stored in this region.

4.173.9 wxRegion::Xor**bool Xor**(long *x*, long *y*, long *width*, long *height*)

Finds the Xor of this region and another, rectangular region, specified using position and size.

bool Xor(const wxRect& *rect*)

Finds the Xor of this region and another, rectangular region.

bool Xor(const wxRegion& *region*)

Finds the Xor of this region and another region.

Return value

TRUE if successful, FALSE otherwise.

Remarks

This operation creates a region that combines all of this region and the second region, except for any overlapping areas. The result is stored in this region.

4.173.10 wxRegion::operator =**void operator =**(const wxRegion& *region*)

Copies *region* by reference counting.

4.174 wxRegionIterator

This class is used to iterate through the rectangles in a region, typically when examining the damaged regions of a window within an `OnPaint` call.

To use it, construct an iterator object on the stack and loop through the regions, testing the object and incrementing the iterator at the end of the loop.

See `wxWindow::OnPaint` (p. 825) for an example of use.

Derived from

`wxObject` (p. 471)

Include files

<wx/region.h>

See also

wxWindow::OnPaint (p. 825)

4.174.1 wxRegionIterator::wxRegionIterator

wxRegionIterator()

Default constructor.

wxRegionIterator(const wxRegion& region)

Creates an iterator object given a region.

4.174.2 wxRegionIterator::GetX

long GetX() const

Returns the x value for the current region.

4.174.3 wxRegionIterator::GetY

long GetY() const

Returns the y value for the current region.

4.174.4 wxRegionIterator::GetW

long GetW() const

An alias for GetWidth.

4.174.5 wxRegionIterator::GetWidth

long GetWidth() const

Returns the width value for the current region.

4.174.6 wxRegionIterator::GetH

long GetH() const

An alias for `GetHeight`.

4.174.7 `wxRegionIterator::GetHeight`

`long GetWidth() const`

Returns the width value for the current region.

4.174.8 `wxRegionIterator::GetRect`

`wxRect GetRect() const`

Returns the current rectangle.

4.174.9 `wxRegionIterator::HaveRects`

`bool HaveRects() const`

Returns `TRUE` if there are still some rectangles; otherwise returns `FALSE`.

4.174.10 `wxRegionIterator::Reset`

`void Reset()`

Resets the iterator to the beginning of the rectangles.

`void Reset(const wxRegion& region)`

Resets the iterator to the given region.

4.174.11 `wxRegionIterator::operator ++`

`void operator ++()`

Increment operator. Increments the iterator to the next region.

`wxPython note:`

A `wxPython` alias for this operator is called `Next`.

4.174.12 `wxRegionIterator::operator bool`

`operator bool() const`

Returns TRUE if there are still some rectangles; otherwise returns FALSE.

You can use this to test the iterator object as if it were of type bool.

4.175 wxSashEvent

A sash event is sent when the sash of a *wxSashWindow* (p. 573) has been dragged by the user.

Derived from

wxCommandEvent (p. 103)

wxEvent (p. 221)

wxObject (p. 471)

Include files

<wx/sashwin.h>

Event table macros

To process an activate event, use these event handler macros to direct input to a member function that takes a *wxSashEvent* argument.

EVT_SASH_DRAGGED(id, func)	Process a <i>wxEVT_SASH_DRAGGED</i> event, when the user has finished dragging a sash.
EVT_SASH_DRAGGED_RANGE(id1, id2, func)	Process a <i>wxEVT_SASH_DRAGGED_RANGE</i> event, when the user has finished dragging a sash. The event handler is called when windows with ids in the given range have their sashes dragged.

Data structures

```
enum wxSashDragStatus
{
    wxSASH_STATUS_OK,
    wxSASH_STATUS_OUT_OF_RANGE
};
```

Remarks

When a sash belonging to a sash window is dragged by the user, and then released, this event is sent to the window, where it may be processed by an event table entry in a derived class, a plug-in event handler or an ancestor class.

Note that the *wxSashWindow* doesn't change the window's size itself. It relies on the

application's event handler to do that. This is because the application may have to handle other consequences of the resize, or it may wish to veto it altogether. The event handler should look at the drag rectangle: see *wxSashEvent::GetDragRect* (p. 570) to see what the new size of the window would be if the resize were to be applied. It should also call *wxSashEvent::GetDragStatus* (p. 570) to see whether the drag was OK or out of the current allowed range.

See also

wxSashWindow (p. 573), *Event handling overview* (p. 939)

4.175.1 wxSashEvent::wxSashEvent

wxSashEvent(int *id* = 0, wxSashEdgePosition *edge* = wxSASH_NONE)

Constructor.

4.175.2 wxSashEvent::GetEdge

wxSashEdgePosition GetEdge() const

Returns the dragged edge. The return value is one of wxSASH_TOP, wxSASH_RIGHT, wxSASH_BOTTOM, wxSASH_LEFT.

4.175.3 wxSashEvent::GetDragRect

wxRect GetDragRect() const

Returns the rectangle representing the new size the window would be if the resize was applied. It is up to the application to set the window size if required.

4.175.4 wxSashEvent::GetDragStatus

wxSashDragStatus GetDragStatus() const

Returns the status of the sash: one of wxSASH_STATUS_OK, wxSASH_STATUS_OUT_OF_RANGE. If the drag caused the notional bounding box of the window to flip over, for example, the drag will be out of range.

4.176 wxSashLayoutWindow

wxSashLayoutWindow responds to OnCalculateLayout events generated by *wxLayoutAlgorithm* (p. 362). It allows the application to use simple accessors to specify how the window should be laid out, rather than having to respond to events. The fact

that the class derives from `wxSashWindow` allows sashes to be used if required, to allow the windows to be user-resizable.

The documentation for `wxLayoutAlgorithm` (p. 362) explains the purpose of this class in more detail.

Derived from

`wxSashWindow` (p. 573)
`wxWindow` (p. 798)
`wxEvtHandler` (p. 224)
`wxObject` (p. 471)

Include files

`<wx/laywin.h>`

Window styles

See `wxSashWindow` (p. 573).

Event handling

This class handles the `EVT_QUERY_LAYOUT_INFO` and `EVT_CALCULATE_LAYOUT` events for you. However, if you use sashes, see `wxSashWindow` (p. 573) for relevant event information.

See also `wxLayoutAlgorithm` (p. 362) for information about the layout events.

See also

`wxLayoutAlgorithm` (p. 362), `wxSashWindow` (p. 573), *Event handling overview* (p. 939)

4.176.1 `wxSashLayoutWindow::wxSashLayoutWindow`

`wxSashLayoutWindow()`

Default constructor.

`wxSashLayoutWindow(wxSashLayoutWindow* parent, wxSashLayoutWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxCLIP_CHILDREN | wxSW_3D, const wxString& name = "layoutWindow")`

Constructs a sash layout window, which can be a child of a frame, dialog or any other non-control window.

Parameters

parent

Pointer to a parent window.

id

Window identifier. If -1, will automatically create an identifier.

pos

Window position. `wxDefaultPosition` is (-1, -1) which indicates that `wxSashLayoutWindows` should generate a default position for the window. If using the `wxSashLayoutWindow` class directly, supply an actual position.

size

Window size. `wxDefaultSize` is (-1, -1) which indicates that `wxSashLayoutWindows` should generate a default size for the window.

style

Window style. For window styles, please see *wxSashLayoutWindow* (p. 570).

name

Window name.

4.176.2 `wxSashLayoutWindow::~~wxSashLayoutWindow`

`~wxSashLayoutWindow()`

Destructor.

4.176.3 `wxSashLayoutWindow::GetAlignment`

`wxLayoutAlignment GetAlignment() const`

Returns the alignment of the window: one of `wxLAYOUT_TOP`, `wxLAYOUT_LEFT`, `wxLAYOUT_RIGHT`, `wxLAYOUT_BOTTOM`.

4.176.4 `wxSashLayoutWindow::GetOrientation`

`wxLayoutOrientation GetOrientation() const`

Returns the orientation of the window: one of `wxLAYOUT_HORIZONTAL`, `wxLAYOUT_VERTICAL`.

4.176.5 `wxSashLayoutWindow::OnCalculateLayout`

`void OnCalculateLayout(wxCalculateLayoutEvent& event)`

The default handler for the event that is generated by `wxLayoutAlgorithm`. The implementation of this function calls `wxCalculateLayoutEvent::SetRect` to shrink the provided size according to how much space this window takes up. For further details, see *wxLayoutAlgorithm* (p. 362) and *wxCalculateLayoutEvent* (p. 68).

4.176.6 `wxSashLayoutWindow::OnQueryLayoutInfo`

void OnQueryLayoutInfo(`wxQueryLayoutInfoEvent&` *event*)

The default handler for the event that is generated by `OnCalculateLayout` to get size, alignment and orientation information for the window. The implementation of this function uses member variables as set by accessors called by the application. For further details, see *wxLayoutAlgorithm* (p. 362) and *wxQueryLayoutInfoEvent* (p. 535).

4.176.7 `wxSashLayoutWindow::SetAlignment`

void SetAlignment(`wxLayoutAlignment` *alignment*)

Sets the alignment of the window (which edge of the available parent client area the window is attached to). *alignment* is one of `wxLAYOUT_TOP`, `wxLAYOUT_LEFT`, `wxLAYOUT_RIGHT`, `wxLAYOUT_BOTTOM`.

4.176.8 `wxSashLayoutWindow::SetDefaultSize`

void SetDefaultSize(`const wxSize&` *size*)

Sets the default dimensions of the window. The dimension other than the orientation will be fixed to this value, and the orientation dimension will be ignored and the window stretched to fit the available space.

4.176.9 `wxSashLayoutWindow::SetOrientation`

void SetOrientation(`wxLayoutOrientation` *orientation*)

Sets the orientation of the window (the direction the window will stretch in, to fill the available parent client area). *orientation* is one of `wxLAYOUT_HORIZONTAL`, `wxLAYOUT_VERTICAL`.

4.177 `wxSashWindow`

`wxSashWindow` allows any of its edges to have a sash which can be dragged to resize the window. The actual content window will be created by the application as a child of `wxSashWindow`. The window (or an ancestor) will be notified of a drag via a *wxSashEvent* (p. 569) notification.

Derived from

wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/sashwin.h>

Window styles

The following styles apply in addition to the normal *wxWindow* styles.

wxSW_3D Draws the sashes in 3D.

See also *window styles overview* (p. 959).

Event handling

EVT_SASH_DRAGGED(id, func) Process a *wxEVT_SASH_DRAGGED* event, when the user has finished dragging a sash.

EVT_SASH_DRAGGED_RANGE(id1, id2, func) Process a *wxEVT_SASH_DRAGGED_RANGE* event, when the user has finished dragging a sash. The event handler is called when windows with ids in the given range have their sashes dragged.

Data types

```
enum wxSashEdgePosition {  
    wxSASH_TOP = 0,  
    wxSASH_RIGHT,  
    wxSASH_BOTTOM,  
    wxSASH_LEFT,  
    wxSASH_NONE = 100  
};
```

See also

wxSashEvent (p. 569), *wxSashLayoutWindow* (p. 570), *Event handling overview* (p. 939)

4.177.1 **wxSashWindow::wxSashWindow**

wxSashWindow()

Default constructor.

wxSashWindow(wxSashWindow* parent, wxSashWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxCLIP_CHILDREN | wxSW_3D, const wxString& name = "sashWindow")

Constructs a sash window, which can be a child of a frame, dialog or any other non-control window.

Parameters

parent

Pointer to a parent window.

id

Window identifier. If -1, will automatically create an identifier.

pos

Window position. wxDefaultPosition is (-1, -1) which indicates that wxSashWindows should generate a default position for the window. If using the wxSashWindow class directly, supply an actual position.

size

Window size. wxDefaultSize is (-1, -1) which indicates that wxSashWindows should generate a default size for the window.

style

Window style. For window styles, please see *wxSashWindow* (p. 573).

name

Window name.

4.177.2 wxSashWindow::~~wxSashWindow**~wxSashWindow()**

Destructor.

4.177.3 wxSashWindow::GetSashVisible

bool GetSashVisible(wxSashEdgePosition edge) const

Returns TRUE if a sash is visible on the given edge, FALSE otherwise.

Parameters

edge

Edge. One of `wxSASH_TOP`, `wxSASH_RIGHT`, `wxSASH_BOTTOM`, `wxSASH_LEFT`.

[See also](#)

`wxSashWindow::SetSashVisible` (p. 577)

4.177.4 `wxSashWindow::GetMaximumSizeX`

`int GetMaximumSizeX() const`

Gets the maximum window size in the x direction.

4.177.5 `wxSashWindow::GetMaximumSizeY`

`int GetMaximumSizeY() const`

Gets the maximum window size in the y direction.

4.177.6 `wxSashWindow::GetMinimumSizeX`

`int GetMinimumSizeX()`

Gets the minimum window size in the x direction.

4.177.7 `wxSashWindow::GetMinimumSizeY`

`int GetMinimumSizeY(int min) const`

Gets the minimum window size in the y direction.

4.177.8 `wxSashWindow::HasBorder`

`bool HasBorder(wxSashEdgePosition edge) const`

Returns TRUE if the sash has a border, FALSE otherwise.

[Parameters](#)

edge

Edge. One of `wxSASH_TOP`, `wxSASH_RIGHT`, `wxSASH_BOTTOM`, `wxSASH_LEFT`.

[See also](#)

wxSashWindow::SetSashBorder (p. 578)

4.177.9 wxSashWindow::SetMaximumSizeX

void SetMaximumSizeX(int *min*)

Sets the maximum window size in the x direction.

4.177.10 wxSashWindow::SetMaximumSizeY

void SetMaximumSizeY(int *min*)

Sets the maximum window size in the y direction.

4.177.11 wxSashWindow::SetMinimumSizeX

void SetMinimumSizeX(int *min*)

Sets the minimum window size in the x direction.

4.177.12 wxSashWindow::SetMinimumSizeY

void SetMinimumSizeY(int *min*)

Sets the minimum window size in the y direction.

4.177.13 wxSashWindow::SetSashVisible

void SetSashVisible(wxSashEdgePosition *edge*, bool *visible*)

Call this function to make a sash visible or invisible on a particular edge.

Parameters

edge

Edge to change. One of wxSASH_TOP, wxSASH_RIGHT, wxSASH_BOTTOM, wxSASH_LEFT.

visible

TRUE to make the sash visible, FALSE to make it invisible.

See also

wxSashWindow::GetSashVisible (p. 575)

4.177.14 wxSashWindow::SetSashBorder

void SetSashBorder(**wxSashEdgePosition** *edge*, **bool** *hasBorder*)

Call this function to give the sash a border, or remove the border.

Parameters

edge

Edge to change. One of `wxSASH_TOP`, `wxSASH_RIGHT`, `wxSASH_BOTTOM`, `wxSASH_LEFT`.

hasBorder

TRUE to give the sash a border visible, FALSE to remove it.

See also

wxSashWindow::HashBorder (p. 576)

4.178 wxScreenDC

A `wxScreenDC` can be used to paint on the screen. This should normally be constructed as a temporary stack object; don't store a `wxScreenDC` object.

Derived from

wxDC (p. 151)

Include files

<wx/dcscreen.h>

See also

wxDC (p. 151), *wxMemoryDC* (p. 415), *wxPaintDC* (p. 483), *wxClientDC* (p. 81), *wxWindowDC* (p. 843)

4.178.1 wxScreenDC::wxScreenDC

wxScreenDC()

Constructor.

4.178.2 wxScreenDC::StartDrawingOnTop

bool StartDrawingOnTop(wxWindow* window)

bool StartDrawingOnTop(wxRect* rect = NULL)

Use this in conjunction with *EndDrawingOnTop* (p. 579) to ensure that drawing to the screen occurs on top of existing windows. Without this, some window systems (such as X) only allow drawing to take place underneath other windows.

By using the first form of this function, an application is specifying that the area that will be drawn on coincides with the given window.

By using the second form, an application can specify an area of the screen which is to be drawn on. If NULL is passed, the whole screen is available.

It is recommended that an area of the screen is specified because with large regions, flickering effects are noticeable when destroying the temporary transparent window used to implement this feature.

You might use this pair of functions when implementing a drag feature, for example as in the *wxSplitterWindow* (p. 626) implementation.

4.178.3 wxScreenDC::EndDrawingOnTop

bool EndDrawingOnTop()

Use this in conjunction with *StartDrawingOnTop* (p. 578).

This function destroys the temporary window created to implement on-top drawing (X only).

4.179 wxScrollBar

A *wxScrollBar* is a control that represents a horizontal or vertical scrollbar. It is distinct from the two scrollbars that some windows provide automatically, but the two types of scrollbar share the way events are received.

Derived from

wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/scrolbar.h>

Remarks

A scrollbar has the following main attributes: *range*, *thumb size*, *page size*, and *position*.

The range is the total number of units associated with the view represented by the scrollbar. For a table with 15 columns, the range would be 15.

The thumb size is the number of units that are currently visible. For the table example, the window might be sized so that only 5 columns are currently visible, in which case the application would set the thumb size to 5. When the thumb size becomes the same as or greater than the range, the scrollbar will be automatically hidden on most platforms.

The page size is the number of units that the scrollbar should scroll by, when 'paging' through the data. This value is normally the same as the thumb size length, because it is natural to assume that the visible window size defines a page.

The scrollbar position is the current thumb position.

Most applications will find it convenient to provide a function called **AdjustScrollbars** which can be called initially, from an **OnSize** event handler, and whenever the application data changes in size. It will adjust the view, object and page size according to the size of the window and the size of the data.

Window styles

wxSB_HORIZONTAL	Specifies a horizontal scrollbar.
wxSB_VERTICAL	Specifies a vertical scrollbar.

See also *window styles overview* (p. 959).

Event handling

To process input from a scrollbar, use one of these event handler macros to direct input to member functions that take a *wxScrollEvent* (p. 584) argument:

EVT_COMMAND_SCROLL(id, func)	Catch all scroll commands.
EVT_COMMAND_TOP(id, func)	Catch a command to put the scroll thumb at the maximum position.
EVT_COMMAND_BOTTOM(id, func)	Catch a command to put the scroll thumb at the maximum position.
EVT_COMMAND_LINEUP(id, func)	Catch a line up command.
EVT_COMMAND_LINEDOWN(id, func)	Catch a line down command.
EVT_COMMAND_PAGEUP(id, func)	Catch a page up command.
EVT_COMMAND_PAGEDOWN(id, func)	Catch a page down command.
EVT_COMMAND_THUMBTRACK(id, func)	Catch a thumbtrack command (continuous movement of the scroll thumb).

See also

Scrolling overview (p. 932), *Event handling overview* (p. 939), *wxScrolledWindow* (p. 586)

4.179.1 **wxScrollBar::wxScrollBar**

wxScrollBar()

Default constructor.

```
wxScrollBar(wxWindow* parent, wxWindowID id, const wxPoint& pos =  
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxSB_HORIZONTAL, const wxValidator& validator = wxDefaultValidator, const  
wxString& name = "scrollBar")
```

Constructor, creating and showing a scrollbar.

Parameters

parent

Parent window. Must not be NULL.

id

Window identifier. A value of -1 indicates a default value.

pos

Window position. If the position (-1, -1) is specified then a default position is chosen.

size

Window size. If the default size (-1, -1) is specified then a default size is chosen.

style

Window style. See *wxScrollBar* (p. 579).

validator

Window validator.

name

Window name.

See also

wxScrollBar::Create (p. 582), *wxValidator* (p. 781)

4.179.2 **wxScrollBar::~~wxScrollBar**

void ~wxScrollBar()

Destructor, destroying the scrollbar.

4.179.3 **wxScrollBar::Create**

bool Create(**wxWindow*** *parent*, **wxWindowID** *id*, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = *wxSB_HORIZONTAL*, **const wxValidator&** *validator* = *wxDefaultValidator*, **const wxString&** *name* = "scrollBar")

Scrollbar creation function called by the scrollbar constructor. See *wxScrollBar::wxScrollBar* (p. 581) for details.

4.179.4 **wxScrollBar::GetRange**

int GetRange() **const**

Returns the length of the scrollbar.

[See also](#)

wxScrollBar::SetScrollbar (p. 583)

4.179.5 **wxScrollBar::GetPageSize**

int GetPageSize() **const**

Returns the page size of the scrollbar. This is the number of scroll units that will be scrolled when the user pages up or down. Often it is the same as the thumb size.

[See also](#)

wxScrollBar::SetScrollbar (p. 583)

4.179.6 **wxScrollBar::GetThumbPosition**

int GetThumbPosition() **const**

Returns the current position of the scrollbar thumb.

[See also](#)

wxScrollBar::SetThumbPosition (p. 583)

4.179.7 **wxScrollBar::GetThumbLength**

int GetThumbLength() **const**

Returns the thumb or 'view' size.

See also

wxScrollBar::SetScrollbar (p. 583)

4.179.8 wxScrollBar::SetThumbPosition

void SetThumbPosition(int *viewStart*)

Sets the position of the scrollbar.

Parameters

viewStart

The position of the scrollbar thumb.

See also

wxScrollBar::GetThumbPosition (p. 582)

4.179.9 wxScrollBar::SetScrollbar

virtual void SetScrollbar(int *position*, int *thumbSize*, int *range*, int *pageSize*, const bool *refresh* = *TRUE*)

Sets the scrollbar properties.

Parameters

position

The position of the scrollbar in scroll units.

thumbSize

The size of the thumb, or visible portion of the scrollbar, in scroll units.

range

The maximum position of the scrollbar.

pageSize

The size of the page size in scroll units. This is the number of units the scrollbar will scroll when it is paged up or down. Often it is the same as the thumb size.

refresh

TRUE to redraw the scrollbar, FALSE otherwise.

Remarks

Let's say you wish to display 50 lines of text, using the same font. The window is sized

so that you can only see 16 lines at a time.

You would use:

```
scrollbar->SetScrollbar(0, 16, 50, 15);
```

The page size is 1 less than the thumb size so that the last line of the previous page will be visible on the next page, to help orient the user.

Note that with the window at this size, the thumb position can never go above 50 minus 16, or 34.

You can determine how many lines are currently visible by dividing the current view size by the character height in pixels.

When defining your own scrollbar behaviour, you will always need to recalculate the scrollbar settings when the window size changes. You could therefore put your scrollbar calculations and `SetScrollbar` call into a function named `AdjustScrollbars`, which can be called initially and also from a `wxWindow::OnSize` (p. 828) event handler function.

See also

Scrolling overview (p. 932), *wxWindow::SetScrollbar* (p. 837), *wxScrolledWindow* (p. 586)

4.180 wxScrollEvent

A scroll event holds information about events sent from scrollbars and scrolling windows.

Derived from

wxCommandEvent (p. 103)
wxEvent (p. 221)
wxObject (p. 471)

Include files

```
<wx/event.h>
```

Event table macros

To process a scroll event, use these event handler macros to direct input to member functions that take a `wxScrollEvent` argument. You can use `EVT_COMMAND_SCROLL...` macros with window IDs for when intercepting scroll events from controls, or `EVT_SCROLL...` macros without window IDs for intercepting scroll events from the receiving window.

EVT_SCROLL(func)	Process all scroll events.
-------------------------	----------------------------

EVT_SCROLL_TOP(func)	Process wxEVT_SCROLL_TOP scroll-to-top events.
EVT_SCROLL_BOTTOM(func)	Process wxEVT_SCROLL_TOP scroll-to-bottom events.
EVT_SCROLL_LINEUP(func)	Process wxEVT_SCROLL_LINEUP line up events.
EVT_SCROLL_LINEDOWN(func)	Process wxEVT_SCROLL_LINEDOWN line down events.
EVT_SCROLL_PAGEUP(func)	Process wxEVT_SCROLL_PAGEUP page up events.
EVT_SCROLL_PAGEDOWN(func)	Process wxEVT_SCROLL_PAGEDOWN page down events.
EVT_SCROLL_THUMBTRACK(func)	Process wxEVT_SCROLL_THUMBTRACK thumbtrack events (frequent events sent as the user drags the thumbtrack).
EVT_COMMAND_SCROLL(id, func)	Process all scroll events.
EVT_COMMAND_SCROLL_TOP(id, func)	Process wxEVT_SCROLL_TOP scroll-to-top events.
EVT_COMMAND_SCROLL_BOTTOM(id, func)	Process wxEVT_SCROLL_TOP scroll-to-bottom events.
EVT_COMMAND_SCROLL_LINEUP(id, func)	Process wxEVT_SCROLL_LINEUP line up events.
EVT_COMMAND_SCROLL_LINEDOWN(id, func)	Process wxEVT_SCROLL_LINEDOWN line down events.
EVT_COMMAND_SCROLL_PAGEUP(id, func)	Process wxEVT_SCROLL_PAGEUP page up events.
EVT_COMMAND_SCROLL_PAGEDOWN(id, func)	Process wxEVT_SCROLL_PAGEDOWN page down events.
EVT_COMMAND_SCROLL_THUMBTRACK(id, func)	Process wxEVT_SCROLL_THUMBTRACK thumbtrack events (frequent events sent as the user drags the thumbtrack).

Remarks

Note that unless specifying a scroll control identifier, you will need to test for scrollbar orientation with *wxScrollEvent::GetOrientation* (p. 586), since horizontal and vertical scroll events are processed using the same event handler.

See also

wxWindow::OnScroll (p. 827), *wxScrollBar* (p. 579), *Event handling overview* (p. 939)

4.180.1 wxScrollEvent::wxScrollEvent

wxScrollEvent(WXTYPE *commandType* = 0, int *id* = 0, int *pos* = 0, int *orientation* = 0)

Constructor.

4.180.2 wxScrollEvent::GetOrientation

int GetOrientation() const

Returns wxHORIZONTAL or wxVERTICAL, depending on the orientation of the scrollbar.

4.180.3 wxScrollEvent::GetPosition

int GetPosition() const

Returns the position of the scrollbar.

4.181 wxScrolledWindow

The wxScrolledWindow class manages scrolling for its client area, transforming the coordinates according to the scrollbar positions, and setting the scroll positions, thumb sizes and ranges according to the area in view.

As with all windows, an application can draw onto a wxScrolledWindow using a *device context* (p. 926).

You have the option of handling the *OnPaint* (p. 590) handler or overriding the *OnDraw* (p. 590) function, which is passed a pre-scrolled device context (prepared by *PrepareDC* (p. 590)).

If you don't wish to calculate your own scrolling, you must call *PrepareDC* when not drawing from within *OnDraw*, to set the device origin for the device context according to the current scroll position.

Derived from

wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/scrolwin.h>

Window styles

wxRETAINED Uses a backing pixmap to speed refreshes. Motif only.

See also *window styles overview* (p. 959).

Remarks

Use `wxScrolledWindow` for applications where the user scrolls by a fixed amount, and where a 'page' can be interpreted to be the current visible portion of the window. For more sophisticated applications, use the `wxScrolledWindow` implementation as a guide to build your own scroll behaviour.

See also

`wxScrollBar` (p. 579), `wxClientDC` (p. 81), `wxPaintDC` (p. 483)

4.181.1 `wxScrolledWindow::wxScrolledWindow`

`wxScrolledWindow()`

Default constructor.

`wxScrolledWindow(wxWindow* parent, wxWindowID id = -1, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxHSCROLL | wxVSCROLL, const wxString& name = "scrolledWindow")`

Constructor.

Parameters

parent

Parent window.

id

Window identifier. A value of -1 indicates a default value.

pos

Window position. If a position of (-1, -1) is specified then a default position is chosen.

size

Window size. If a size of (-1, -1) is specified then the window is sized appropriately.

style

Window style. See *wxScrolledWindow* (p. 586).

name

Window name.

Remarks

The window is initially created without visible scrollbars. Call *wxScrolledWindow::SetScrollbars* (p. 591) to specify how big the virtual window size should be.

4.181.2 **wxScrolledWindow::~~wxScrolledWindow**

~wxScrolledWindow()

Destructor.

4.181.3 **wxScrolledWindow::Create**

bool Create(*wxWindow* parent*, *wxWindowID id* = -1, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = *wxHSCROLL* | *wxVSCROLL*, **const wxString&** *name* = "scrolledWindow")

Creates the window for two-step construction. Derived classes should call or replace this function. See *wxScrolledWindow::wxScrolledWindow* (p. 587) for details.

4.181.4 **wxScrolledWindow::EnableScrolling**

void EnableScrolling(**const bool** *xScrolling*, **const bool** *yScrolling*)

Enable or disable physical scrolling in the given direction. Physical scrolling is the physical transfer of bits up or down the screen when a scroll event occurs. If the application scrolls by a variable amount (e.g. if there are different font sizes) then physical scrolling will not work, and you should switch it off.

Parameters

xScrolling

If TRUE, enables physical scrolling in the x direction.

yScrolling

If TRUE, enables physical scrolling in the y direction.

Remarks

Physical scrolling may not be available on all platforms. Where it is available, it is enabled by default.

4.181.5 **wxScrolledWindow::GetScrollPixelsPerUnit**

void GetScrollPixelsPerUnit(*int* xUnit*, *int* yUnit*) **const**

Get the number of pixels per scroll unit (line), in each direction, as set by *wxScrolledWindow::SetScrollbars* (p. 591). A value of zero indicates no scrolling in that direction.

Parameters

xUnit
Receives the number of pixels per horizontal unit.

yUnit
Receives the number of pixels per vertical unit.

See also

wxScrolledWindow::SetScrollbars (p. 591), *wxScrolledWindow::GetVirtualSize* (p. 589)

4.181.6 **wxScrolledWindow::GetVirtualSize**

void GetVirtualSize(int* x, int* y) const

Gets the size in device units of the scrollable window area (as opposed to the client size, which is the area of the window currently visible).

Parameters

x
Receives the length of the scrollable window, in pixels.

y
Receives the height of the scrollable window, in pixels.

Remarks

Use *wxDC::DeviceToLogicalX* (p. 153) and *wxDC::DeviceToLogicalY* (p. 154) to translate these units to logical units.

See also

wxScrolledWindow::SetScrollbars (p. 591), *wxScrolledWindow::GetScrollPixelsPerUnit* (p. 588)

4.181.7 **wxScrolledWindow::IsRetained**

bool IsRetained() const

TRUE if the window has a backing bitmap.

4.181.8 wxScrolledWindow::PrepareDC**void PrepareDC(wxDC& dc)**

Call this function to prepare the device context for drawing a scrolled image. It sets the device origin according to the current scroll position.

PrepareDC is called automatically within the default *wxScrolledWindow::OnPaint* (p. 590) event handler, so your *wxScrolledWindow::OnDraw* (p. 590) override will be passed a 'pre-scrolled' device context. However, if you wish to draw from outside of *OnDraw* (via *OnPaint*), or you wish to implement *OnPaint* yourself, you must call this function yourself. For example:

```
void MyWindow::OnEvent(wxMouseEvent& event)
{
    wxClientDC dc(this);
    PrepareDC(dc);

    dc.SetPen(*wxBLACK_PEN);
    float x, y;
    event.Position(&x, &y);
    if (xpos > -1 && ypos > -1 && event.Dragging())
    {
        dc.DrawLine(xpos, ypos, x, y);
    }
    xpos = x;
    ypos = y;
}
```

4.181.9 wxScrolledWindow::OnDraw**virtual void OnDraw(wxDC& dc)**

Called by the default *wxScrolledWindow::OnPaint* (p. 590) implementation to allow the application to define painting behaviour without having to worry about calling *wxScrolledWindow::PrepareDC* (p. 590).

4.181.10 wxScrolledWindow::OnPaint**void OnPaint(wxPaintEvent& event)**

Sent to the window when the window must be refreshed.

For more details, see *wxWindow::OnPaint* (p. 825).

The default implementation for *wxScrolledWindow*'s *OnPaint* handler is simply:

```
void wxScrolledWindow::OnPaint(wxPaintEvent& event)
{
    wxPaintDC dc(this);
    PrepareDC(dc);
}
```

```
        OnDraw(dc);  
    }
```

4.181.11 wxScrolledWindow::OnScroll

void OnScroll(wxScrollEvent& event)

Override this function to intercept scroll events. This member function implements the default scroll behaviour. If you do not call the default function, you will have to manage all scrolling behaviour including drawing the window contents at an appropriate position relative to the scrollbars.

For more details, see *wxWindow::OnScroll* (p. 827).

[See also](#)

wxScrollEvent (p. 584)

4.181.12 wxScrolledWindow::Scroll

void Scroll(int x, int y)

Scrolls a window so the view start is at the given point.

Parameters

x
The x position to scroll to, in scroll units.

y
The y position to scroll to, in scroll units.

Remarks

The positions are in scroll units, not pixels, so to convert to pixels you will have to multiply by the number of pixels per scroll increment. If either parameter is -1, that position will be ignored (no change in that direction).

[See also](#)

wxScrolledWindow::SetScrollbars (p. 591), *wxScrolledWindow::GetScrollPixelsPerUnit* (p. 588)

4.181.13 wxScrolledWindow::SetScrollbars

void SetScrollbars(int pixelsPerUnitX, int pixelsPerUnitY, int noUnitsX, int noUnitsY, int xPos = 0, int yPos = 0)

Sets up vertical and/or horizontal scrollbars.

Parameters

pixelsPerUnitX

Pixels per scroll unit in the horizontal direction.

pixelsPerUnitY

Pixels per scroll unit in the vertical direction.

noUnitsX

Number of units in the horizontal direction.

noUnitsY

Number of units in the vertical direction.

xPos

Position to initialize the scrollbars in the horizontal direction, in scroll units.

yPos

Position to initialize the scrollbars in the vertical direction, in scroll units.

Remarks

The first pair of parameters give the number of pixels per 'scroll step', i.e. amount moved when the up or down scroll arrows are pressed. The second pair gives the length of scrollbar in scroll steps, which sets the size of the virtual window.

xPos and *yPos* optionally specify a position to scroll to immediately.

For example, the following gives a window horizontal and vertical scrollbars with 20 pixels per scroll step, and a size of 50 steps (1000 pixels) in each direction.

```
window->SetScrollbars(20, 20, 50, 50);
```

`wxScrolledWindow` manages the page size itself, using the current client window size as the page size.

Note that for more sophisticated scrolling applications, for example where scroll steps may be variable according to the position in the document, it will be necessary to derive a new class from `wxWindow`, overriding **OnSize** and adjusting the scrollbars appropriately.

4.181.14 `wxScrolledWindow::ViewStart`

void ViewStart(int* x, int* y) const

Get the position at which the visible portion of the window starts.

Parameters

- x*
Receives the first visible *x* position in scroll units.
- y*
Receives the first visible *y* position in scroll units.

Remarks

If either of the scrollbars is not at the home position, *x* and/or *y* will be greater than zero. Combined with `wxWindow::GetClientSize` (p. 807), the application can use this function to efficiently redraw only the visible portion of the window. The positions are in logical scroll units, not pixels, so to convert to pixels you will have to multiply by the number of pixels per scroll increment.

See also

`wxScrolledWindow::SetScrollbars` (p. 591)

4.182 wxSingleChoiceDialog

This class represents a dialog that shows a list of strings, and allows the user to select one. Double-clicking on a list item is equivalent to single-clicking and then pressing OK.

Derived from

`wxDialog` (p. 178)
`wxWindow` (p. 798)
`wxEvtHandler` (p. 224)
`wxObject` (p. 471)

Include files

<wx/choicdlg.h>

See also

`wxSingleChoiceDialog` overview (p. 918)

4.182.1 wxSingleChoiceDialog::wxSingleChoiceDialog

`wxSingleChoiceDialog(wxWindow* parent, const wxString& message, const wxString& caption, int n, const wxString* choices, char clientData = NULL, long style = wxOK | wxCANCEL | wxCENTRE, const wxPoint& pos = wxDefaultPosition)`**

Constructor, taking an array of `wxString` choices and optional client data.

`wxSingleChoiceDialog(wxWindow* parent, const wxString& message, const`

wxString& caption, const wxStringList& choices, char clientData = NULL, long style = wxOK | wxCANCEL | wxCENTRE, const wxPoint& pos = wxDefaultPosition)**

Constructor, taking a string list and optional client data.

Parameters

parent

Parent window.

message

Message to show on the dialog.

caption

The dialog caption.

n

The number of choices.

choices

An array of strings, or a string list, containing the choices.

style

A dialog style (bitlist) containing flags chosen from the following:

wxOK	Show an OK button.
wxCANCEL	Show a Cancel button.
wxCENTRE	Centre the message. Not Windows.

pos

Dialog position. Not Windows.

Remarks

Use *wxSingleChoiceDialog::ShowModal* (p. 595) to show the dialog.

4.182.2 **wxSingleChoiceDialog::~wxSingleChoiceDialog**

~wxSingleChoiceDialog()

Destructor.

4.182.3 **wxSingleChoiceDialog::GetSelection**

int GetSelection() const

Returns the index of selected item.

4.182.4 wxSingleChoiceDialog::GetSelectionClientData**char* GetSelectionClientData() const**

Returns the client data associated with the selection.

4.182.5 wxSingleChoiceDialog::GetStringSelection**wxString GetStringSelection() const**

Returns the selected string.

4.182.6 wxSingleChoiceDialog::ShowModal**int ShowModal()**

Shows the dialog, returning either `wxID_OK` or `wxID_CANCEL`.

4.183 wxSize

A **wxSize** is a useful data structure for graphics operations. It simply contains integer `x` and `y` members.

wxPython note:

wxPython defines aliases for the `x` and `y` members named `width` and `height` since it makes much more sense for sizes. There is also corresponding aliases for the `GetWidth` and `GetHeight` methods.

Derived from

None

Include files

<wx/gdicmn.h>

See also

wxPoint (p. 503), *wxRealPoint* (p. 546)

4.183.1 wxSize::wxSize**wxSize()**

wxSize(int x, int y)

Creates a size object.

4.183.2 wxSize::x

int x

x member.

4.183.3 wxSize::y

int y

y member.

4.183.4 wxSize::GetX

int GetX() const

Gets the x member.

4.183.5 wxSize::GetY

int GetY() const

Gets the y member.

4.183.6 wxSize::Set

void Set(int x, int y)

Sets the x and y members.

4.183.7 wxSize::operator =

void operator =(const wxSize& sz)

Assignment operator.

4.184 wxSizeEvent

A size event holds information about size change events.

Derived from

wxEvt (p. 221)
wxObject (p. 471)

Include files

<wx/event.h>

Event table macros

To process a size event, use this event handler macro to direct input to a member function that takes a *wxSizeEvent* argument.

EVT_SIZE(func) Process a *wxEVT_SIZE* event.

See also

wxWindow::OnSize (p. 828), *wxSize* (p. 595), *Event handling overview* (p. 939)

4.184.1 wxSizeEvent::wxSizeEvent

wxSizeEvent(const wxSize& sz, int id = 0)

Constructor.

4.184.2 wxSizeEvent::GetSize

wxSize GetSize() const

Returns the entire size of the window generating the size change event.

4.185 wxSlider

A slider is a control with a handle which can be pulled back and forth to change the value.

In Windows versions below Windows 95, a scrollbar is used to simulate the slider. In Windows 95, the track bar control is used.

Slider events are handled in the same way as a scrollbar.

Derived from

wxControl (p. 125)

wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/slider.h>

Window styles

wxSL_HORIZONTAL	Displays the slider horizontally.
wxSL_VERTICAL	Displays the slider vertically.
wxSL_AUTOTICKS	Displays tick marks.
wxSL_LABELS	Displays minimum, maximum and value labels.
wxSL_LEFT	Displays ticks on the left, if a vertical slider.
wxSL_RIGHT	Displays ticks on the right, if a vertical slider.
wxSL_TOP	Displays ticks on the top, if a horizontal slider.
wxSL_SELRANGE	Allows the user to select a range on the slider. Windows 95 only.

See also *window styles overview* (p. 959).

Event handling

To process input from a slider, use one of these event handler macros to direct input to member functions that take a *wxScrollEvent* (p. 584) argument:

EVT_COMMAND_SCROLL(id, func)	Catch all scroll commands.
EVT_COMMAND_TOP(id, func)	Catch a command to put the scroll thumb at the maximum position.
EVT_COMMAND_BOTTOM(id, func)	Catch a command to put the scroll thumb at the maximum position.
EVT_COMMAND_LINEUP(id, func)	Catch a line up command.
EVT_COMMAND_LINEDOWN(id, func)	Catch a line down command.
EVT_COMMAND_PAGEUP(id, func)	Catch a page up command.
EVT_COMMAND_PAGEDOWN(id, func)	Catch a page down command.
EVT_COMMAND_THUMBTRACK(id, func)	Catch a thumbtrack command (continuous movement of the scroll thumb).
EVT_SLIDER(id, func)	Process a wxEVT_COMMAND_SLIDER_UPDATED event, when the slider is moved. Though provided for backward compatibility, this is obsolete.

See also

Event handling overview (p. 939), *wxScrollBar* (p. 579)

4.185.1 wxSlider::wxSlider

wxSlider()

Default slider.

wxSlider(wxWindow* parent, wxWindowID id, int value, int minValue, int maxValue, const wxPoint& point = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxSL_HORIZONTAL, const wxValidator& validator = wxDefaultValidator, const wxString& name = "slider")

Constructor, creating and showing a slider.

Parameters

parent

Parent window. Must not be NULL.

id

Window identifier. A value of -1 indicates a default value.

value

Initial position for the slider.

minValue

Minimum slider position.

maxValue

Maximum slider position.

size

Window size. If the default size (-1, -1) is specified then a default size is chosen.

style

Window style. See *wxSlider* (p. 597).

validator

Window validator.

name

Window name.

See also

wxSlider::Create (p. 600), *wxValidator* (p. 781)

4.185.2 wxSlider::~~wxSlider

void ~wxSlider()

Destructor, destroying the slider.

4.185.3 wxSlider::ClearSel

void ClearSel()

Clears the selection, for a slider with the **wxSL_SELRANGE** style.

Remarks

Windows 95 only.

4.185.4 wxSlider::ClearTicks

void ClearTicks()

Clears the ticks.

Remarks

Windows 95 only.

4.185.5 wxSlider::Create

bool Create(**wxWindow*** *parent*, **wxWindowID** *id*, **int** *value*, **int** *minValue*, **int** *maxValue*, **const wxPoint&** *point* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = *wxSL_HORIZONTAL*, **const wxValidator&** *validator* = *wxDefaultValidator*, **const wxString&** *name* = "slider")

Used for two-step slider construction. See *wxSlider::wxSlider* (p. 599) for further details.

4.185.6 wxSlider::GetLineSize

int GetLineSize() const

Returns the line size.

See also

wxSlider::SetLineSize (p. 603)

4.185.7 wxSlider::GetMax

int GetMax() const

Gets the maximum slider value.

See also

wxSlider::GetMin (p. 601), *wxSlider::SetRange* (p. 603)

4.185.8 wxSlider::GetMin**int GetMin() const**

Gets the minimum slider value.

See also

wxSlider::GetMin (p. 601), *wxSlider::SetRange* (p. 603)

4.185.9 wxSlider::GetPageSize**int GetPageSize() const**

Returns the page size.

See also

wxSlider::SetPageSize (p. 604)

4.185.10 wxSlider::GetSelEnd**int GetSelEnd() const**

Returns the selection end point.

Remarks

Windows 95 only.

See also

wxSlider::GetSelStart (p. 601), *wxSlider::SetSelection* (p. 604)

4.185.11 wxSlider::GetSelStart**int GetSelStart() const**

Returns the selection start point.

Remarks

Windows 95 only.

See also

wxSlider::GetSelEnd (p. 601), *wxSlider::SetSelection* (p. 604)

4.185.12 **wxSlider::GetThumbLength**

int GetThumbLength() const

Returns the thumb length.

Remarks

Windows 95 only.

See also

wxSlider::SetThumbLength (p. 604)

4.185.13 **wxSlider::GetTickFreq**

int GetTickFreq() const

Returns the tick frequency.

Remarks

Windows 95 only.

See also

wxSlider::SetTickFreq (p. 603)

4.185.14 **wxSlider::GetValue**

int GetValue() const

Gets the current slider value.

See also

wxSlider::GetMin (p. 601), *wxSlider::GetMax* (p. 600), *wxSlider::SetValue* (p. 605)

4.185.15 wxSlider::SetRange**void SetRange**(int *minValue*, int *maxValue*)

Sets the minimum and maximum slider values.

See also

wxSlider::GetMin (p. 601), *wxSlider::GetMax* (p. 600)

4.185.16 wxSlider::SetTickFreq**void SetTickFreq**(int *n*, int *pos*)

Sets the tick mark frequency and position.

Parameters

n

Frequency. For example, if the frequency is set to two, a tick mark is displayed for every other increment in the slider's range.

pos

Position. Must be greater than zero. TODO: what is this for?

Remarks

Windows 95 only.

See also

wxSlider::GetTickFreq (p. 602)

4.185.17 wxSlider::SetLineSize**void SetLineSize**(int *lineSize*)

Sets the line size for the slider.

Parameters

lineSize

The number of steps the slider moves when the user moves it up or down a line.

See also

wxSlider::GetLineSize (p. 600)

4.185.18 wxSlider::SetPageSize

void SetPageSize(int *pageSize*)

Sets the page size for the slider.

Parameters

pageSize

The number of steps the slider moves when the user pages up or down.

See also

wxSlider::GetPageSize (p. 601)

4.185.19 wxSlider::SetSelection

void SetSelection(int *startPos*, int *endPos*)

Sets the selection.

Parameters

startPos

The selection start position.

endPos

The selection end position.

Remarks

Windows 95 only.

See also

wxSlider::GetSelStart (p. 601), *wxSlider::GetSelEnd* (p. 601)

4.185.20 wxSlider::SetThumbLength

void SetThumbLength(int *len*)

Sets the slider thumb length.

Parameters

len

The thumb length.

Remarks

Windows 95 only.

See also

wxSlider::GetThumbLength (p. 602)

4.185.21 **wxSlider::SetTick**

void SetTick(int *tickPos*)

Sets a tick position.

Parameters

tickPos

The tick position.

Remarks

Windows 95 only.

See also

wxSlider::SetTickFreq (p. 603)

4.185.22 **wxSlider::SetValue**

void SetValue(int *value*)

Sets the slider position.

Parameters

value

The slider position.

See also

wxSlider::GetValue (p. 602)

4.186 **wxSocketAddress**

You are unlikely to need to use this class: only `wxSocketBase` uses it.

Derived from

wxObject (p. 471)

Include files

<wx/socket.h>

See also

wxSocketBase (p. 607)

4.186.1 wxSocketAddress::wxSocketAddress

wxSocketAddress()

Default constructor.

4.186.2 wxSocketAddress::~~wxSocketAddress

~wxSocketAddress()

Default destructor.

4.186.3 wxSocketAddress::Clear

void Clear()

Delete all informations about the address.

4.186.4 wxSocketAddress::Build

void Build(struct sockaddr *& addr, size_t& len)

Build a coded socket address.

4.186.5 wxSocketAddress::Disassemble

void Disassemble(struct sockaddr *addr, size_t len)

Decode a socket address. **Actually, you don't have to use this function: only *wxSocketBase* use it.**

4.186.6 wxSocketAddress::SockAddrLen**int SockAddrLen();**

Returns the length of the socket address.

4.187 wxSocketBase**Derived from**

wxEvtHandler (p. 224)

Include files

<wx/socket.h>

Event handling

To process events from a socket, use the following event handler macro to direct input to member functions that take a *wxSocketEvent* (p. 617) argument.

EVT_SOCKET(id, func) A socket event occurred.

See also

wxSocketEvent (p. 617)
wxSocketClient (p. 615)
wxSocketServer (p. 620)

4.187.1 wxSocketBase::wxSocketBase**wxSocketBase()**

Default constructor but don't use it, you must use *wxSocketClient* (p. 615) or *wxSocketServer* (p. 620).

4.187.2 wxSocketBase::~~wxSocketBase**~wxSocketBase()**

Destroys the *wxSocketBase* object.

4.187.3 wxSocketBase::Ok

bool Ok() const

Returns TRUE if the socket is initialized and ready and FALSE in other cases.

4.187.4 wxSocketBase::Error**bool Error() const**

Returns TRUE if an error occurred.

4.187.5 wxSocketBase::IsConnected**bool IsConnected() const**

Returns TRUE if the socket is connected.

4.187.6 wxSocketBase::IsData**bool IsData() const**

Returns TRUE if some data is arrived on the socket.

4.187.7 wxSocketBase::IsDisconnected**bool IsDisconnected() const**

Returns TRUE if the socket is disconnected.

4.187.8 wxSocketBase::IsNoWait**bool IsNoWait() const**

Returns TRUE if the socket mustn't wait.

4.187.9 wxSocketBase::LastCount**size_t LastCount() const**

Returns the number of bytes read or written by the last IO call.

4.187.10 wxSocketBase::LastError**int LastError() const**

Returns an error in the `errno` format (see your C programmer's guide).

4.187.11 `wxSocketBase::Peek`

`wxSocketBase& Peek(char * buffer, size_t nbytes)`

This function peeks a buffer of *nbytes* bytes from the socket. Peeking a buffer doesn't delete it from the system socket in-queue.

Parameters

buffer
Buffer where to put peeked data.

nbytes
Number of bytes.

Return value

Returns a reference to the current object.

See also

`wxSocketBase::Error` (p. 608)
`wxSocketBase::LastCount` (p. 608)
`wxSocketBase::LastError` (p. 608)

4.187.12 `wxSocketBase::Read`

`wxSocketBase& Read(char * buffer, size_t nbytes)`

This function reads a buffer of *nbytes* bytes from the socket.

Parameters

buffer
Buffer where to put read data.

nbytes
Number of bytes.

Return value

Returns a reference to the current object.

Remark/Warning

By default, `Read` uses an internal asynchronous manager: it will send data when the

socket requests them. It is particularly interesting when you enter a long data transfer (e.g. a big file, an image, ...). But it is also buggy when you simply discuss with the peer using user data. In this case, wxSocket prepares itself to send data (Write wait for them to be sent) and during a GUI refresh the user enters new data, which involves a new Read call though the previous isn't finished. Well, in most cases it can work but it might fail too. So I advise you to use the SPEED flag, which disables the asynchronous manager, when you just want to discuss with the peer.

This remark is also valid for all IO call.

See also

wxSocketBase::Error (p. 608), *wxSocketBase::LastCount* (p. 608),
wxSocketBase::LastError (p. 608)

4.187.13 wxSocketBase::SetFlags

void SetFlags(wxSockFlags flags)

wxSocketBase::NONE	Normal fonctionnalités.
wxSocketBase::NOWAIT	Get the available data in the input queue and exit immediately.
wxSocketBase::WAITALL	Wait for all required data unless an error occurred.
wxSocketBase::SPEED	Disable the asynchronous IO functionality.

4.187.14 wxSocketBase::Write

wxSocketBase& Write(const char * buffer, size_t nbytes)

This function writes a buffer of *nbytes* bytes from the socket.

Remark/Warning

By default, Write uses an internal asynchronous manager: it will send data when the socket requests them. It is particularly interesting when you enter a long data transfer (e.g. a big file, an image, ...). But it is also buggy when you simply discuss with the peer using user data. In this case, wxSocket prepares itself to send data (Write wait for them to be sent) and during a GUI refresh the user enters new data, which involves a new Write call though the previous isn't finished. Well, in most cases it can work but it might fail too. So I advise you to use the SPEED flag, which disables the asynchronous manager, when you just want to discuss with the peer.

Parameters

buffer
 Buffer where to get the data to write.

nbytes
 Number of bytes.

Return value

Returns a reference to the current object.

See also

wxSocketBase::Error (p. 608)
wxSocketBase::LastCount (p. 608)
wxSocketBase::LastError (p. 608)

4.187.15 **wxSocketBase::WriteMsg**

wxSocketBase& WriteMsg(const char * *buffer*, size_t *nbytes*)

This function writes a buffer of *nbytes* bytes from the socket. But it writes a short header before so that ReadMsg can alloc the right size for the buffer. So a buffer sent with WriteMsg **must** be read with ReadMsg.

Parameters

buffer
Buffer where to put data peeked.

nbytes
Number of bytes.

Return value

Returns a reference to the current object.

See also

wxSocketBase::Error (p. 608)
wxSocketBase::LastCount (p. 608)
wxSocketBase::LastError (p. 608)
wxSocketBase::ReadMsg (p. 611)

4.187.16 **wxSocketBase::ReadMsg**

wxSocketBase& ReadMsg(char * *buffer*, size_t *nbytes*)

This function reads a buffer sent by WriteMsg on a socket. If the buffer passed to the function isn't big enough, the function filled it and then discard the bytes left. This function always wait for the buffer to be entirely filled.

Parameters

buffer

Buffer where to put read data.

nbytes

Number of bytes allocated for the buffer.

Return value

Returns a reference to the current object.

See also

wxSocketBase::Error (p. 608)

wxSocketBase::LastCount (p. 608)

wxSocketBase::LastError (p. 608)

wxSocketBase::WriteMsg (p. 611)

4.187.17 **wxSocketBase::UnRead**

wxSocketBase& UnRead(const char * *buffer*, size_t *nbytes*)

This function unreads a buffer. It means that the buffer is put in the top of the incoming queue. But, it is put also at the end of all unread buffers. It is useful for sockets because we can't seek it.

Parameters

buffer

Buffer to be unread.

nbytes

Number of bytes.

Return value

Returns a reference to the current object.

See also

wxSocketBase::Error (p. 608)

wxSocketBase::LastCount (p. 608)

wxSocketBase::LastError (p. 608)

4.187.18 **wxSocketBase::Discard**

wxSocketBase& Discard()

This function simply deletes all bytes in the incoming queue. This function doesn't wait.

4.187.19 wxSocketBase::Wait**bool Wait(long seconds = -1, long microsecond = 0)**

This function waits for an event: it could be an incoming byte, the possibility for the client to write, a lost connection, an incoming connection, an established connection.

Parameters*seconds*

Number of seconds to wait. By default, it waits infinitely.

microsecond

Number of microseconds to wait.

Return value

Returns TRUE if an event occurred, FALSE if the timeout was reached.

See also*wxSocketBase::WaitForRead* (p. 613)*wxSocketBase::WaitForWrite* (p. 614)*wxSocketBase::WaitForLost* (p. 614)**4.187.20 wxSocketBase::WaitForRead****bool WaitForRead(long seconds = -1, long microsecond = 0)**

This function waits for a read event.

Parameters*seconds*

Number of seconds to wait. By default, it waits infinitely.

microsecond

Number of microseconds to wait.

Return value

Returns TRUE if a byte arrived, FALSE if the timeout was reached.

See also*wxSocketBase::Wait* (p. 613)*wxSocketBase::WaitForWrite* (p. 614)*wxSocketBase::WaitForLost* (p. 614)

4.187.21 wxSocketBase::WaitForWrite**bool WaitForWrite**(long *seconds* = -1, long *microsecond* = 0)

This function waits for a write event.

Parameters*seconds*

Number of seconds to wait. By default, it waits infinitely.

microsecond

Number of microseconds to wait.

Return value

Returns TRUE if a write event occurred, FALSE if the timeout was reached.

See also*wxSocketBase::Wait* (p. 613)*wxSocketBase::WaitForRead* (p. 613)*wxSocketBase::WaitForLost* (p. 614)**4.187.22 wxSocketBase::WaitForLost****bool Wait**(long *seconds* = -1, long *microsecond* = 0)

This function waits for a "lost" event. For instance, the peer may have closed the connection, or the connection may have been broken.

Parameters*seconds*

Number of seconds to wait. By default, it waits infinitely.

microsecond

Number of microseconds to wait.

Return value

Returns TRUE if a "lost" event occurred, FALSE if the timeout was reached.

See also*wxSocketBase::WaitForRead* (p. 613)*wxSocketBase::WaitForWrite* (p. 614)*wxSocketBase::WaitForLost* (p. 614)

4.187.23 wxSocketBase::RestoreState**void RestoreState()**

This function restores a previously saved state.

[See also](#)

wxSocketBase::SaveState (p. 615)

4.187.24 wxSocketBase::SaveState**void SaveState()**

This function saves the current state of the socket object in a stack: actually it saves all flags and the state of the asynchronous callbacks.

[See also](#)

wxSocketBase::RestoreState (p. 615)

4.187.25 wxSocketBase::SetEventHandler**void SetEventHandler(wxEvtHandler& *evt_hdlr*, int *id* = -1)**

Sets an event handler to be called when a socket event occurred.

Parameters

evt_hdlr
Specifies the event handler you want to use.

id
The id of socket event.

[See also](#)

wxSocketEvent (p. 617)

4.188 wxSocketClient**Derived from**

wxSocketBase (p. 607)

Include files

<wx/socket.h>

4.188.1 **wxSocketClient::wxSocketClient**

wxSocketClient(**wxSockFlags** *flags* = *wxSocketBase::NONE*)

Constructs a new wxSocketClient.

Warning ! The new socket client needs to be registered to a socket handler (See *wxSocketHandler* (p. 618)).

Parameters

flags

Socket flags (See *wxSocketBase::SetFlags* (p. 610))

4.188.2 **wxSocketClient::~~wxSocketClient**

~wxSocketClient()

Destroys a wxSocketClient object.

4.188.3 **wxSocketClient::Connect**

bool **Connect**(**wxSockAddress&** *address*, **bool** *wait* = *TRUE*)

Connects to a server using the specified address. If *wait* is TRUE, Connect will wait for the socket ready to send or receive data.

Parameters

address

Address of the server.

wait

If true, waits for the connection to be ready.

Return value

Returns TRUE if the connection is established and no error occurs.

See also

wxSocketClient::WaitOnConnect (p. 616)

4.188.4 **wxSocketClient::WaitOnConnect**

bool WaitOnConnect(long seconds = -1, long microseconds = 0)

Wait for a "connect" event.

See also

wxSocketBase::Wait (p. 613) for a detailed description.

4.189 wxSocketEvent

This event class contains information about socket events.

Derived from

wxEvent (p. 221)

Include files

<wx/socket.h>

Event table macros

To process a socket event, use these event handler macros to direct input to member functions that take a *wxSocketEvent* argument.

EVT_SOCKET(id, func)	Process a socket event, supplying the member function.
-----------------------------	--------------------------------------------------------

See also

wxSocketHandler (p. 618), *wxSocketBase* (p. 607), *wxSocketClient* (p. 615), *wxSocketServer* (p. 620)

4.189.1 wxSocketEvent::wxSocketEvent

wxSocketEvent(int id = 0)

Constructor.

4.189.2 wxSocketEvent::SocketEvent

wxSocketBase::wxRequestEvent SocketEvent() const

Returns the socket event type.

4.190 wxSocketHandler

Derived from

wxObject (p. 471)

Include files

<wx/socket.h>

4.190.1 wxSocketHandler::wxSocketHandler

wxSocketHandler()

Constructs a new wxSocketHandler.

It is advised to use *wxSocketHandler::Master* (p. 619) to get a socket handler. But creating a socket handler is useful to group many sockets.

4.190.2 wxSocketHandler::~~wxSocketHandler

~wxSocketHandler()

Destroys a wxSocketHandler object.

4.190.3 wxSocketHandler::Register

void Register(wxSocketBase *socket)

Register a socket: if it is already registered in this handler it will just return immediately.

Parameters

socket
Socket to be registered.

4.190.4 wxSocketHandler::UnRegister

void UnRegister(wxSocketBase *socket)

UnRegister a socket: if it isn't registered in this handler it will just return.

Parameters

socket

Socket to be unregistered.

4.190.5 wxSocketHandler::Count

unsigned long Count() const

Returns the number of sockets registered in the handler.

Return value

Number of sockets registered.

4.190.6 wxSocketHandler::CreateServer

**wxSocketServer * CreateServer(wxSockAddress& address,
wxSocketBase::wxSockFlags flags = wxSocketbase::NONE)**

Creates a new wxSocketServer object. The object is automatically registered to the current socket handler. For a detailed description of the parameters, see *wxSocketServer::wxSocketServer* (p. 620).

Return value

Returns a new socket server.

4.190.7 wxSocketHandler::CreateClient

**wxSocketServer * CreateClient(wxSocketBase::wxSockFlags flags =
wxSocketbase::NONE)**

Creates a new wxSocketClient object. The object is automatically registered to the current socket handler.

For a detailed description of the parameters, see *wxSocketClient::Connect* (p. 616).

Return value

Returns a new socket client.

4.190.8 wxSocketHandler::Master

static wxSocketHandler& Master()

Returns a default socket handler.

4.190.9 wxSocketHandler::Wait**int Wait**(long *seconds*, long *microseconds*)

Wait for an event on all registered sockets.

Parameters*seconds*

Number of seconds to wait. By default, it waits infinitely.

microsecond

Number of microseconds to wait.

Return value

Returns 0 if a timeout occurred, else the number of events detected.

See also

wxSocketBase::Wait (p. 613)

4.190.10 wxSocketHandler::YieldSock**void YieldSock**()

Execute pending requests in all registered sockets.

4.191 wxSocketServer**Derived from**

wxSocketBase (p. 607)

Include files

<wx/socket.h>

4.191.1 wxSocketServer::wxSocketServer**wxSocketServer**(wxSockAddress& *address*, wxSockFlags *flags* = *wxSocketBase::NONE*)

Constructs a new wxSocketServer.

Warning ! The created object needs to be registered to a socket handler (see *wxSocketHandler* (p. 618)).

Parameters

address

Specifies the local address for the server (e.g. port number).

flags

Socket flags (See *wxSocketBase::SetFlags* (p. 610))

4.191.2 **wxSocketServer::~wxSocketServer**

~wxSocketServer()

Destroys a *wxSocketServer* object (it doesn't close the accepted connection).

4.191.3 **wxSocketServer::Accept**

wxSocketBase * Accept()

Creates a new object *wxSocketBase* and accepts an incoming connection. **Warning !** This function will block the GUI.

Return value

Returns an opened socket connection.

See also

wxSocketServer::AcceptWith (p. 621)

4.191.4 **wxSocketServer::AcceptWith**

bool AcceptWith(wxSocketBase& socket)

Accept an incoming connection using the specified socket object. This is useful when someone wants to inherit *wxSocketBase*.

Parameters

socket

Socket to be initialized

Return value

Returns TRUE if no error occurs, else FALSE.

4.192 wxSocketInputStream

Derived from

wxInputStream (p. 346)

Include files

<wx/sckstrm.h>

See also

wxStreamBuffer (p. 648), *wxSocketBase* (p. 607)

4.192.1 wxSocketInputStream::wxSocketInputStream

wxSocketInputStream(wxSocketBase& s)

Initializes a new read-only socket stream using the specified initialized socket connection.

4.193 wxSocketOutputStream

Derived from

wxOutputStream (p. 476)

Include files

<wx/sckstrm.h>

See also

wxStreamBuffer (p. 648), *wxSocketBase* (p. 607)

4.193.1 wxSocketOutputStream::wxSocketOutputStream

wxSocketOutputStream(wxSocketBase& s)

Initializes a new write-only socket stream using the specified initialized socket connection.

4.194 wxSpinButton

A `wxSpinButton` has two small up and down (or left and right) arrow buttons. It is often used next to a text control for increment and decrementing a value.

Derived from

`wxControl` (p. 125)
`wxWindow` (p. 798)
`wxEvtHandler` (p. 224)
`wxObject` (p. 471)

Include files

<wx/spinbutt.h>

Window styles

wxSP_HORIZONTAL	Specifies a horizontal spin button.
wxSP_VERTICAL	Specifies a vertical spin button.
wxSP_ARROW_KEYS	The user can use arrow keys.
wxSP_WRAP	The value wraps at the minimum and maximum.

See also *window styles overview* (p. 959).

Event handling

To process input from a spin button, use one of these event handler macros to direct input to member functions that take a `wxScrollEvent` (p. 584) argument:

EVT_SPIN(id, func)	Catch all scroll commands.
EVT_SPIN_UP(id, func)	Catch up (or left) commands.
EVT_SPIN_DOWN(id, func)	Catch down (or right) commands.
EVT_COMMAND_TOP(id, func)	Catch a command to put the scroll thumb at the maximum position.
EVT_COMMAND_SCROLL(id, func)	Catch all scroll commands.
EVT_COMMAND_TOP(id, func)	Catch a command to put the scroll thumb at the maximum position.
EVT_COMMAND_BOTTOM(id, func)	Catch a command to put the scroll thumb at the maximum position.
EVT_COMMAND_LINEUP(id, func)	Catch a line up command.
EVT_COMMAND_LINEDOWN(id, func)	Catch a line down command.
EVT_COMMAND_PAGEUP(id, func)	Catch a page up command.
EVT_COMMAND_PAGEDOWN(id, func)	Catch a page down command.
EVT_COMMAND_THUMBTRACK(id, func)	Catch a thumbtrack command (continuous movement of the scroll thumb).

See also

Event handling overview (p. 939)

4.194.1 **wxSpinButton::wxSpinButton**

wxSpinButton()

Default constructor.

wxSpinButton(**wxWindow*** *parent*, **wxWindowID** *id*, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = *wxSP_HORIZONTAL*, **const wxValidator&** *validator* = *wxDefaultValidator*, **const wxString&** *name* = "spinButton")

Constructor, creating and showing a spin button.

Parameters

parent

Parent window. Must not be NULL.

id

Window identifier. A value of -1 indicates a default value.

pos

Window position. If the position (-1, -1) is specified then a default position is chosen.

size

Window size. If the default size (-1, -1) is specified then a default size is chosen.

style

Window style. See *wxSpinButton* (p. 622).

validator

Window validator.

name

Window name.

See also

wxSpinButton::Create (p. 625), *wxValidator* (p. 781)

4.194.2 **wxSpinButton::~~wxSpinButton**

void ~wxSpinButton()

Destructor, destroying the spin button.

4.194.3 **wxSpinButton::Create**

bool Create(*wxWindow* parent*, *wxWindowID id*, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = *wxSP_HORIZONTAL*, **const wxValidator&** *validator* = *wxDefaultValidator*, **const wxString&** *name* = "spinButton")

Scrollbar creation function called by the spin button constructor. See *wxSpinButton::wxSpinButton* (p. 624) for details.

4.194.4 **wxSpinButton::GetMax**

int GetMax() **const**

Returns the maximum permissible value.

[See also](#)

wxSpinButton::SetRange (p. 625)

4.194.5 **wxSpinButton::GetMin**

int GetMin() **const**

Returns the minimum permissible value.

[See also](#)

wxSpinButton::SetRange (p. 625)

4.194.6 **wxSpinButton::GetValue**

int GetValue() **const**

Returns the current spin button value.

[See also](#)

wxSpinButton::SetValue (p. 626)

4.194.7 **wxSpinButton::SetRange**

void SetRange(*int min*, *int max*)

Sets the range of the spin button.

Parameters

min

The minimum value for the spin button.

max

The maximum value for the spin button.

See also

wxSpinButton::GetMin (p. 625), *wxSpinButton::GetMax* (p. 625)

4.194.8 wxSpinButton::SetValue

void SetValue(int value)

Sets the value of the spin button.

Parameters

value

The value for the spin button.

See also

wxSpinButton::GetValue (p. 625)

4.195 wxSplitterWindow

wxSplitterWindow overview (p. 912)

This class manages up to two subwindows. The current view can be split into two programmatically (perhaps from a menu command), and unsplit either programmatically or via the *wxSplitterWindow* user interface.

Appropriate 3D shading for the Windows 95 user interface is an option.

Window styles

wxSP_3D

Draws a 3D effect border and sash.

wxSP_BORDER

Draws a thin black border around the window, and a black sash.

wxSP_NOBORDER

No border, and a black sash.

See also *window styles overview* (p. 959).

Derived from

wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/splitter.h>

4.195.1 **wxSplitterWindow::wxSplitterWindow**

wxSplitterWindow()

Default constructor.

wxSplitterWindow(*wxWindow** *parent*, *wxWindowID* *id*, *int* *x*, **const** *wxPoint&* *point* = *wxDefaultPosition*, **const** *wxSize&* *size* = *wxDefaultSize*, **long** *style*=*wxSP_3D*, **const** *wxString&* *name* = *"splitterWindow"*)

Constructor for creating the window.

Parameters

parent

The parent of the splitter window.

id

The window identifier.

pos

The window position.

size

The window size.

style

The window style. See *wxSplitterWindow* (p. 626).

name

The window name.

Remarks

After using this constructor, you must create either one or two subwindows with the splitter window as parent, and then call one of *wxSplitterWindow::Initialize* (p. 629), *wxSplitterWindow::SplitVertically* (p. 633) and *wxSplitterWindow::SplitHorizontally* (p. 633) in order to set the pane(s).

You can create two windows, with one hidden when not being shown; or you can create and delete the second pane on demand.

See also

wxSplitterWindow::Initialize (p. 629), *wxSplitterWindow::SplitVertically* (p. 633), *wxSplitterWindow::SplitHorizontally* (p. 633), *wxSplitterWindow::Create* (p. 628)

4.195.2 wxSplitterWindow::~~wxSplitterWindow

~wxSplitterWindow()

Destroys the *wxSplitterWindow* and its children.

4.195.3 wxSplitterWindow::Create

bool Create(wxWindow* parent, wxWindowID id, int x, const wxPoint& point = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style=wxSP_3D, const wxString& name = "splitterWindow")

Creation function, for two-step construction. See *wxSplitterWindow::wxSplitterWindow* (p. 627) for details.

4.195.4 wxSplitterWindow::GetMinimumPaneSize

int GetMinimumPaneSize() const

Returns the current minimum pane size (defaults to zero).

See also

wxSplitterWindow::SetMinimumPaneSize (p. 632)

4.195.5 wxSplitterWindow::GetSashPosition

int GetSashPosition()

Returns the current sash position.

See also

wxSplitterWindow::SetSashPosition (p. 631)

4.195.6 wxSplitterWindow::GetSplitMode

int GetSplitMode() const

Gets the split mode.

See also

wxSplitterWindow::SetSplitMode (p. 632), *wxSplitterWindow::SplitVertically* (p. 633), *wxSplitterWindow::SplitHorizontally* (p. 633).

4.195.7 wxSplitterWindow::GetWindow1**wxWindow* GetWindow1() const**

Returns the left/top or only pane.

4.195.8 wxSplitterWindow::GetWindow2**wxWindow* GetWindow2() const**

Returns the right/bottom pane.

4.195.9 wxSplitterWindow::Initialize**void Initialize(wxWindow* window)**

Initializes the splitter window to have one pane.

Parameters

window

The pane for the unsplit window.

Remarks

This should be called if you wish to initially view only a single pane in the splitter window.

See also

wxSplitterWindow::SplitVertically (p. 633), *wxSplitterWindow::SplitHorizontally* (p. 633)

4.195.10 wxSplitterWindow::IsSplit**bool IsSplit() const**

Returns TRUE if the window is split, FALSE otherwise.

4.195.11 wxSplitterWindow::OnDoubleClickSash**virtual void OnDoubleClickSash(int x, int y)**

Application-overrideable function called when the sash is double-clicked with the left mouse button.

Parameters

x
The x position of the mouse cursor.

y
The y position of the mouse cursor.

Remarks

The default implementation of this function calls *Unsplit* (p. 634) if the minimum pane size is zero.

See also

wxSplitterWindow::Unsplit (p. 634)

4.195.12 wxSplitterWindow::OnUnsplit**virtual void OnUnsplit(wxWindow* removed)**

Application-overrideable function called when the window is unsplit, either programmatically or using the *wxSplitterWindow* user interface.

Parameters

removed
The window being removed.

Remarks

The default implementation of this function simply hides *removed*. You may wish to delete the window.

4.195.13 wxSplitterWindow::OnSashPositionChange**virtual bool OnSashPositionChange(int newSashPosition)**

Application-overrideable function called when the sash position is changed by user. It may return FALSE to prevent the change or TRUE to allow it.

Parameters

newSashPosition

The new sash position (always positive or zero)

Remarks

The default implementation of this function verifies that the sizes of both panes of the splitter are greater than minimum pane size.

4.195.14 **wxSplitterWindow::ReplaceWindow**

bool ReplaceWindow(wxWindow * winOld, wxWindow * winNew)

This function replaces one of the windows managed by the wxSplitterWindow with another one. It is in general better to use it instead of calling `Unsplit()` and then resplitting the window back because it will provoke much less flicker (if any). It is valid to call this function whether the splitter has two windows or only one.

Both parameters should be non NULL and *winOld* must specify one of the windows managed by the splitter. If the parameters are incorrect or the window couldn't be replaced, FALSE is returned. Otherwise the function will return TRUE, but please notice that it will not delete the replaced window and you may wish to do it yourself.

See also

wxSplitterWindow::GetMinimumPaneSize (p. 628)

See also

wxSplitterWindow::Unsplit (p. 634)

wxSplitterWindow::SplitVertically (p. 633)

wxSplitterWindow::SplitHorizontally (p. 633)

4.195.15 **wxSplitterWindow::SetSashPosition**

void SetSashPosition(int position, const bool redraw = TRUE)

Sets the sash position.

Parameters

position

The sash position in pixels.

redraw

If TRUE, resizes the panes and redraws the sash and border.

Remarks

Does not currently check for an out-of-range value.

See also

wxSplitterWindow::GetSashPosition (p. 628)

4.195.16 wxSplitterWindow::SetMinimumPaneSize

void SetMinimumPaneSize(int *paneSize*)

Sets the minimum pane size.

Parameters

paneSize
Minimum pane size in pixels.

Remarks

The default minimum pane size is zero, which means that either pane can be reduced to zero by dragging the sash, thus removing one of the panes. To prevent this behaviour (and veto out-of-range sash dragging), set a minimum size, for example 20 pixels.

See also

wxSplitterWindow::GetMinimumPaneSize (p. 628)

4.195.17 wxSplitterWindow::SetSplitMode

void SetSplitMode(int *mode*)

Sets the split mode.

Parameters

mode
Can be `wxSPLIT_VERTICAL` or `wxSPLIT_HORIZONTAL`.

Remarks

Only sets the internal variable; does not update the display.

See also

wxSplitterWindow::GetSplitMode (p. 628), *wxSplitterWindow::SplitVertically* (p. 633), *wxSplitterWindow::SplitHorizontally* (p. 633).

4.195.18 wxSplitterWindow::SplitHorizontally

bool SplitHorizontally(**wxWindow*** *window1*, **wxWindow*** *window2*, **int** *sashPosition* = 0)

Initializes the top and bottom panes of the splitter window.

Parameters

window1

The top pane.

window2

The bottom pane.

sashPosition

The initial position of the sash. If this value is positive, it specifies the size of the upper pane. If it's negative, its absolute value gives the size of the lower pane. Finally, specify 0 (default) to choose the default position (half of the total window height).

Return value

TRUE if successful, FALSE otherwise (the window was already split).

Remarks

This should be called if you wish to initially view two panes. It can also be called at any subsequent time, but the application should check that the window is not currently split using *IsSplit* (p. 629).

See also

wxSplitterWindow::SplitVertically (p. 633), *wxSplitterWindow::IsSplit* (p. 629), *wxSplitterWindow::Unsplit* (p. 634)

4.195.19 wxSplitterWindow::SplitVertically

bool SplitVertically(**wxWindow*** *window1*, **wxWindow*** *window2*, **int** *sashPosition* = 0)

Initializes the left and right panes of the splitter window.

Parameters

window1

The left pane.

window2

The right pane.

sashPosition

The initial position of the sash. If this value is positive, it specifies the size of the left pane. If it's negative, its absolute value gives the size of the right pane. Finally, specify 0 (default) to choose the default position (half of the total window width).

Return value

TRUE if successful, FALSE otherwise (the window was already split).

Remarks

This should be called if you wish to initially view two panes. It can also be called at any subsequent time, but the application should check that the window is not currently split using *IsSplit* (p. 629).

See also

wxSplitterWindow::SplitHorizontally (p. 633), *wxSplitterWindow::IsSplit* (p. 629), *wxSplitterWindow::Unsplit* (p. 634).

4.195.20 wxSplitterWindow::Unsplit

bool Unsplit(wxWindow* toRemove = NULL)

Unsplits the window.

Parameters***toRemove***

The pane to remove, or NULL to remove the right or bottom pane.

Return value

TRUE if successful, FALSE otherwise (the window was not split).

Remarks

This call will not actually delete the pane being removed; it calls *OnUnsplit* (p. 630) which can be overridden for the desired behaviour. By default, the pane being removed is hidden.

See also

wxSplitterWindow::SplitHorizontally (p. 633), *wxSplitterWindow::SplitVertically* (p. 633), *wxSplitterWindow::IsSplit* (p. 629), *wxSplitterWindow::OnUnsplit* (p. 630)

4.196 wxStaticBitmap

A static bitmap control displays a bitmap.

Derived from

wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/statbmp.h>

Window styles

There are no special styles for this control.

See also *window styles overview* (p. 959).

See also

wxStaticBitmap (p. 634), *wxStaticBox* (p. 637)

Remarks

The bitmap to be displayed should have a small number of colours, such as 16, to avoid palette problems.

4.196.1 **wxStaticBitmap::wxStaticBitmap**

wxStaticBitmap()

Default constructor.

wxStaticBitmap(*wxWindow** *parent*, *wxWindowID* *id*, **const** *wxBitmap&* *label* = "",
const *wxPoint&* *pos*, **const** *wxSize&* *size* = *wxDefaultSize*, **long** *style* = 0, **const**
wxString& *name* = "staticBitmap")

Constructor, creating and showing a text control.

Parameters

parent

Parent window. Should not be NULL.

id

Control identifier. A value of -1 denotes a default value.

label

Bitmap label.

pos

Window position.

size

Window size.

style

Window style. See *wxStaticBitmap* (p. 634).

name

Window name.

See also

wxStaticBitmap::Create (p. 636)

4.196.2 **wxStaticBitmap::Create**

bool Create(wxWindow* parent, wxWindowID id, const wxBitmap& label = "", const wxPoint& pos, const wxSize& size = wxDefaultSize, long style = 0, const wxString& name = "staticBitmap")

Creation function, for two-step construction. For details see *wxStaticBitmap::wxStaticBitmap* (p. 635).

4.196.3 **wxStaticBitmap::GetBitmap**

wxBitmap& GetBitmap() const

Returns a reference to the label bitmap.

See also

wxStaticBitmap::SetBitmap (p. 636)

4.196.4 **wxStaticBitmap::SetBitmap**

virtual void SetBitmap(const wxBitmap& label)

Sets the bitmap label.

Parameters

label

The new bitmap.

See also

wxStaticBitmap::GetBitmap
wxstaticbitmapgetbitmap

4.197 wxStaticBox

A static box is a rectangle drawn around other panel items to denote a logical grouping of items.

Derived from

wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/statbox.h>

Window styles

There are no special styles for this control.

See also *window styles overview* (p. 959).

See also

wxStaticText (p. 638)

4.197.1 wxStaticBox::wxStaticBox

wxStaticBox()

Default constructor.

wxStaticBox(*wxWindow** parent, *wxWindowID* id, **const wxString&** label, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = 0, **const wxString&** name = "staticBox")

Constructor, creating and showing a static box.

Parameters

parent
Parent window. Must not be NULL.

id

Window identifier. A value of -1 indicates a default value.

label

Text to be displayed in the static box, the empty string for no label.

pos

Window position. If the position (-1, -1) is specified then a default position is chosen.

size

Checkbox size. If the size (-1, -1) is specified then a default size is chosen.

style

Window style. See *wxStaticBox* (p. 637).

name

Window name.

See also

wxStaticBox::Create (p. 638)

4.197.2 **wxStaticBox::~wxStaticBox**

void ~wxStaticBox()

Destructor, destroying the group box.

4.197.3 **wxStaticBox::Create**

bool Create(wxWindow* parent, wxWindowID id, const wxString& label, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0, const wxString& name = "staticBox")

Creates the static box for two-step construction. See *wxStaticBox::wxStaticBox* (p. 637) for further details.

4.198 **wxStaticText**

A static text control displays one or more lines of read-only text.

Derived from

wxControl (p. 125)

wxWindow (p. 798)

wxEvtHandler (p. 224)

wxObject (p. 471)

Include files

<wx/stattext.h>

Window styles

There are no special styles for this control.

See also *window styles overview* (p. 959).

See also

wxStaticBitmap (p. 634), *wxStaticBox* (p. 637)

4.198.1 **wxStaticText::wxStaticText**

wxStaticText()

Default constructor.

wxStaticText(wxWindow* parent, wxWindowID id, const wxString& label = "", const wxPoint& pos, const wxSize& size = wxDefaultSize, long style = 0, const wxString& name = "staticText")

Constructor, creating and showing a text control.

Parameters

parent

Parent window. Should not be NULL.

id

Control identifier. A value of -1 denotes a default value.

label

Text label.

pos

Window position.

size

Window size.

style

Window style. See *wxStaticText* (p. 638).

name

Window name.

See also

wxStaticText::Create (p. 640)

4.198.2 **wxStaticText::Create**

bool Create(*wxWindow* parent*, *wxWindowID id*, **const wxString& label** = "", **const wxPoint& pos**, **const wxSize& size** = *wxDefaultSize*, **long style** = 0, **const wxString& name** = "staticText")

Creation function, for two-step construction. For details see *wxStaticText::wxStaticText* (p. 639).

4.198.3 **wxStaticText::GetLabel**

wxString GetLabel() **const**

Returns the contents of the control.

4.198.4 **wxStaticText::SetLabel**

virtual void SetLabel(**const wxString& label**)

Sets the static text label.

Parameters

label

The new label to set. It may contain newline characters.

4.199 **wxStatusBar**

A status bar is a narrow window that can be placed along the bottom of a frame to give small amounts of status information. It can contain one or more fields, one or more of which can be variable length according to the size of the window.

wxWindow (p. 798)

wxEvtHandler (p. 224)

wxObject (p. 471)

Derived from

wxWindow (p. 798)

wxEvtHandler (p. 224)

wxObject (p. 471)

Include files

<wx/statusbr.h>

Window styles

wxSB_SIZEGRIP On Windows 95, displays a gripper at right-hand side of the status bar.

See also *window styles overview* (p. 959).

Remarks

It is possible to create controls and other windows on the status bar. Position these windows from an **OnSize** event handler.

See also

wxFrame (p. 275)

4.199.1 **wxStatusBar::wxStatusBar**

wxStatusBar()

Default constructor.

wxStatusBar(**wxWindow*** *parent*, **wxWindowID** *id*, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = 0, **const wxString&** *name* = "statusBar")

Constructor, creating the window.

Parameters

parent

The window parent, usually a frame.

id

The window identifier. It may take a value of -1 to indicate a default value.

pos

The window position. A value of (-1, -1) indicates a default position, chosen by either the windowing system or wxWindows, depending on platform.

size

The window size. A value of (-1, -1) indicates a default size, chosen by either the windowing system or `wxWindows`, depending on platform.

style

The window style. See `wxStatusBar` (p. 640).

name

The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

See also

`wxStatusBar::Create` (p. 642)

4.199.2 `wxStatusBar::~~wxStatusBar`

void `~wxStatusBar()`

Destructor.

4.199.3 `wxStatusBar::Create`

bool `Create(wxWindow* parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0, const wxString& name = "statusBar")`

Creates the window, for two-step construction.

See `wxStatusBar::wxStatusBar` (p. 641) for details.

4.199.4 `wxStatusBar::GetFieldRect`

virtual bool `GetFieldRect(int i, wxRect& rect) const`

Returns the size and position of a field's internal bounding rectangle.

Parameters

i

The field in question.

rect

The rectangle values are placed in this variable.

Return value

TRUE if the field index is valid, FALSE otherwise.

See also

wxRect (p. 546)

4.199.5 wxStatusBar::GetFieldsCount

int GetFieldsCount() const

Returns the number of fields in the status bar.

4.199.6 wxStatusBar::GetStatusText

virtual wxString GetStatusText(int *ir* = 0) const

Returns the string associated with a status bar field.

Parameters

i
The number of the status field to retrieve, starting from zero.

Return value

The status field string if the field is valid, otherwise the empty string.

See also

wxStatusBar::SetStatusText (p. 645)

4.199.7 wxStatusBar::DrawField

virtual void DrawField(wxDC& *dc*, int *i*)

Draws a field, including shaded borders and text.

Parameters

dc
The device context to draw onto.

i
The field to be drawn.

See also

wxStatusBar::DrawFieldText (p. 644)

4.199.8 wxStatusBar::DrawFieldText

virtual void DrawFieldText(wxDC& *dc*, int *i*)

Draws a field's text.

Parameters

dc

The device context to draw onto.

i

The field whose text is to be drawn.

See also

wxStatusBar::DrawField (p. 643)

4.199.9 wxStatusBar::InitColours

virtual void InitColours()

Sets up the background colour and shading pens using suitable system colours (Windows) or tasteful shades of grey (other platforms).

Remarks

This function is called when the window is created, and also from *wxStatusBar::OnSysColourChanged* (p. 644) on Windows.

See also

wxStatusBar::OnSysColourChanged (p. 644)

4.199.10 wxStatusBar::OnSysColourChanged

void OnSysColourChanged(wxSysColourChangedEvent& *event*)

Handles a system colour change by calling *wxStatusBar::InitColours* (p. 644), and refreshes the window.

Parameters

event

The colour change event.

See also

wxStatusBar::InitColours (p. 644)

4.199.11 wxStatusBar::SetFieldsCount

virtual void SetFieldsCount(int *number* = 1, int* *widths* = NULL)

Sets the number of fields, and optionally the field widths.

Parameters

number

The number of fields.

widths

An array of *n* integers, each of which is a status field width in pixels. A value of -1 indicates that the field is variable width; at least one field must be -1.

4.199.12 wxStatusBar::SetStatusText

virtual void SetStatusText(const wxString& *text*, int *i* = 0)

Sets the text for one field.

Parameters

text

The text to be set. Use an empty string ("") to clear the field.

i

The field to set, starting from zero.

See also

wxStatusBar::GetStatusText (p. 643), *wxFrame::SetStatusText* (p. 285)

4.199.13 wxStatusBar::SetStatusWidths

virtual void SetStatusWidths(int *n*, int **widths*)

Sets the widths of the fields in the status line.

Parameters

n

The number of fields in the status bar.

widths

Must contain an array of n integers, each of which is a status field width in pixels. A value of -1 indicates that the field is variable width; at least one field must be -1. You should delete this array after calling **SetStatusWidths**.

Remarks

The widths of the variable fields are calculated from the total width of all fields, minus the sum of widths of the non-variable fields, divided by the number of variable fields.

See also

wxStatusBar::SetFieldsCount (p. 645), *wxFrame::SetStatusWidths* (p. 286)

4.200 wxStreamBase

Derived from

None

Include files

<wx/stream.h>

See also

wxStreamBuffer (p. 648)

4.200.1 wxStreamBase::wxStreamBase

wxStreamBase()

Creates a dummy stream object. It doesn't do anything.

4.200.2 wxStreamBase::~~wxStreamBase

~wxStreamBase()

Destructor.

4.200.3 wxStreamBase::LastError

wxStreamError LastError() const

This function returns the last error. **wxStream_NOERROR** No error occurred.

wxStream_EOF An End-Of-File occurred.

wxStream_WRITE_ERR A generic error occurred on the last write call.

wxStream_READ_ERR A generic error occurred on the last read call.

4.200.4 **wxStreamBase::OnSysRead**

size_t OnSysRead(void* *buffer*, size_t *bufsize*)

Internal function. It is called when the stream buffer needs a buffer of the specified size. It should return the size that was actually read.

4.200.5 **wxStreamBase::OnSysSeek**

off_t OnSysSeek(off_t *pos*, wxSeekMode *mode*)

Internal function. It is called when the stream buffer needs to change the current position in the stream. See *wxStreamBuffer::Seek* (p. 651)

4.200.6 **wxStreamBase::OnSysTell**

off_t OnSysTell() const

Internal function. Is called when the stream buffer needs to know the real position in the stream.

4.200.7 **wxStreamBase::OnSysWrite**

size_t OnSysWrite(void **buffer*, size_t *bufsize*)

See *OnSysRead* (p. 647).

4.200.8 **wxStreamBase::StreamSize**

size_t StreamSize() const

This function returns the size of the stream. For example, for a file it is the size of the file).

Warning

There are streams which do not have size by definition, such as socket streams. In that cases, *StreamSize* returns an invalid size represented by

~(size_t)0

4.201

wxStreamBuffer

Derived from

None

Include files

<wx/stream.h>

See also

wxStreamBase (p. 646)

4.201.1 **wxStreamBuffer::wxStreamBuffer**

wxStreamBuffer(**wxStreamBase**& *stream*, **BufMode** *mode*)

Constructor, creates a new stream buffer using *stream* as a parent stream and *mode* as the IO mode. *mode* can be: `wxStreamBuffer::read`, `wxStreamBuffer::write`, `wxStreamBuffer::read_write`. One stream can have many stream buffers but only one is used internally to pass IO call (e.g. `wxInputStream::Read()` -> `wxStreamBuffer::Read()`). But you can call directly `wxStreamBuffer::Read` without any problems.

Warning

All errors and messages linked to the stream are stored in the stream object.

```
streambuffer.Read(...);
streambuffer2.Read(...); /* This one erases previous error messages
set by
```

```
``streambuffer'' */
```

wxStreamBuffer(**BufMode** *mode*)

Constructor, creates a new empty stream buffer which won't flush any data to a stream. *mode* specifies the type of the buffer (read, write, read_write). This stream buffer has the advantage to be stream independent and to work only on memory buffers but it is still compatible with the rest of the `wxStream` classes. You can write, read to this special stream and it will grow (if it is allowed by the user) its internal buffer. Briefly, it has all functionality of a "normal" stream.

Warning

The "read_write" mode may not work: it isn't completely finished. You can create "memory" streams by this way:


```
wxStreamBuffer *sb = new wxStreamBuffer(wxStreamBuffer::read)
wxInputStream *input = new wxInputStream(sb);

sb->Fixed(FALSE); // It can change the size of the buffer.

// input is now a read-only memory stream.
```

But you should take care when destroying the stream buffer yourself.

wxStreamBuffer(const wxStreamBuffer&buffer)

Constructor. It initializes the stream buffer with the data of the specified stream buffer. The new stream buffer is nearly exactly the same as the original: it has the same attributes, the same size, the same position, shares the same internal buffer. The interesting point is that they can differ in the future but the root is the same.

Warning

The fact that the two stream buffers shared the same buffer could generate segmentation violation if the parent is destroyed and the children continues operating. It is advised to use this feature only in very local area of the program.

See also

wxStreamBuffer::SetBufferIO (p. 652)

4.201.2 wxStreamBuffer::~~wxStreamBuffer

wxStreamBuffer(~wxStreamBuffer)

Destructor. It finalizes all IO calls and frees all internal buffers if necessary. In the case of a children stream buffer, the internal buffer isn't freed, this is the job of the parent. The "Write-Back" buffer is freed.

4.201.3 wxStreamBuffer::Read

size_t Read(void *buffer, size_t size)

Reads a block of the specified *size* and stores datas in *buffer*. This function uses also the "Write-Back" buffer: in the case there are datas waiting in this buffer, they are used before anything else. After that, if there are still datas to be read, the stream is read and the stream buffer position is incremented.

Return value

It returns the real read size. If returned size is different of the specified *size*, an error occurred and should be tested using *LastError* (p. 646).

See also

wxStreamBuffer::WriteBack (p. 650)

size_t Read(wxStreamBuffer *buffer)

Reads a *buffer*. The function returns when *buffer* is full or when there aren't datas anymore in the current buffer.

4.201.4 wxStreamBuffer::Write

size_t Write(const void *buffer, size_t size)

Writes a block of the specified *size* using datas of *buffer*. The datas are cached in a buffer before being sent in one block to the stream.

size_t Write(wxStreamBuffer *buffer)

See *Read* (p. 649).

4.201.5 wxStreamBuffer::WriteBack

size_t WriteBack(const char* buffer, size_t size)

This function is only useful in *read* mode. It is the manager of the "Write-Back" buffer. This buffer acts like a temporary buffer where datas which has to be read during the next read IO call are put. This is useful when you get a big block of data which you didn't want to read: you can replace them at the top of the input queue by this way.

Return value

Returns the amount of bytes saved in the Write-Back buffer.

size_t WriteBack(char c)

This function acts like the previous one except that it takes only one character: it is sometimes shorter to use than the generic function.

4.201.6 wxStreamBuffer::GetChar

char GetChar()

Gets a single char from the stream buffer. It acts like the *Read* call.

Problem

You aren't directly notified if an error occurred during the IO call.

See also

wxStreamBuffer::Read (p. 649)

4.201.7 wxStreamBuffer::PutChar

void PutChar(char c)

Puts a single char to the stream buffer.

Problem

You aren't directly notified if an error occurred during the IO call.

See also

wxStreamBuffer::Read (p. 650)

4.201.8 wxStreamBuffer::Tell

off_t Tell() const

Gets the current position in the stream. This position is calculated from the *real* position in the stream and from the internal buffer position: so it gives you the position in the *real* stream counted from the start of the stream.

Return value

Returns the current position in the stream if possible, `wxInvalidOffset` in the other case.

4.201.9 wxStreamBuffer::Seek

off_t Seek(off_t pos, wxSeekMode mode)

Changes the current position.

mode may be one of the following:

wxFromStart	The position is counted from the start of the stream.
wxFromCurrent	The position is counted from the current position of the stream.
wxFromEnd	The position is counted from the end of the stream.

Return value

Upon successful completion, it returns the new offset as measured in bytes from the

beginning of the stream. Otherwise, it returns `wxInvalidOffset`.

4.201.10 **`wxStreamBuffer::ResetBuffer`**

`void ResetBuffer()`

Resets to the initial state variables concerning the buffer.

4.201.11 **`wxStreamBuffer::SetBufferIO`**

`void SetBufferIO(char* buffer_start, char* buffer_end)`

Specifies which pointers to use for stream buffering. You need to pass a pointer on the start of the buffer end and another on the end. The object will use this buffer to cache stream data. It may be used also as a source/destination buffer when you create an empty stream buffer (See `wxStreamBuffer::wxStreamBuffer` (p. 648)).

Remarks

When you use this function, you'll have to destroy the IO buffers yourself after the stream buffer is destroyed or don't use it anymore. In the case you use it with an empty buffer, the stream buffer will not grow it when it is full.

See also

wxStreamBuffer constructor (p. 648)

wxStreamBuffer::Fixed (p. 653)

wxStreamBuffer::Flushable (p. 654)

`void SetBufferIO(size_t bufsize)`

Destroys or invalidates the previous IO buffer and allocates a new one of the specified size.

Warning

All previous pointers aren't valid anymore.

Remark

The created IO buffer is growable by the object.

See also

wxStreamBuffer::Fixed (p. 653)

wxStreamBuffer::Flushable (p. 654)

4.201.12 wxStreamBuffer::GetBufferStart**char * GetBufferStart() const**

Returns a pointer on the start of the stream buffer.

4.201.13 wxStreamBuffer::GetBufferEnd**char * GetBufferEnd() const**

Returns a pointer on the end of the stream buffer.

4.201.14 wxStreamBuffer::GetBufferPos**char * GetBufferPos() const**

Returns a pointer on the current position of the stream buffer.

4.201.15 wxStreamBuffer::GetIntPosition**off_t GetIntPosition() const**

Returns the current position (counted in bytes) in the stream buffer.

4.201.16 wxStreamBuffer::SetIntPosition**void SetIntPosition()**

Sets the current position (in bytes) in the stream buffer.

Warning

Since it is a very low-level function, there is no check on the position: specify an invalid position can induce unexpected results.

4.201.17 wxStreamBuffer::GetLastAccess**size_t GetLastAccess() const**

Returns the amount of bytes read during the last IO call to the parent stream.

4.201.18 wxStreamBuffer::Fixed**void Fixed(bool *fixed*)**

Toggles the fixed flag. Usually this flag is toggled at the same time as *flushable*. This flag allows (when it has the FALSE value) or forbids (when it has the TRUE value) the stream buffer to resize dynamically the IO buffer.

See also

wxStreamBuffer::SetBufferIO (p. 652)

4.201.19 wxStreamBuffer::Flushable

void Flushable(bool *flushable*)

Toggles the flushable flag. If *flushable* is disabled, no datas are sent to the parent stream.

4.201.20 wxStreamBuffer::FlushBuffer

bool FlushBuffer()

Flushes the IO buffer.

4.201.21 wxStreamBuffer::FillBuffer

bool FillBuffer()

Fill the IO buffer.

4.201.22 wxStreamBuffer::GetDataLeft

size_t GetDataLeft()

Returns the amount of available datas in the buffer.

4.201.23 wxStreamBuffer::Stream

wxStreamBase* Stream()

Returns the parent stream of the stream buffer.

4.202 wxString

wxString is a class representing a character string. Please see the *wxString overview* (p. 899) for more information about it. As explained there, *wxString* implements about 90% of methods of the `std::string` class (iterators are not supported, nor all methods which

use them). These standard functions are not documented in this manual so please see the STL documentation. The behaviour of all these functions is identical to the behaviour described there.

Derived from

None

Include files

<wx/string.h>

Predefined objects

Objects:

wxEmptyString

See also

Overview (p. 899)

4.202.1 Constructors and assignment operators

A string may be constructed either from a C string, (some number of copies of) a single character or a wide (UNICODE) string. For all constructors (except the default which creates an empty string) there is also a corresponding assignment operator.

wxString (p. 661)
operator = (p. 672)
~wxString (p. 662)

4.202.2 String length

These functions return the string length and check whether the string is empty or empty it.

Len (p. 668)
IsEmpty (p. 666)
operator! (p. 672)
Empty (p. 664)
Clear (p. 663)

4.202.3 Character access

Many functions in this section take a character index in the string. As with C strings and/or arrays, the indices start from 0, so the first character of a string is `string[0]`. Attempt to access a character beyond the end of the string (which may be even 0 if the string is empty) will provoke an assert failure in *debug build* (p. 927), but no checks are done in release builds.

This section also contains both implicit and explicit conversions to C style strings. Although implicit conversion is quite convenient, it is advised to use explicit `c_str()` (p. 663) method for the sake of clarity. Also see *overview* (p. 900) for the cases where it is necessary to use it.

GetChar (p. 665)
GetWritableChar (p. 665)
SetChar (p. 670)
Last (p. 667)
operator [] (p. 673)
c_str (p. 663)
*operator const char** (p. 674)

4.202.4 Concatenation

Anything may be concatenated (appended to) with a string. However, you can't append something to a C string (including literal constants), so to do this it should be converted to a `wxString` first.

operator << (p. 673)
operator += (p. 673)
operator + (p. 672)
Append (p. 662)
Prepend (p. 669)

4.202.5 Comparison

The default comparison function *Cmp* (p. 663) is case-sensitive and so is the default version of *IsSameAs* (p. 667). For case insensitive comparisons you should use *CmpNoCase* (p. 664) or give a second parameter to *IsSameAs*. This last function is may be more convenient if only equality of the strings matters because it returns a boolean true value if the strings are the same and not 0 (which is usually FALSE in C) as *Cmp* does.

Matches (p. 668) is a poor man's regular expression matcher: it only understands '*' and '?' metacharacters in the sense of DOS command line interpreter.

Cmp (p. 663)
CmpNoCase (p. 664)
IsSameAs (p. 667)
Matches (p. 668)

4.202.6 Substring extraction

These functions allow to extract substring from this string. All of them don't modify the original string and return a new string containing the extracted substring.

Mid (p. 669)
operator() (p. 673)
Left (p. 667)
Right (p. 670)
BeforeFirst (p. 663)
BeforeLast (p. 663)
AfterFirst (p. 662)
AfterLast (p. 663)

4.202.7 Case conversion

The *MakeXXX()* variants modify the string in place, while the other functions return a new string which contains the original text converted to the upper or lower case and leave the original string unchanged.

MakeUpper (p. 668)
Upper (p. 671)
MakeLower (p. 668)
Lower (p. 668)

4.202.8 Searching and replacing

These functions replace the standard *strchr()* and *strstr()* functions.

Find (p. 664)
Replace (p. 670)

4.202.9 Writing values into the string

Both formatted versions (*Printf* (p. 669)) and stream-like insertion operators exist (for basic types only).

Printf (p. 669)
PrintfV (p. 669)
operator << (p. 673)

4.202.10 Memory management

These are "advanced" functions and they will be needed quite rarely. *Alloc* (p. 662) and *Shrink* (p. 670) are only interesting for optimization purposes. *GetWriteBuf* (p. 666) may be very useful when working with some external API which requires the caller to provide

a writable buffer, but extreme care should be taken when using it: before performing any other operation on the string *UngetWriteBuf* (p. 671) **must** be called!

Alloc (p. 662)
Shrink (p. 670)
GetWriteBuf (p. 666)
UngetWriteBuf (p. 671)

4.202.11 Miscellaneous

Other string functions.

Trim (p. 671)
Pad (p. 669)
Truncate (p. 671)

4.202.12 wxWindows 1.xx compatibility functions

These functions are deprecated, please consider using new wxWindows 2.0 functions instead of them (or, even better, `std::string` compatible variants).

SubString (p. 671)
sprintf (p. 671)
CompareTo (p. 664)
Length (p. 668)
Freq (p. 665)
LowerCase (p. 668)
UpperCase (p. 672)
Strip (p. 671)
Index (p. 666)
Remove (p. 669)
First (p. 665)
Last (p. 667)
Contains (p. 664)
IsNull (p. 666)
IsAscii (p. 666)
IsNumber (p. 667)
IsWord (p. 667)

4.202.13 `std::string` compatibility functions

The supported functions are only listed here, please see any STL reference for their documentation.

```
// take nLen chars starting at nPos
wxString(const wxString& str, size_t nPos, size_t nLen);
// take all characters from pStart to pEnd (poor man's iterators)
wxString(const void *pStart, const void *pEnd);
```

```

// lib.string.capacity
// return the length of the string
size_t size() const;
// return the length of the string
size_t length() const;
// return the maximum size of the string
size_t max_size() const;
// resize the string, filling the space with c if c != 0
void resize(size_t nSize, char ch = '\0');
// delete the contents of the string
void clear();
// returns true if the string is empty
bool empty() const;

// lib.string.access
// return the character at position n
char at(size_t n) const;
// returns the writable character at position n
char& at(size_t n);

// lib.string.modifiers
// append a string
wxString& append(const wxString& str);
// append elements str[pos], ..., str[pos+n]
wxString& append(const wxString& str, size_t pos, size_t n);
// append first n (or all if n == npos) characters of sz
wxString& append(const char *sz, size_t n = npos);

// append n copies of ch
wxString& append(size_t n, char ch);

// same as `this_string = str'
wxString& assign(const wxString& str);
// same as ` = str[pos..pos + n]
wxString& assign(const wxString& str, size_t pos, size_t n);
// same as ` = first n (or all if n == npos) characters of sz'
wxString& assign(const char *sz, size_t n = npos);
// same as ` = n copies of ch'
wxString& assign(size_t n, char ch);

// insert another string
wxString& insert(size_t nPos, const wxString& str);
// insert n chars of str starting at nStart (in str)
wxString& insert(size_t nPos, const wxString& str, size_t nStart,
size_t n);

// insert first n (or all if n == npos) characters of sz
wxString& insert(size_t nPos, const char *sz, size_t n = npos);
// insert n copies of ch
wxString& insert(size_t nPos, size_t n, char ch);

// delete characters from nStart to nStart + nLen
wxString& erase(size_t nStart = 0, size_t nLen = npos);

// replaces the substring of length nLen starting at nStart
wxString& replace(size_t nStart, size_t nLen, const char* sz);
// replaces the substring with nCount copies of ch
wxString& replace(size_t nStart, size_t nLen, size_t nCount, char ch);
// replaces a substring with another substring
wxString& replace(size_t nStart, size_t nLen,

```

```

        const wxString& str, size_t nStart2, size_t nLen2);
// replaces the substring with first nCount chars of sz
wxString& replace(size_t nStart, size_t nLen,
                 const char* sz, size_t nCount);

// swap two strings
void swap(wxString& str);

// All find() functions take the nStart argument which specifies the
// position to start the search on, the default value is 0. All
functions
// return npos if there were no match.

// find a substring
size_t find(const wxString& str, size_t nStart = 0) const;

// find first n characters of sz
size_t find(const char* sz, size_t nStart = 0, size_t n = npos) const;

// find the first occurrence of character ch after nStart
size_t find(char ch, size_t nStart = 0) const;

// rfind() family is exactly like find() but works right to left

// as find, but from the end
size_t rfind(const wxString& str, size_t nStart = npos) const;

// as find, but from the end
size_t rfind(const char* sz, size_t nStart = npos,
             size_t n = npos) const;
// as find, but from the end
size_t rfind(char ch, size_t nStart = npos) const;

// find first/last occurrence of any character in the set

//
size_t find_first_of(const wxString& str, size_t nStart = 0) const;
//
size_t find_first_of(const char* sz, size_t nStart = 0) const;
// same as find(char, size_t)
size_t find_first_of(char c, size_t nStart = 0) const;
//
size_t find_last_of (const wxString& str, size_t nStart = npos) const;
//
size_t find_last_of (const char* s, size_t nStart = npos) const;
// same as rfind(char, size_t)
size_t find_last_of (char c, size_t nStart = npos) const;

// find first/last occurrence of any character not in the set

//
size_t find_first_not_of(const wxString& str, size_t nStart = 0)
const;
//
size_t find_first_not_of(const char* s, size_t nStart = 0) const;
//
size_t find_first_not_of(char ch, size_t nStart = 0) const;
//
size_t find_last_not_of(const wxString& str, size_t nStart=npos)
const;
//

```

```

size_t find_last_not_of(const char* s, size_t nStart = npos) const;
//
size_t find_last_not_of(char ch, size_t nStart = npos) const;

// All compare functions return a negative, zero or positive value
// if the [sub]string is less, equal or greater than the compare()
argument.

// just like strcmp()
int compare(const wxString& str) const;
// comparison with a substring
int compare(size_t nStart, size_t nLen, const wxString& str) const;
// comparison of 2 substrings
int compare(size_t nStart, size_t nLen,
            const wxString& str, size_t nStart2, size_t nLen2) const;
// just like strcmp()
int compare(const char* sz) const;
// substring comparison with first nCount characters of sz
int compare(size_t nStart, size_t nLen,
            const char* sz, size_t nCount = npos) const;

// substring extraction
wxString substr(size_t nStart = 0, size_t nLen = npos) const;

```

4.202.14 **wxString::wxString**

wxString()

Default constructor.

wxString(const wxString& x)

Copy constructor.

wxString(char ch, size_t n = 1)

Constructs a string of *n* copies of character *ch*.

wxString(const char* psz, size_t nLength = wxSTRING_MAXLEN)

Takes first *nLength* characters from the C string *psz*. The default value of *wxSTRING_MAXLEN* means take all the string.

wxString(const unsigned char* psz, size_t nLength = wxSTRING_MAXLEN)

For compilers using unsigned char: takes first *nLength* characters from the C string *psz*. The default value of *wxSTRING_MAXLEN* means take all the string.

wxString(const wchar_t* psz)

Constructs a string from the wide (UNICODE) string.

4.202.15 wxString::~~wxString**~wxString()**

String destructor. Note that this is not virtual, so wxString must not be inherited from.

4.202.16 wxString::Alloc**void Alloc(size_t nLen)**

Preallocate enough space for wxString to store *nLen* characters. This function may be used to increase speed when the string is constructed by repeated concatenation as in

```
// delete all vowels from the string
wxString DeleteAllVowels(const wxString& original)
{
    wxString result;

    size_t len = original.length();

    result.Alloc(len);

    for ( size_t n = 0; n < len; n++ )
    {
        if ( strchr("aeuio", tolower(original[n])) == NULL )
            result += original[n];
    }

    return result;
}
```

because it will avoid the need of reallocating string memory many times (in case of long strings). Note that it does not set the maximal length of a string - it will still expand if more than *nLen* characters are stored in it. Also, it does not truncate the existing string (use *Truncate()* (p. 671) for this) even if its current length is greater than *nLen*

4.202.17 wxString::Append**wxString& Append(const char* psz)**

Concatenates *psz* to this string, returning a reference to it.

wxString& Append(char ch, int count = 1)

Concatenates character *ch* to this string, *count* times, returning a reference to it.

4.202.18 wxString::AfterFirst

wxString AfterFirst(char *ch*) const

Gets all the characters after the first occurrence of *ch*. Returns the empty string if *ch* is not found.

4.202.19 wxString::AfterLast**wxString AfterLast(char *ch*) const**

Gets all the characters after the last occurrence of *ch*. Returns the whole string if *ch* is not found.

4.202.20 wxString::BeforeFirst**wxString BeforeFirst(char *ch*) const**

Gets all characters before the first occurrence of *ch*. Returns the whole string if *ch* is not found.

4.202.21 wxString::BeforeLast**wxString BeforeLast(char *ch*) const**

Gets all characters before the last occurrence of *ch*. Returns the empty string if *ch* is not found.

4.202.22 wxString::c_str**const char * c_str() const**

Returns a pointer to the string data.

4.202.23 wxString::Clear**void Clear()**

Empties the string and frees memory occupied by it.

See also: *Empty* (p. 664)

4.202.24 wxString::Cmp**int Cmp(const char* *psz*) const**

Case-sensitive comparison.

Returns a positive value if the string is greater than the argument, zero if it is equal to it or negative value if it is less than argument (same semantics as the standard *strcmp()* function).

See also *CmpNoCase* (p. 664), *IsSameAs* (p. 667).

4.202.25 wxString::CmpNoCase

int CmpNoCase(const char* psz) const

Case-insensitive comparison.

Returns a positive value if the string is greater than the argument, zero if it is equal to it or negative value if it is less than argument (same semantics as the standard *strcmp()* function).

See also *Cmp* (p. 663), *IsSameAs* (p. 667).

4.202.26 wxString::CompareTo

```
#define NO_POS ((int)(-1)) // undefined position
enum caseCompare {exact, ignoreCase};
```

int CompareTo(const char* psz, caseCompare cmp = exact) const

Case-sensitive comparison. Returns 0 if equal, 1 if greater or -1 if less.

4.202.27 wxString::Contains

bool Contains(const wxString& str) const

Returns 1 if target appears anywhere in wxString; else 0.

4.202.28 wxString::Empty

void Empty()

Makes the string empty, but doesn't free memory occupied by the string.

See also: *Clear()* (p. 663).

4.202.29 wxString::Find

int Find(char *ch*, bool *fromEnd* = FALSE) const

Searches for the given character. Returns the starting index, or -1 if not found.

int Find(const char* *sz*) const

Searches for the given string. Returns the starting index, or -1 if not found.

4.202.30 wxString::First

size_t First(char *c*)

size_t First(const char* *psz*) const

size_t First(const wxString& *str*) const

size_t First(const char *ch*) const

Returns the first occurrence of the item.

4.202.31 wxString::Freq

int Freq(char *ch*) const

Returns the number of occurrences of *ch* in the string.

4.202.32 wxString::GetChar

char GetChar(size_t *n*) const

Returns the character at position *n* (read-only).

4.202.33 wxString::GetData

const char* GetData() const

wxWindows compatibility conversion. Returns a constant pointer to the data in the string.

4.202.34 wxString::GetWritableChar

char& GetWritableChar(size_t *n*)

Returns a reference to the character at position *n*.

4.202.35 wxString::GetWriteBuf**char* GetWriteBuf(size_t len)**

Returns a writable buffer of at least *len* bytes.

Call *wxString::UngetWriteBuf* (p. 671) as soon as possible to put the string back into a reasonable state.

4.202.36 wxString::Index**size_t Index(char ch, int startpos = 0) const**

Same as *wxString::Find* (p. 664).

size_t Index(const char* sz) const

Same as *wxString::Find* (p. 664).

size_t Index(const char* sz, bool caseSensitive = TRUE, bool fromEnd = FALSE) const

Search the element in the array, starting from either side.

If *fromEnd* is TRUE, reverse search direction.

If **caseSensitive**, comparison is case sensitive (the default).

Returns the index of the first item matched, or NOT_FOUND.

4.202.37 wxString::IsAscii**bool IsAscii() const**

Returns TRUE if the string is ASCII.

4.202.38 wxString::IsEmpty**bool IsEmpty() const**

Returns TRUE if the string is NULL.

4.202.39 wxString::IsNull**bool IsNull() const**

Returns TRUE if the string is NULL (same as `IsEmpty`).

4.202.40 `wxString::IsNumber`

`bool IsNumber() const`

Returns TRUE if the string is a number.

4.202.41 `wxString::IsSameAs`

`bool IsSameAs(const char* psz, bool caseSensitive = TRUE) const`

Test for string equality, case-sensitive (default) or not.

`caseSensitive` is TRUE by default (case matters).

Returns TRUE if strings are equal, FALSE otherwise.

See also *Cmp* (p. 663), *CmpNoCase* (p. 664).

4.202.42 `wxString::IsWord`

`bool IsWord() const`

Returns TRUE if the string is a word. TODO: what's the definition of a word?

4.202.43 `wxString::Last`

`char Last() const`

Returns the last character.

`char& Last()`

Returns a reference to the last character (writable).

4.202.44 `wxString::Left`

`wxString Left(size_t count) const`

Returns the first *count* characters.

`wxString Left(char ch) const`

Returns all characters before the first occurrence of *ch*. Returns the whole string if *ch* is

not found.

4.202.45 wxString::Len

size_t Len() const

Returns the length of the string.

4.202.46 wxString::Length

size_t Length() const

Returns the length of the string (same as Len).

4.202.47 wxString::Lower

wxString Lower() const

Returns this string converted to the lower case.

4.202.48 wxString::LowerCase

void LowerCase()

Same as MakeLower.

4.202.49 wxString::MakeLower

void MakeLower()

Converts all characters to lower case.

4.202.50 wxString::MakeUpper

void MakeUpper()

Converts all characters to upper case.

4.202.51 wxString::Matches

bool Matches(const char* szMask) const

Returns TRUE if the string contents matches a mask containing '*' and '?'.

4.202.52 wxString::Mid**wxString Mid(size_t first, size_t count = wxSTRING_MAXLEN) const**

Returns a substring starting at *first*, with length *count*, or the rest of the string if *count* is the default value.

4.202.53 wxString::Pad**wxString& Pad(size_t count, char pad = ' ', bool fromRight = TRUE)**

Adds *count* copies of *pad* to the beginning, or to the end of the string (the default).

Removes spaces from the left or from the right (default).

4.202.54 wxString::Prepend**wxString& Prepend(const wxString& str)**

Prepends *str* to this string, returning a reference to this string.

4.202.55 wxString::Printf**int Printf(const char* pszFormat, ...)**

Similar to the standard function *sprintf()*. Returns the number of characters written, or an integer less than zero on error.

NB: This function will use a safe version of *vsprintf()* (usually called *vsnprintf()*) whenever available to always allocate the buffer of correct size. Unfortunately, this function is not available on all platforms and the dangerous *vsprintf()* will be used then which may lead to buffer overflows.

4.202.56 wxString::PrintfV**int PrintfV(const char* pszFormat, va_list argPtr)**

Similar to *vprintf*. Returns the number of characters written, or an integer less than zero on error.

4.202.57 wxString::Remove**wxString& Remove(size_t pos)**

Same as `Truncate`. Removes the portion from *pos* to the end of the string.

wxString& Remove(size_t pos, size_t len)

Removes the last *len* characters from the string, starting at *pos*.

4.202.58 wxString::RemoveLast

wxString& RemoveLast()

Removes the last character.

4.202.59 wxString::Replace

size_t Replace(const char* szOld, const char* szNew, bool replaceAll = TRUE)

Replace first (or all) occurrences of substring with another one.

replaceAll: global replace (default), or only the first occurrence.

Returns the number of replacements made.

4.202.60 wxString::Right

wxString Right(size_t count) const

Returns the last *count* characters.

wxString Right(char ch) const

Returns all characters after the last occurrence of *ch*. Returns the whole string if *ch* is not found.

4.202.61 wxString::SetChar

void SetChar(size_t n, char ch)

Sets the character at position *n*.

4.202.62 wxString::Shrink

void Shrink()

Minimizes the string's memory. This can be useful after a call to *Alloc()* (p. 662) if too

much memory were preallocated.

4.202.63 wxString::sprintf

void sprintf(const char* *fmt*)

The same as Printf.

4.202.64 wxString::Strip

```
enum stripType {leading = 0x1, trailing = 0x2, both = 0x3};
```

wxString Strip(stripType *s* = *trailing*) const

Strip characters at the front and/or end. The same as Trim except that it doesn't change this string.

4.202.65 wxString::SubString

wxString SubString(size_t *to*, size_t *from*) const

Same as *Mid* (p. 669).

4.202.66 wxString::Trim

wxString& Trim(bool *fromRight* = *TRUE*)

Removes spaces from the left or from the right (default).

4.202.67 wxString::Truncate

wxString& Truncate(size_t *len*)

Truncate the string to the given length.

4.202.68 wxString::UngetWriteBuf

void UngetWriteBuf()

Puts the string back into a reasonable state, after *wxString::GetWriteBuf* (p. 666) was called.

4.202.69 wxString::Upper

wxString Upper() const

Returns this string converted to upper case.

4.202.70 wxString::UpperCase**void UpperCase()**

The same as MakeUpper.

4.202.71 wxString::operator!**bool operator!() const**

Empty string is FALSE, so !string will only return TRUE if the string is empty. This allows the tests for NULLness of a *const char ** pointer and emptiness of the string to look the same in the code and makes it easier to port old code to wxString.

See also *IsEmpty()* (p. 666).

4.202.72 wxString::operator =

wxString& operator =(const wxString& str)

wxString& operator =(const char* psz)

wxString& operator =(char c)

wxString& operator =(const unsigned char* psz)

wxString& operator =(const wchar_t* pwz)

Assignment: the effect of each operation is the same as for the corresponding constructor (see *wxString constructors* (p. 661)).

4.202.73 operator wxString::+

Concatenation: all these operators return a new string equal to the sum of the operands.

wxString operator +(const wxString& x, const wxString& y)

wxString operator +(const wxString& x, const char* y)

wxString operator +(const wxString& x, char y)

wxString operator +(const char* x, const wxString& y)

4.202.74 wxString::operator +=

void operator +=(const wxString& str)

void operator +=(const char* psz)

void operator +=(char c)

Concatenation in place: the argument is appended to the string.

4.202.75 wxString::operator []

char& operator [] (size_t i)

char operator [] (size_t i)

char operator [] (int i)

Element extraction.

4.202.76 wxString::operator ()

wxString operator () (size_t start, size_t len)

Same as Mid (substring extraction).

4.202.77 wxString::operator <<

wxString& operator <<(const wxString& str)

wxString& operator <<(const char* psz)

wxString& operator <<(char ch)

Same as +=.

wxString& operator <<(int i)

wxString& operator <<(float f)

wxString& operator <<(double d)

These functions work as C++ stream insertion operators: they insert the given value into the string. Precision or format cannot be set using them, you can use *Printf* (p. 669) for

this.

4.202.78 wxString::operator >>

friend istream& operator >>(istream& *is*, wxString& *str*)

Extraction from a stream.

4.202.79 wxString::operator const char*

operator const char*() const

Implicit conversion to a C string.

4.202.80 Comparison operators

bool operator ==(const wxString& *x*, const wxString& *y*)

bool operator ==(const wxString& *x*, const char* *t*)

bool operator !=(const wxString& *x*, const wxString& *y*)

bool operator !=(const wxString& *x*, const char* *t*)

bool operator >(const wxString& *x*, const wxString& *y*)

bool operator >(const wxString& *x*, const char* *t*)

bool operator >=(const wxString& *x*, const wxString& *y*)

bool operator >=(const wxString& *x*, const char* *t*)

bool operator <(const wxString& *x*, const wxString& *y*)

bool operator <(const wxString& *x*, const char* *t*)

bool operator <=(const wxString& *x*, const wxString& *y*)

bool operator <=(const wxString& *x*, const char* *t*)

Remarks

These comparisons are case-sensitive.

4.203 wxStringList

A string list is a list which is assumed to contain strings. Memory is allocated when

strings are added to the list, and deallocated by the destructor or by the **Delete** member.

Derived from

wxList (p. 367)

wxObject (p. 471)

Include files

<wx/list.h>

See also

wxString (p. 654), *wxList* (p. 367)

4.203.1 **wxStringList::wxStringList**

wxStringList()

Constructor.

void wxStringList(char* first, ...)

Constructor, taking NULL-terminated string argument list. *wxStringList* allocates memory for the strings.

4.203.2 **wxStringList::~~wxStringList**

~wxStringList()

Deletes string list, deallocating strings.

4.203.3 **wxStringList::Add**

wxNode * Add(const wxString& s)

Adds string to list, allocating memory.

4.203.4 **wxStringList::Clear**

void Clear()

Clears all strings from the list.

4.203.5 wxStringList::Delete**void Delete(const wxString& s)**

Searches for string and deletes from list, deallocating memory.

4.203.6 wxStringList::ListToArray**char* ListToArray(bool new_copies = FALSE)**

Converts the list to an array of strings, only allocating new memory if **new_copies** is TRUE.

4.203.7 wxStringList::Member**bool Member(const wxString& s)**

Returns TRUE if **s** is a member of the list (tested using **strcmp**).

4.203.8 wxStringList::Sort**void Sort()**

Sorts the strings in ascending alphabetical order. Note that all nodes (but not strings) get deallocated and new ones allocated.

4.204 wxStringTokenizer

wxStringTokenizer helps you to break a string up into a number of tokens.

Derived from

wxObject (p. 471)

Include files

<wx/tokenzr.h>

4.204.1 wxStringTokenizer::wxStringTokenizer**wxStringTokenizer()**

Default constructor.

wxStringTokenizer(const wxString& to_tokenize, const wxString& delims = " \t\r\n", bool ret_delim = FALSE)

Constructor. Pass the string to tokenize, a string containing delimiters, a flag specifying whether delimiters are retained.

4.204.2 wxStringTokenizer::~~wxStringTokenizer

~wxStringTokenizer()

Destructor.

4.204.3 wxStringTokenizer::CountTokens

int CountTokens() const

Returns the number of tokens in the input string.

4.204.4 wxStringTokenizer::HasMoreTokens

bool HasMoreTokens() const

Returns TRUE if the tokenizer has further tokens.

4.204.5 wxStringTokenizer::GetNextToken

wxString GetNextToken() const

Returns the next token.

4.204.6 wxStringTokenizer::GetString

wxString GetString() const

Returns the input string.

4.204.7 wxStringTokenizer::SetString

void SetString(const wxString& to_tokenize, const wxString& delims = " \t\r\n", bool ret_delim = FALSE)

Initializes the tokenizer.

Pass the string to tokenize, a string containing delimiters, a flag specifying whether delimiters are retained.

4.205 **wxSysColourChangedEvent**

This class is used for system colour change events, which are generated when the user changes the colour settings using the control panel. This is only appropriate under Windows.

Derived from

wxEvent (p. 221)

wxObject (p. 471)

Include files

<wx/event.h>

Event table macros

To process a system colour changed event, use this event handler macro to direct input to a member function that takes a *wxSysColourChanged* argument.

EVT_SYS_COLOUR_CHANGED(func) Process a *wxEVT_SYS_COLOUR_CHANGED* event.

Remarks

The default event handler for this event propagates the event to child windows, since Windows only sends the events to top-level windows. If intercepting this event for a top-level window, remember to call the base class handler, or to pass the event on to the window's children explicitly.

See also

wxWindow::OnSysColourChanged (p. 828), *Event handling overview* (p. 939)

4.205.1 **wxSysColourChangedEvent::wxSysColourChanged**

wxSysColourChanged()

Constructor.

4.206 **wxSystemSettings**

wxSystemSettings allows the application to ask for details about the system. This can

include settings such as standard colours, fonts, and user interface element sizes.

Derived from

wxObject (p. 471)

Include files

<wx/settings.h>

See also

wxFont (p. 264), *wxColour* (p. 86)

4.206.1 wxSystemSettings::wxSystemSettings

wxSystemSettings()

Default constructor. You don't need to create an instance of *wxSystemSettings* since all of its functions are static.

4.206.2 wxSystemSettings::GetSystemColour

static wxColour GetSystemColour(int *index*)

Returns a system colour.

index can be one of:

wxSYS_COLOUR_SCROLLBAR	The scrollbar grey area.
wxSYS_COLOUR_BACKGROUND	The desktop colour.
wxSYS_COLOUR_ACTIVECAPTION	Active window caption.
wxSYS_COLOUR_INACTIVECAPTION	Inactive window caption.
wxSYS_COLOUR_MENU	Menu background.
wxSYS_COLOUR_WINDOW	Window background.
wxSYS_COLOUR_WINDOWFRAME	Window frame.
wxSYS_COLOUR_MENUTEXT	Menu text.
wxSYS_COLOUR_WINDOWTEXT	Text in windows.
wxSYS_COLOUR_CAPTIONTEXT	Text in caption, size box and scrollbar arrow box.
wxSYS_COLOUR_ACTIVEBORDER	Active window border.
wxSYS_COLOUR_INACTIVEBORDER	Inactive window border.
wxSYS_COLOUR_APPWORKSPACE	Background colour MDI applications.
wxSYS_COLOUR_HIGHLIGHT	Item(s) selected in a control.
wxSYS_COLOUR_HIGHLIGHTTEXT	Text of item(s) selected in a control.
wxSYS_COLOUR_BTNFACE	Face shading on push buttons.
wxSYS_COLOUR_BTNSHADOW	Edge shading on push buttons.

wxSYS_COLOUR_GRAYTEXT	Greyed (disabled) text.
wxSYS_COLOUR_BTNTEXT	Text on push buttons.
wxSYS_COLOUR_INACTIVECAPTIONTEXT	Colour of text in active captions.
wxSYS_COLOUR_BTNHIGHLIGHT	Highlight colour for buttons (same as wxSYS_COLOUR_3DHILIGHT).
wxSYS_COLOUR_3DDKSHADOW	Dark shadow for three-dimensional display elements.
wxSYS_COLOUR_3DLIGHT	Light colour for three-dimensional display elements.
wxSYS_COLOUR_INFOTEXT	Text colour for tooltip controls.
wxSYS_COLOUR_INFOBK	Background colour for tooltip controls.
wxSYS_COLOUR_DESKTOP	Same as wxSYS_COLOUR_BACKGROUND.
wxSYS_COLOUR_3DFACE	Same as wxSYS_COLOUR_BTNFACE.
wxSYS_COLOUR_3DSHADOW	Same as wxSYS_COLOUR_BTNSHADOW.
wxSYS_COLOUR_3DHIGHLIGHT	Same as wxSYS_COLOUR_BTNHIGHLIGHT.
wxSYS_COLOUR_3DHILIGHT	Same as wxSYS_COLOUR_BTNHIGHLIGHT.
wxSYS_COLOUR_BTNHILIGHT	Same as wxSYS_COLOUR_BTNHIGHLIGHT.

4.206.3 wxSystemSettings::GetSystemFont

static wxFont GetSystemFont(int index)

Returns a system font.

index can be one of:

wxSYS_OEM_FIXED_FONT	Original equipment manufacturer dependent fixed-pitch font.
wxSYS_ANSI_FIXED_FONT	Windows fixed-pitch font.
wxSYS_ANSI_VAR_FONT	Windows variable-pitch (proportional) font.
wxSYS_SYSTEM_FONT	System font.
wxSYS_DEVICE_DEFAULT_FONT	Device-dependent font (Windows NT only).
wxSYS_DEFAULT_GUI_FONT	Default font for user interface objects such as menus and dialog boxes. Not available in versions of Windows earlier than Windows 95 or Windows NT 4.0.

4.206.4 wxSystemSettings::GetSystemMetric

static int GetSystemMetric(int index)

Returns a system metric.

index can be one of:

wxSYS_MOUSE_BUTTONS	Number of buttons on mouse, or zero if no mouse was installed.
----------------------------	----------------------------------------------------------------

wxSYS_BORDER_X	Width of single border.
wxSYS_BORDER_Y	Height of single border.
wxSYS_CURSOR_X	Width of cursor.
wxSYS_CURSOR_Y	Height of cursor.
wxSYS_DCLICK_X	Width in pixels of rectangle within which two successive mouse clicks must fall to generate a double-click.
wxSYS_DCLICK_Y	Height in pixels of rectangle within which two successive mouse clicks must fall to generate a double-click.
wxSYS_DRAG_X	Width in pixels of a rectangle centered on a drag point to allow for limited movement of the mouse pointer before a drag operation begins.
wxSYS_DRAG_Y	Height in pixels of a rectangle centered on a drag point to allow for limited movement of the mouse pointer before a drag operation begins.
wxSYS_EDGE_X	Width of a 3D border, in pixels.
wxSYS_EDGE_Y	Height of a 3D border, in pixels.
wxSYS_HSCROLL_ARROW_X	Width of arrow bitmap on horizontal scrollbar.
wxSYS_HSCROLL_ARROW_Y	Height of arrow bitmap on horizontal scrollbar.
wxSYS_HTHUMB_X	Width of horizontal scrollbar thumb.
wxSYS_ICON_X	The default width of an icon.
wxSYS_ICON_Y	The default height of an icon.
wxSYS_ICONSPACING_X	Width of a grid cell for items in large icon view, in pixels. Each item fits into a rectangle of this size when arranged.
wxSYS_ICONSPACING_Y	Height of a grid cell for items in large icon view, in pixels. Each item fits into a rectangle of this size when arranged.
wxSYS_WINDOWMIN_X	Minimum width of a window.
wxSYS_WINDOWMIN_Y	Minimum height of a window.
wxSYS_SCREEN_X	Width of the screen in pixels.
wxSYS_SCREEN_Y	Height of the screen in pixels.
wxSYS_FRAME_SIZE_X	Width of the window frame for a wxTHICK_FRAME window.
wxSYS_FRAME_SIZE_Y	Height of the window frame for a wxTHICK_FRAME window.
wxSYS_SMALLICON_X	Recommended width of a small icon (in window captions, and small icon view).
wxSYS_SMALLICON_Y	Recommended height of a small icon (in window captions, and small icon view).
wxSYS_HSCROLL_Y	Height of horizontal scrollbar in pixels.
wxSYS_VSCROLL_X	Width of vertical scrollbar in pixels.
wxSYS_VSCROLL_ARROW_X	Width of arrow bitmap on a vertical scrollbar.
wxSYS_VSCROLL_ARROW_Y	Height of arrow bitmap on a vertical scrollbar.
wxSYS_VTHUMB_Y	Height of vertical scrollbar thumb.
wxSYS_CAPTION_Y	Height of normal caption area.
wxSYS_MENU_Y	Height of single-line menu bar.
wxSYS_NETWORK_PRESENT	1 if there is a network present, 0 otherwise.
wxSYS_PENWINDOWS_PRESENT	1 if PenWindows is installed, 0 otherwise.
wxSYS_SHOW_SOUNDS	Non-zero if the user requires an application to

wxSYS_SWAP_BUTTONS

present information visually in situations where it would otherwise present the information only in audible form; zero otherwise.

Non-zero if the meanings of the left and right mouse buttons are swapped; zero otherwise.

4.207 wxTabbedDialog

A dialog suitable for handling tabs.

Derived from

wxDialog (p. 178)

wxEvtHandler (p. 224)

wxObject (p. 471)

Include files

<wx/tab.h>

See also

Tab classes overview (p. 959)

4.207.1 wxTabbedDialog::wxTabbedDialog

wxTabbedDialog(*wxWindow* *parent, *wxWindowID* id, **const wxString&** title, **const wxPoint&** pos, **const wxSize&** size, **long** style=wxDEFAULT_DIALOG_STYLE, **const wxString&** name="dialogBox")

Constructor.

4.207.2 wxTabbedDialog::~~wxTabbedDialog

~wxTabbedDialog()

Destructor. This destructor deletes the tab view associated with the dialog box. If you do not wish this to happen, set the tab view to NULL before destruction (for example, in the OnCloseWindow event handler).

4.207.3 wxTabbedDialog::SetTabView

void SetTabView(*wxTabView* *view)

Sets the tab view associated with the dialog box.

4.207.4 **wxTabbedDialog::GetTabView**

wxTabView * GetTabView()

Returns the tab view associated with the dialog box.

4.208 **wxTabbedPanel**

A panel suitable for handling tabs.

Derived from

wxPanel (p. 488)

wxEvtHandler (p. 224)

wxObject (p. 471)

Include files

<wx/tab.h>

See also

Tab classes overview (p. 959)

4.208.1 **wxTabbedPanel::wxTabbedPanel**

wxTabbedPanel(wxWindow *parent, wxWindowID id, const wxPoint& pos, const wxSize& size, long style=0, const wxString& name="panel")

Constructor.

4.208.2 **wxTabbedPanel::SetTabView**

void SetTabView(wxTabView *view)

Sets the tab view associated with the panel.

4.208.3 **wxTabbedPanel::GetTabView**

wxTabView * GetTabView()

Returns the tab view associated with the panel.

4.209 wxTabControl

You will rarely need to use this class directly.

Derived from

wxObject (p. 471)

Include files

<wx/tab.h>

See also

Tab classes overview (p. 959)

4.209.1 wxTabControl::wxTabControl

void wxTabControl(wxTabView *view = NULL)

Constructor.

4.209.2 wxTabControl::GetColPosition

int GetColPosition()

Returns the position of the tab in the tab column.

4.209.3 wxTabControl::GetFont

wxFont * GetFont()

Returns the font to be used for this tab.

4.209.4 wxTabControl::GetHeight

int GetHeight()

Returns the tab height.

4.209.5 wxTabControl::GetId

int GetId()

Returns the tab identifier.

4.209.6 wxTabControl::GetLabel**wxString GetLabel()**

Returns the tab label.

4.209.7 wxTabControl::GetRowPosition**int GetRowPosition()**

Returns the position of the tab in the layer or row.

4.209.8 wxTabControl::GetSelected**bool GetSelected()**

Returns the selected flag.

4.209.9 wxTabControl::GetWidth**int GetWidth()**

Returns the tab width.

4.209.10 wxTabControl::GetX**int GetX()**

Returns the x offset from the top-left of the view area.

4.209.11 wxTabControl::GetY**int GetY()**

Returns the y offset from the top-left of the view area.

4.209.12 wxTabControl::HitTest**bool HitTest(int x, int y)**

Returns TRUE if the point *x*, *y* is within the tab area.

4.209.13 wxTabControl::OnDraw

void OnDraw(wxDC& *dc*, bool *lastInRow*)

Draws the tab control on the given device context.

4.209.14 wxTabControl::SetColPosition

void SetColPosition(int *pos*)

Sets the position in the column.

4.209.15 wxTabControl::SetFont

void SetFont(wxFont **font*)

Sets the font to be used for this tab.

4.209.16 wxTabControl::SetId

void SetId(int *id*)

Sets the tab identifier.

4.209.17 wxTabControl::SetLabel

void SetLabel(const wxString& *str*)

Sets the label for the tab.

4.209.18 wxTabControl::SetPosition

void SetPosition(int *x*, int *y*)

Sets the *x* and *y* offsets for this tab, measured from the top-left of the view area.

4.209.19 wxTabControl::SetRowPosition

void SetRowPosition(int *pos*)

Sets the position on the layer (row).

4.209.20 **wxTabControl::SetSelected**

void SetSelected(bool *selected*)

Sets the selection flag for this tab (does not set the current tab for the view; use `wxTabView::SetSelectedTab` for that).

4.209.21 **wxTabControl::SetSize**

void SetSize(int *width*, int *height*)

Sets the width and height for this tab.

4.210 **wxTabView**

Responsible for drawing tabs onto a window, and dealing with input.

Derived from

wxObject (p. 471)

Include files

<wx/tab.h>

See also

wxTabView overview (p. 963), *wxPanelTabView* (p. 491)

4.210.1 **wxTabView::wxTabView**

wxTabView(long *style* = `wxTAB_STYLE_DRAW_BOX | wxTAB_STYLE_COLOUR_INTERIOR`)

Constructor.

style may be a bit list of the following:

<code>wxTAB_STYLE_DRAW_BOX</code>	Draw a box around the view area. Most commonly used for dialogs.
<code>wxTAB_STYLE_COLOUR_INTERIOR</code>	Draw tab backgrounds in the specified colour. Omitting this style will ensure that the tab background matches the dialog background.

4.210.2 wxTabView::AddTab

**wxTabControl * AddTab(int id, const wxString& label, wxTabControl
*existingTab=NULL)**

Adds a tab to the view.

id is the application-chosen identifier for the tab, which will be used in subsequent tab operations.

label is the label to give the tab.

existingTab maybe NULL to specify a new tab, or non-NULL to indicate that an existing tab should be used.

A new layer (row) is started when the current layer has been filled up with tabs.

4.210.3 wxTabView::CalculateTabWidth

int CalculateTabWidth(int noTabs, bool adjustView = FALSE)

The application can specify the tab width using this function, in terms of the number of tabs per layer (row) which will fit the view area, which should have been set previously with SetViewRect.

noTabs is the number of tabs which should take up the full width of the view area.

adjustView can be set to TRUE in order to readjust the view width to exactly fit the given number of tabs.

The new tab width is returned.

4.210.4 wxTabView::ClearTabs

void ClearTabs(bool deleteTabs=TRUE)

Clears the tabs, deleting them if *deleteTabs* is TRUE.

4.210.5 wxTabView::Draw

void Draw(wxDC& dc)

Draws the tabs and (optionally) a box around the view area.

4.210.6 wxTabView::FindTabControlForId**wxTabControl * FindTabControlForId(int id)**

Finds the wxTabControl corresponding to *id*.

4.210.7 wxTabView::FindTabControlForPosition**wxTabControl * FindTabControlForPosition(int layer, int position)**

Finds the wxTabControl at layer *layer*, position in layer *position*, both starting from zero. Note that tabs change layer as they are selected or deselected.

4.210.8 wxTabView::GetBackgroundBrush**wxBrush * GetBackgroundBrush()**

Returns the brush used to draw in the background colour. It is set when SetBackgroundColour is called.

4.210.9 wxTabView::GetBackgroundColour**wxColour GetBackgroundColour()**

Returns the colour used for each tab background. By default, this is light grey. To ensure a match with the dialog or panel background, omit the wxTAB_STYLE_COLOUR_INTERIOR flag from the wxTabView constructor.

4.210.10 wxTabView::GetBackgroundPen**wxPen * GetBackgroundPen()**

Returns the pen used to draw in the background colour. It is set when SetBackgroundColour is called.

4.210.11 wxTabView::GetHighlightColour**wxColour GetHighlightColour()**

Returns the colour used for bright highlights on the left side of '3D' surfaces. By default, this is white.

4.210.12 wxTabView::GetHighlightPen

wxPen * GetHighlightPen()

Returns the pen used to draw 3D effect highlights. This is set when SetHighlightColour is called.

4.210.13 wxTabView::GetHorizontalTabOffset**int GetHorizontalTabOffset()**

Returns the horizontal spacing by which each tab layer is offset from the one below.

4.210.14 wxTabView::GetNumberOfLayers**int GetNumberOfLayers()**

Returns the number of layers (rows of tabs).

4.210.15 wxTabView::GetSelectedTabFont**wxFont * GetSelectedTabFont()**

Returns the font to be used for the selected tab label.

4.210.16 wxTabView::GetShadowColour**wxColour GetShadowColour()**

Returns the colour used for shadows on the right-hand side of '3D' surfaces. By default, this is dark grey.

4.210.17 wxTabView::GetTabHeight**int GetTabHeight()**

Returns the tab default height.

4.210.18 wxTabView::GetTabFont**wxFont * GetTabFont()**

Returns the tab label font.

4.210.19 wxTabView::GetTabSelectionHeight**int GetTabSelectionHeight()**

Returns the height to be used for the currently selected tab; normally a few pixels higher than the other tabs.

4.210.20 wxTabView::GetTabStyle**long GetTabStyle()**

Returns the tab style. See constructor documentation for details of valid styles.

4.210.21 wxTabView::GetTabWidth**int GetTabWidth()**

Returns the tab default width.

4.210.22 wxTabView::GetTextColour**wxColour GetTextColour()**

Returns the colour used to draw label text. By default, this is black.

4.210.23 wxTabView::GetTopMargin**int GetTopMargin()**

Returns the height between the top of the view area and the bottom of the first row of tabs.

4.210.24 wxTabView::GetShadowPen**wxPen * GetShadowPen()**

Returns the pen used to draw 3D effect shadows. This is set when SetShadowColour is called.

4.210.25 wxTabView::GetViewRect**wxRectangle GetViewRect()**

Returns the rectangle specifying the view area (above which tabs are placed).

4.210.26 wxTabView::GetVerticalTabTextSpacing**int GetVerticalTabTextSpacing()**

Returns the vertical spacing between the top of an unselected tab, and the tab label.

4.210.27 wxTabView::GetWindow**wxWindow * GetWindow()**

Returns the window for the view.

4.210.28 wxTabView::OnCreateTabControl**wxTabControl * OnCreateTabControl()**

Creates a new tab control. By default, this returns a wxTabControl object, but the application may wish to define a derived class, in which case the tab view should be subclassed and this function overridden.

4.210.29 wxTabView::Layout**void Layout()**

Recalculates the positions of the tabs, and adjusts the layer of the selected tab if necessary.

You may want to call this function if the view width has changed (for example, from an OnSize handler).

4.210.30 wxTabView::OnEvent**bool OnEvent(wxMouseEvent& event)**

Processes mouse events sent from the panel or dialog. Returns TRUE if the event was processed, FALSE otherwise.

4.210.31 wxTabView::OnTabActivate**void OnTabActivate(int activateId, int deactivateId)**

Called when a tab is activated, with the new active tab id, and the former active tab id.

4.210.32 wxTabView::OnTabPreActivate**bool OnTabPreActivate**(int *activateld*, int *deactivateld*)

Called just before a tab is activated, with the new active tab id, and the former active tab id.

If the function returns FALSE, the tab is not activated.

4.210.33 wxTabView::SetBackgroundColour**void SetBackgroundColour**(const wxColour& *col*)

Sets the colour to be used for each tab background. By default, this is light grey. To ensure a match with the dialog or panel background, omit the wxTAB_STYLE_COLOUR_INTERIOR flag from the wxTabView constructor.

4.210.34 wxTabView::SetHighlightColour**void SetHighlightColour**(const wxColour& *col*)

Sets the colour to be used for bright highlights on the left side of '3D' surfaces. By default, this is white.

4.210.35 wxTabView::SetHorizontalTabOffset**void SetHorizontalTabOffset**(int *offset*)

Sets the horizontal spacing by which each tab layer is offset from the one below.

4.210.36 wxTabView::SetSelectedTabFont**void SetSelectedTabFont**(wxFont **font*)

Sets the font to be used for the selected tab label.

4.210.37 wxTabView::SetShadowColour**void SetShadowColour**(const wxColour& *col*)

Sets the colour to be used for shadows on the right-hand side of '3D' surfaces. By default, this is dark grey.

4.210.38 wxTabView::SetTabFont**void SetTabFont**(**wxFont** **font*)

Sets the tab label font.

4.210.39 wxTabView::SetTabStyle**void SetTabStyle**(**long** *tabStyle*)

Sets the tab style. See constructor documentation for details of valid styles.

4.210.40 wxTabView::SetTabSize**void SetTabSize**(**int** *width*, **int** *height*)

Sets the tab default width and height.

4.210.41 wxTabView::SetTabSelectionHeight**void SetTabSelectionHeight**(**int** *height*)

Sets the height to be used for the currently selected tab; normally a few pixels higher than the other tabs.

4.210.42 wxTabView::SetTabSelection**void SetTabSelection**(**int** *sel*, **bool** *activateTool=TRUE*)

Sets the selected tab, calling the application's OnTabActivate function.

If *activateTool* is FALSE, OnTabActivate will not be called.

4.210.43 wxTabView::SetTextColour**void SetTextColour**(**const wxColour&** *col*)

Sets the colour to be used to draw label text. By default, this is black.

4.210.44 wxTabView::SetTopMargin**void SetTopMargin**(**int** *margin*)

Sets the height between the top of the view area and the bottom of the first row of tabs.

4.210.45 wxTabView::SetVerticalTabTextSpacing**void SetVerticalTabTextSpacing**(int *spacing*)

Sets the vertical spacing between the top of an unselected tab, and the tab label.

4.210.46 wxTabView::SetViewRect**void SetViewRect**(const wxRectangle& *rect*)

Sets the rectangle specifying the view area (above which tabs are placed). This must be set by the application.

4.210.47 wxTabView::SetWindow**void SetWindow**(wxWindow **window*)

Set the window that the tab view will use for drawing onto.

4.211 wxTabCtrl

This class represents a tab control, which manages multiple tabs.

Derived from

wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/tabctrl.h>

See also

wxTabEvent (p. 700), *wxImageList* (p. 339), *wxNotebook* (p. 464)

4.211.1 wxTabCtrl::wxTabCtrl**wxTabCtrl**()

Default constructor.

wxTabCtrl(wxWindow* *parent*, wxWindowID *id*, const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size*, long *style* = 0, const wxString& *name* = "tabCtrl")

Constructs a tab control.

Parameters

parent

The parent window. Must be non-NULL.

id

The window identifier.

pos

The window position.

size

The window size.

style

The window style. Its value is a bit list of zero or more of **wxTC_MULTILINE**, **wxTC_RIGHTJUSTIFY**, **wxTC_FIXEDWIDTH** and **wxTC_OWNERDRAW**.

4.211.2 wxTabCtrl::~~wxTabCtrl

~wxTabCtrl()

Destroys the wxTabCtrl object.

4.211.3 wxTabCtrl::Create

bool Create(wxWindow* *parent*, wxWindowID *id*, const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size*, long *style* = 0, const wxString& *name* = "tabCtrl")

Creates a tab control. See *wxTabCtrl::wxTabCtrl* (p. 695) for a description of the parameters.

4.211.4 wxTabCtrl::DeleteAllItems

bool DeleteAllItems()

Deletes all tab items.

4.211.5 wxTabCtrl::DeleteItem**bool DeleteItem(int item)**

Deletes the specified tab item.

4.211.6 wxTabCtrl::GetCurFocus**int GetCurFocus() const**

Returns the index for the tab with the focus, or -1 if none has the focus.

4.211.7 wxTabCtrl::GetImageList**wxImageList* GetImageList() const**

Returns the associated image list.

[See also](#)

wxImageList (p. 339), *wxTabCtrl::SetImageList* (p. 699)

4.211.8 wxTabCtrl::GetItemCount**int GetItemCount() const**

Returns the number of tabs in the tab control.

4.211.9 wxTabCtrl::GetItemData**void* GetItemData() const**

Returns the client data for the given tab.

4.211.10 wxTabCtrl::GetItemImage**int GetItemImage() const**

Returns the image index for the given tab.

4.211.11 wxTabCtrl::GetItemRect**bool GetItemRect(int item, wxRect& rect) const**

Returns the rectangle bounding the given tab.

[See also](#)

wxRect (p. 546)

4.211.12 **wxTabCtrl::GetItemText**

wxString GetItemText() const

Returns the string for the given tab.

4.211.13 **wxTabCtrl::GetRowCount**

int GetRowCount() const

Returns the number of rows in the tab control.

4.211.14 **wxTabCtrl::GetSelection**

int GetSelection() const

Returns the index for the currently selected tab.

[See also](#)

wxTabCtrl::SetSelection (p. 700)

4.211.15 **wxTabCtrl::HitTest**

int HitTest(const wxPoint& pt, long& flags)

Tests whether a tab is at the specified position.

[Parameters](#)

pt

Specifies the point for the hit test.

flags

Return value for detailed information. One of the following values:

wxTAB_HITTEST_NOWHERE
wxTAB_HITTEST_ONICON
wxTAB_HITTEST_ONLABEL
wxTAB_HITTEST_ONITEM

There was no tab under this point.
The point was over an icon.
The point was over a label.
The point was over an item, but not on

the label or icon.

Return value

Returns the zero-based tab index or -1 if no tab is at the specified position.

4.211.16 wxTabCtrl::InsertItem

void InsertItem(int *item*, const wxString& *text*, int *imageId* = -1, void* *clientData* = NULL)

Inserts a new tab.

Parameters

item

Specifies the index for the new item.

text

Specifies the text for the new item.

imageId

Specifies the optional image index for the new item.

clientData

Specifies the optional client data for the new item.

Return value

TRUE if successful, FALSE otherwise.

4.211.17 wxTabCtrl::SetItemData

bool SetItemData(int *item*, void* *data*)

Sets the client data for a tab.

4.211.18 wxTabCtrl::SetItemImage

bool SetItemImage(int *item*, int *image*)

Sets the image index for the given tab. *image* is an index into the image list which was set with *wxTabCtrl::SetImageList* (p. 699).

4.211.19 wxTabCtrl::SetImageList

void SetImageList(wxImageList* *imageList*)

Sets the image list for the tab control.

[See also](#)

wxImageList (p. 339)

4.211.20 wxTabCtrl::SetItemSize

void SetItemSize(const wxSize& *size*)

Sets the width and height of the tabs.

4.211.21 wxTabCtrl::SetItemText

bool SetItemText(int *item*, const wxString& *text*)

Sets the text for the given tab.

4.211.22 wxTabCtrl::SetPadding

void SetPadding(const wxSize& *padding*)

Sets the amount of space around each tab's icon and label.

4.211.23 wxTabCtrl::SetSelection

int SetSelection(int *item*)

Sets the selection for the given tab, returning the index of the previously selected tab. Returns -1 if the call was unsuccessful.

[See also](#)

wxTabCtrl::GetSelection (p. 698)

4.212 wxTabEvent

This class represents the events generated by a tab control.

[Derived from](#)

wxCommandEvent (p. 103)

wxEvent (p. 221)

wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/tabctrl.h>

Event table macros

To process a tab event, use these event handler macros to direct input to member functions that take a *wxTabEvent* argument.

EVT_TAB_SEL_CHANGED(id, func)	Process a <i>wxEVT_TAB_SEL_CHANGED</i> event, indicating that the tab selection has changed.
EVT_TAB_SEL_CHANGING(id, func)	Process a <i>wxEVT_TAB_SEL_CHANGING</i> event, indicating that the tab selection is changing.

See also

wxTabCtrl (p. 695)

4.212.1 wxTabEvent::wxTabEvent

wxTabEvent(*WXTYPE* *commandType* = 0, *int* *id* = 0)

Constructor.

4.213 wxTaskBarIcon

This class represents a Windows 95 taskbar icon, appearing in the 'system tray' and responding to mouse clicks. An icon has an optional tooltip. This class is only supported for Windows 95/NT.

Derived from

wxObject (p. 471)

Include files

<wx/taskbar.h>

4.213.1 wxTaskBarIcon::wxTaskBarIcon**wxTaskBarIcon()**

Default constructor.

4.213.2 wxTaskBarIcon::~~wxTaskBarIcon**~wxTaskBarIcon()**

Destroys the wxTaskBarIcon object, removing the icon if not already removed.

4.213.3 wxTaskBarIcon::IsIconInstalled**bool IsIconInstalled()**

Returns TRUE if *SetIcon* (p. 703) was called with no subsequent *RemoveIcon* (p. 703).

4.213.4 wxTaskBarIcon::IsOK**bool IsOK()**

Returns TRUE if the object initialized successfully.

4.213.5 wxTaskBarIcon::OnLButtonDown**virtual void OnLButtonDown()**

Override this function to intercept left mouse button down events.

4.213.6 wxTaskBarIcon::OnLButtonDClick**virtual void OnLButtonDClick()**

Override this function to intercept left mouse button double-click events.

4.213.7 wxTaskBarIcon::OnLButtonUp**virtual void OnLButtonUp()**

Override this function to intercept left mouse button up events.

4.213.8 wxTaskBarIcon::OnRButtonDown

virtual void OnRButtonDown()

Override this function to intercept right mouse button down events.

4.213.9 wxTaskBarIcon::OnRButtonDClick**virtual void OnRButtonDClick()**

Override this function to intercept right mouse button double-click events.

4.213.10 wxTaskBarIcon::OnRButtonUp**virtual void OnRButtonUp()**

Override this function to intercept right mouse button up events.

4.213.11 wxTaskBarIcon::OnMouseMove**virtual void OnMouseMove()**

Override this function to intercept mouse move events.

4.213.12 wxTaskBarIcon::RemoveIcon**bool RemoveIcon()**

Removes the icon previously set with *SetIcon* (p. 703).

4.213.13 wxTaskBarIcon::SetIcon**bool SetIcon(const wxIcon& icon, const wxString& tooltip)**

Sets the icon, and optional tooltip text.

4.214 wxTCPClient

A wxTCPClient object represents the client part of a client-server conversation. It emulates a DDE-style protocol, but uses TCP/IP which is available on most platforms.

A DDE-based implementation for Windows is available using *wxDDEClient* (p. 166).

To create a client which can communicate with a suitable server, you need to derive a class from wxTCPConnection and another from wxTCPClient. The custom wxTCPConnection class will intercept communications in a 'conversation' with a server,

and the custom `wxTCPServer` is required so that a user-overridden `wxTCPClient::OnMakeConnection` (p. 704) member can return a `wxTCPConnection` of the required class, when a connection is made.

Derived from

`wxClientBase`
`wxObject` (p. 471)

Include files

`<wx/sckipc.h>`

See also

`wxTCPServer` (p. 709), `wxTCPConnection` (p. 705), *Interprocess communications overview* (p. 946)

4.214.1 `wxTCPClient::wxTCPClient`

`wxTCPClient()`

Constructs a client object.

4.214.2 `wxTCPClient::MakeConnection`

`wxConnectionBase * MakeConnection(const wxString& host, const wxString& service, const wxString& topic)`

Tries to make a connection with a server specified by the host (a machine name under Unix), service name (must contain an integer port number under Unix), and a topic string. If the server allows a connection, a `wxTCPConnection` object will be returned. The type of `wxTCPConnection` returned can be altered by overriding the `wxTCPClient::OnMakeConnection` (p. 704) member to return your own derived connection object.

4.214.3 `wxTCPClient::OnMakeConnection`

`wxConnectionBase * OnMakeConnection()`

The type of `wxTCPConnection` (p. 705) returned from a `wxTCPClient::MakeConnection` (p. 704) call can be altered by deriving the **`OnMakeConnection`** member to return your own derived connection object. By default, a `wxTCPConnection` object is returned.

The advantage of deriving your own connection class is that it will enable you to

intercept messages initiated by the server, such as *wxTCPConnection::OnAdvise* (p. 707). You may also want to store application-specific data in instances of the new class.

4.214.4 wxTCPClient::ValidHost

bool ValidHost(const wxString& host)

Returns TRUE if this is a valid host name, FALSE otherwise.

4.215 wxTCPConnection

A *wxTCPClient* object represents the connection between a client and a server. It emulates a DDE-style protocol, but uses TCP/IP which is available on most platforms.

A DDE-based implementation for Windows is available using *wxDDEConnection* (p. 167).

A *wxTCPConnection* object can be created by making a connection using a *wxTCPClient* (p. 703) object, or by the acceptance of a connection by a *wxTCPServer* (p. 709) object. The bulk of a conversation is controlled by calling members in a **wxTCPConnection** object or by overriding its members.

An application should normally derive a new connection class from *wxTCPConnection*, in order to override the communication event handlers to do something interesting.

Derived from

wxConnectionBase
wxObject (p. 471)

Include files

<wx/sckipc.h>

Types

wxIPCFormat is defined as follows:

```
enum wxIPCFormat
{
    wxIPC_INVALID =          0,
    wxIPC_TEXT =             1, /* CF_TEXT */
    wxIPC_BITMAP =          2, /* CF_BITMAP */
    wxIPC_METAFILE =        3, /* CF_METAFILEPICT */
    wxIPC_SYLK =             4,
    wxIPC_DIF =              5,
    wxIPC_TIFF =             6,
    wxIPC_OEMTEXT =          7, /* CF_OEMTEXT */
    wxIPC_DIB =              8, /* CF_DIB */
    wxIPC_PALETTE =          9,
    wxIPC_PENDATA =         10,
    wxIPC_RIFF =            11,
```

```
wxIPC_WAVE =          12,
wxIPC_UNICODETEXT =   13,
wxIPC_ENHMETAFILE =   14,
wxIPC_FILENAME =      15, /* CF_HDROP */
wxIPC_LOCALE =         16,
wxIPC_PRIVATE =        20
};
```

See also

wxTCPClient (p. 703), *wxTCPServer* (p. 709), *Interprocess communications overview* (p. 946)

4.215.1 wxTCPConnection::wxTCPConnection

wxTCPConnection()

wxTCPConnection(char* buffer, int size)

Constructs a connection object. If no user-defined connection object is to be derived from *wxTCPConnection*, then the constructor should not be called directly, since the default connection object will be provided on requesting (or accepting) a connection. However, if the user defines his or her own derived connection object, the *wxTCPServer::OnAcceptConnection* (p. 710) and/or *wxTCPClient::OnMakeConnection* (p. 704) members should be replaced by functions which construct the new connection object. If the arguments of the *wxTCPConnection* constructor are void, then a default buffer is associated with the connection. Otherwise, the programmer must provide a a buffer and size of the buffer for the connection object to use in transactions.

4.215.2 wxTCPConnection::Advise

bool Advise(const wxString& item, char* data, int size = -1, wxIPCFormat format = wxCF_TEXT)

Called by the server application to advise the client of a change in the data associated with the given item. Causes the client connection's *wxTCPConnection::OnAdvise* (p. 707) member to be called. Returns TRUE if successful.

4.215.3 wxTCPConnection::Execute

bool Execute(char* data, int size = -1, wxIPCFormat format = wxCF_TEXT)

Called by the client application to execute a command on the server. Can also be used to transfer arbitrary data to the server (similar to *wxTCPConnection::Poke* (p. 708) in that respect). Causes the server connection's *wxTCPConnection::OnExecute* (p. 707) member to be called. Returns TRUE if successful.

4.215.4 wxTCPConnection::Disconnect

bool Disconnect()

Called by the client or server application to disconnect from the other program; it causes the *wxTCPConnection::OnDisconnect* (p. 707) message to be sent to the corresponding connection object in the other program. The default behaviour of **OnDisconnect** is to delete the connection, but the calling application must explicitly delete its side of the connection having called **Disconnect**. Returns TRUE if successful.

4.215.5 wxTCPConnection::OnAdvise

virtual bool OnAdvise(const wxString& topic, const wxString& item, char* data, int size, wxIPCFormat format)

Message sent to the client application when the server notifies it of a change in the data associated with the given item.

4.215.6 wxTCPConnection::OnDisconnect

virtual bool OnDisconnect()

Message sent to the client or server application when the other application notifies it to delete the connection. Default behaviour is to delete the connection object.

4.215.7 wxTCPConnection::OnExecute

virtual bool OnExecute(const wxString& topic, char* data, int size, wxIPCFormat format)

Message sent to the server application when the client notifies it to execute the given data. Note that there is no item associated with this message.

4.215.8 wxTCPConnection::OnPoke

virtual bool OnPoke(const wxString& topic, const wxString& item, char* data, int size, wxIPCFormat format)

Message sent to the server application when the client notifies it to accept the given data.

4.215.9 wxTCPConnection::OnRequest

virtual char* OnRequest(const wxString& topic, const wxString& item, int *size,

wxIPCFormat *format*)

Message sent to the server application when the client calls *wxTCPConnection::Request* (p. 708). The server should respond by returning a character string from **OnRequest**, or NULL to indicate no data.

4.215.10 wxTCPConnection::OnStartAdvise

virtual bool OnStartAdvise(const wxString& topic, const wxString& item)

Message sent to the server application by the client, when the client wishes to start an 'advise loop' for the given topic and item. The server can refuse to participate by returning FALSE.

4.215.11 wxTCPConnection::OnStopAdvise

virtual bool OnStopAdvise(const wxString& topic, const wxString& item)

Message sent to the server application by the client, when the client wishes to stop an 'advise loop' for the given topic and item. The server can refuse to stop the advise loop by returning FALSE, although this doesn't have much meaning in practice.

4.215.12 wxTCPConnection::Poke

bool Poke(const wxString& item, char* data, int size = -1, wxIPCFormat format = wxCF_TEXT)

Called by the client application to poke data into the server. Can be used to transfer arbitrary data to the server. Causes the server connection's *wxTCPConnection::OnPoke* (p. 707) member to be called. Returns TRUE if successful.

4.215.13 wxTCPConnection::Request

char* Request(const wxString& item, int *size, wxIPCFormat format = wxIPC_TEXT)

Called by the client application to request data from the server. Causes the server connection's *wxTCPConnection::OnRequest* (p. 707) member to be called. Returns a character string (actually a pointer to the connection's buffer) if successful, NULL otherwise.

4.215.14 wxTCPConnection::StartAdvise

bool StartAdvise(const wxString& item)

Called by the client application to ask if an advise loop can be started with the server.

Causes the server connection's *wxTCPConnection::OnStartAdvise* (p. 708) member to be called. Returns TRUE if the server okays it, FALSE otherwise.

4.215.15 **wxTCPConnection::StopAdvise**

bool StopAdvise(const wxString& item)

Called by the client application to ask if an advise loop can be stopped. Causes the server connection's *wxTCPConnection::OnStopAdvise* (p. 708) member to be called. Returns TRUE if the server okays it, FALSE otherwise.

4.216 **wxTCPServer**

A *wxTCPServer* object represents the server part of a client-server conversation. It emulates a DDE-style protocol, but uses TCP/IP which is available on most platforms.

A DDE-based implementation for Windows is available using *wxDDEServer* (p. 172).

Derived from

wxServerBase
wxObject (p. 471)

Include files

<wx/sckipc.h>

See also

wxTCPClient (p. 703), *wxTCPConnection* (p. 705), *IPC overview* (p. 946)

4.216.1 **wxTCPServer::wxTCPServer**

wxTCPServer()

Constructs a server object.

4.216.2 **wxTCPServer::Create**

bool Create(const wxString& service)

Registers the server using the given service name. Under Unix, the string must contain an integer id which is used as an Internet port number. FALSE is returned if the call failed (for example, the port number is already in use).

4.216.3 wxTCPServer::OnAcceptConnection

virtual wxConnectionBase * OnAcceptConnection(const wxString& topic)

When a client calls **MakeConnection**, the server receives the message and this member is called. The application should derive a member to intercept this message and return a connection object of either the standard wxTCPConnection type, or of a user-derived type. If the topic is "STDIO", the application may wish to refuse the connection. Under Unix, when a server is created the OnAcceptConnection message is always sent for standard input and output.

4.217 wxTempFile

wxTempFile provides a relatively safe way to replace the contents of the existing file. The name is explained by the fact that it may be also used as just a temporary file if you don't replace the old file contents.

Usually, when a program replaces the contents of some file it first opens it for writing, thus losing all of the old data and then starts recreating it. This approach is not very safe because during the regeneration of the file bad things may happen: the program may find that there is an internal error preventing it from completing file generation, the user may interrupt it (especially if file generation takes long time) and, finally, any other external interrupts (power supply failure or a disk error) will leave you without either the original file or the new one.

wxTempFile addresses this problem by creating a temporary file which is meant to replace the original file - but only after it is fully written. So, if the user interrupts the program during the file generation, the old file won't be lost. Also, if the program discovers itself that it doesn't want to replace the old file there is no problem - in fact, wxTempFile will **not** replace the old file by default, you should explicitly call *Commit* (p. 711) to do it. Calling *Discard* (p. 712) explicitly discards any modifications: it closes and deletes the temporary file and leaves the original file unchanged. If you don't call neither of *Commit()* and *Discard()*, the destructor will call *Discard()* automatically.

To summarize: if you want to replace another file, create an instance of wxTempFile passing the name of the file to be replaced to the constructor (you may also use default constructor and pass the file name to *Open* (p. 711)). Then you can *write* (p. 711) to wxTempFile using *wxFile* (p. 241)-like functions and later call *Commit()* to replace the old file (and close this one) or call *Discard()* to cancel the modifications.

Derived from

No base class

Include files

<wx/file.h>

See also:

wxFile (p. 241)

4.217.1 wxTempFile::wxTempFile

wxTempFile()

Default constructor - *Open* (p. 711) must be used to open the file.

4.217.2 wxTempFile::wxTempFile

wxTempFile(const wxString& *strName*)

Associates wxTempFile with the file to be replaced and opens it. You should use *IsOpened* (p. 711) to verify if the constructor succeeded.

4.217.3 wxTempFile::Open

bool Open(const wxString& *strName*)

Open the temporary file (*strName* is the name of file to be replaced), returns TRUE on success, FALSE if an error occurred.

4.217.4 wxTempFile::IsOpened

bool IsOpened() const

Returns TRUE if the file was successfully opened.

4.217.5 wxTempFile::Write

bool Write(const void **p*, size_t *n*)

Write to the file, return TRUE on success, FALSE on failure.

4.217.6 wxTempFile::Write

bool Write(const wxString& *str*)

Write to the file, return TRUE on success, FALSE on failure.

4.217.7 wxTempFile::Commit

bool Commit()

Validate changes: deletes the old file of name `m_strName` and renames the new file to the old name. Returns `TRUE` if both actions succeeded. If `FALSE` is returned it may unfortunately mean two quite different things: either that either the old file couldn't be deleted or that the new file couldn't be renamed to the old name.

4.217.8 wxTempFile::Discard**void Discard()**

Discard changes: the old file contents is not changed, temporary file is deleted.

4.217.9 wxTempFile::~wxTempFile**~wxTempFile()**

Destructor calls *Discard()* (p. 712) if temporary file is still opened.

4.218 wxTextCtrl

A text control allows text to be displayed and edited. It may be single line or multi-line.

Derived from

streambuf
wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/textctrl.h>

Window styles

wxTE_PROCESS_ENTER	The callback function will receive the message <code>wxEVENT_TYPE_TEXT_ENTER_COMMAND</code> . Note that this will break tab traversal for this panel item under Windows.
wxTE_MULTILINE	The text control allows multiple lines.
wxTE_PASSWORD	The text will be echoed as asterisks.
wxTE_READONLY	The text will not be user-editable.
wxHSCROLL	A horizontal scrollbar will be created.

See also *window styles overview* (p. 959) and *wxTextCtrl::wxTextCtrl* (p. 713).

Remarks

This class multiply-inherits from **streambuf** where compilers allow, allowing code such as the following:

```
wxTextCtrl *control = new wxTextCtrl(...);

ostream stream(control)

stream << 123.456 << " some text\n";
stream.flush();
```

If your compiler does not support derivation from **streambuf** and gives a compile error, define the symbol **NO_TEXT_WINDOW_STREAM** in the *wxTextCtrl* header file.

Event handling

To process input from a text control, use these event handler macros to direct input to member functions that take a *wxCommandEvent* (p. 103) argument.

EVT_TEXT(id, func)	Respond to a wxEVT_COMMAND_TEXT_UPDATED event, generated when the text changes.
EVT_TEXT_ENTER(id, func)	Respond to a wxEVT_COMMAND_TEXT_ENTER event, generated when enter is pressed in a single- line text control.

4.218.1 wxTextCtrl::wxTextCtrl

wxTextCtrl()

Default constructor.

wxTextCtrl(wxWindow* parent, wxWindowID id, const wxString& value = "", const wxPoint& pos, const wxSize& size = wxDefaultSize, long style = 0, const wxValidator& validator, const wxString& name = "text")

Constructor, creating and showing a text control.

Parameters

parent

Parent window. Should not be NULL.

<i>id</i>	Control identifier. A value of -1 denotes a default value.
<i>value</i>	Default text value.
<i>pos</i>	Text control position.
<i>size</i>	Text control size.
<i>style</i>	Window style. See <i>wxTextCtrl</i> (p. 712).
<i>validator</i>	Window validator.
<i>name</i>	Window name.

Remarks

The horizontal scrollbar (**wxTE_HSCROLL** style flag) will only be created for multi-line text controls. Without a horizontal scrollbar, text lines that don't fit in the control's size will be wrapped (but no newline character is inserted). Single line controls don't have a horizontal scrollbar, the text is automatically scrolled so that the *insertion point* (p. 715) is always visible.

Under Windows, if the **wxTE_MULTILINE** style is used, the window is implemented as a Windows rich text control with unlimited capacity. Otherwise, normal edit control limits apply.

See also

wxTextCtrl::Create (p. 715), *wxValidator* (p. 781)

4.218.2 **wxTextCtrl::~~wxTextCtrl**

~wxTextCtrl()

Destructor, destroying the text control.

4.218.3 **wxTextCtrl::Clear**

virtual void Clear()

Clears the text in the control.

4.218.4 **wxTextCtrl::Copy**

virtual void Copy()

Copies the selected text to the clipboard under Motif and MS Windows.

4.218.5 **wxTextCtrl::Create**

bool Create(wxWindow* parent, wxWindowID id, const wxString& value = "", const wxPoint& pos, const wxSize& size = wxDefaultSize, long style = 0, const wxValidator& validator, const wxString& name = "text")

Creates the text control for two-step construction. Derived classes should call or replace this function. See *wxTextCtrl::wxTextCtrl* (p. 713) for further details.

4.218.6 **wxTextCtrl::Cut**

virtual void Cut()

Copies the selected text to the clipboard and removes the selection.

4.218.7 **wxTextCtrl::DiscardEdits**

void DiscardEdits()

Resets the internal 'modified' flag as if the current edits had been saved.

4.218.8 **wxTextCtrl::GetInsertionPoint**

virtual long GetInsertionPoint() const

Returns the insertion point. This is defined as the zero based index of the character position to the right of the insertion point. For example, if the insertion point is at the end of the text control, it is equal to both *GetValue()* (p. 717).Length() and *GetLastPosition()* (p. 716).

The following code snippet safely returns the character at the insertion point or the zero character if the point is at the end of the control.

```
char GetCurrentChar(wxTextCtrl *tc) {  
    if (tc->GetInsertionPoint() == tc->GetLastPosition())  
        return '\\0';  
    return tc->GetValue[tc->GetInsertionPoint()];  
}
```

4.218.9 wxTextCtrl::GetLastPosition**virtual long GetLastPosition() const**

Returns the zero based index of the last position in the text control, which is equal to the number of characters in the control.

4.218.10 wxTextCtrl::GetLineLength**int GetLineLength(long *lineNo*) const**

Gets the length of the specified line, not including any trailing newline character(s).

Parameters*lineNo*

Line number (starting from zero).

Return value

The length of the line, or -1 if *lineNo* was invalid.

4.218.11 wxTextCtrl::GetLineText**wxString GetLineText(long *lineNo*) const**

Returns the contents of a given line in the text control, not including any trailing newline character(s).

Parameters*lineNo*

The line number, starting from zero.

Return value

The contents of the line.

4.218.12 wxTextCtrl::GetNumberOfLines**int GetNumberOfLines() const**

Returns the number of lines in the text control buffer.

Remarks

Note that even empty text controls have one line (where the insertion point is), so `GetNumberOfLines()` never returns 0.

For `gtk_text` (multi-line) controls, the number of lines is calculated by actually counting newline characters in the buffer. You may wish to avoid using functions that work with line numbers if you are working with controls that contain large amounts of text.

4.218.13 `wxTextCtrl::GetValue`

`wxString GetValue() const`

Gets the contents of the control.

4.218.14 `wxTextCtrl::IsModified`

`bool IsModified() const`

Returns TRUE if the text has been modified.

4.218.15 `wxTextCtrl::LoadFile`

`bool LoadFile(const wxString& filename)`

Loads and displays the named file, if it exists.

Parameters

filename

The filename of the file to load.

Return value

TRUE if successful, FALSE otherwise.

4.218.16 `wxTextCtrl::OnChar`

`void OnChar(wxKeyEvent& event)`

Default handler for character input.

Remarks

It is possible to intercept character input by overriding this member. Call this function to let the default behaviour take place; not calling it results in the character being ignored. You can replace the *keyCode* member of *event* to translate keystrokes.

Note that Windows and Motif have different ways of implementing the default behaviour. In Windows, calling `wxTextCtrl::OnChar` immediately processes the character. In Motif, calling this function simply sets a flag to let default processing happen. This might affect the way in which you write your `OnChar` function on different platforms.

See also

`wxKeyEvent` (p. 359)

4.218.17 `wxTextCtrl::OnDropFiles`

void OnDropFiles(`wxDropFilesEvent&` *event*)

This event handler function implements default drag and drop behaviour, which is to load the first dropped file into the control.

Parameters

event
The drop files event.

Remarks

This is not yet implemented for the GTK.

See also

`wxDropFilesEvent` (p. 214)

4.218.18 `wxTextCtrl::Paste`

virtual void Paste()

Pastes text from the clipboard to the text item.

4.218.19 `wxTextCtrl::PositionToXY`

long PositionToXY(long *pos*, long **x*, long **y*) const

Converts given position to a zero-based column, line number pair.

Parameters

pos
Position.

x

Receives zero based column number.

y

Receives zero based line number.

Return value

Non-zero on success, zero on failure (most likely due to a too large position parameter).

See also

wxTextCtrl::XYToPosition (p. 722)

wxPython note:

In Python, `PositionToXY()` returns a tuple containing the x and y values, so `(x,y) = PositionToXY()` is equivalent to the call described above.

4.218.20 **wxTextCtrl::Remove**

virtual void Remove(long *from*, long *to*)

Removes the text starting at the first given position up to (but not including) the character at the last position.

Parameters

from

The first position.

to

The last position.

4.218.21 **wxTextCtrl::Replace**

virtual void Replace(long *from*, long *to*, const wxString& *value*)

Replaces the text starting at the first position up to (but not including) the character at the last position with the given text.

Parameters

from

The first position.

to

The last position.

value

The value to replace the existing text with.

4.218.22 wxTextCtrl::SaveFile**bool SaveFile(const wxString& filename)**

Saves the contents of the control in a text file.

Parameters*filename*

The name of the file in which to save the text.

Return value

TRUE if the operation was successful, FALSE otherwise.

4.218.23 wxTextCtrl::SetEditable**virtual void SetEditable(const bool editable)**

Makes the text item editable or read-only, overriding the **wxTE_READONLY** flag.

Parameters*editable*

If TRUE, the control is editable. If FALSE, the control is read-only.

4.218.24 wxTextCtrl::SetInsertionPoint**virtual void SetInsertionPoint(long pos)**

Sets the insertion point at the given position.

Parameters*pos*

Position to set.

4.218.25 wxTextCtrl::SetInsertionPointEnd**virtual void SetInsertionPointEnd()**

Sets the insertion point at the end of the text control. This is equivalent to *SetInsertionPoint* (p. 720)(*GetLastPosition* (p. 716)()).

4.218.26 wxTextCtrl::SetSelection**virtual void SetSelection(long *from*, long *to*)**

Selects the text starting at the first position up to (but not including) the character at the last position.

Parameters*from*

The first position.

to

The last position.

4.218.27 wxTextCtrl::SetValue**virtual void SetValue(const wxString& *value*)**

Sets the text value.

Parameters*value*

The new value to set. It may contain newline characters if the text control is multi-line.

4.218.28 wxTextCtrl::ShowPosition**void ShowPosition(long *pos*)**

Makes the line containing the given position visible.

Parameters*pos*

The position that should be visible.

4.218.29 wxTextCtrl::WriteText**void WriteText(const wxString& *text*)**

Writes the text into the text control at the current insertion position.

Parameters*text*

Text to write to the text control.

Remarks

Newlines in the text string are the only control characters allowed, and they will cause appropriate line breaks. See *wxTextCtrl::<<* (p. 723) and *wxTextCtrl::AppendText* (p. 722) for more convenient ways of writing to the window.

After the write operation, the insertion point will be at the end of the inserted text, so subsequent write operations will be appended. To append text after the user may have interacted with the control, call *wxTextCtrl::SetInsertionPointEnd* (p. 720) before writing.

4.218.30 **wxTextCtrl::AppendText**

void AppendText(const wxString& text)

Appends the text to the end of the text control.

Parameters

text

Text to write to the text control.

Remarks

After the text is appended, the insertion point will be at the end of the text control. If this behaviour is not desired, the programmer should use *GetInsertionPoint* (p. 715) and *SetInsertionPoint* (p. 720).

See also

wxTextCtrl::WriteText (p. 721)

4.218.31 **wxTextCtrl::XYToPosition**

long XYToPosition(long x, long y)

Converts the given zero based column and line number to a position.

Parameters

x

The column number.

y

The line number.

Return value

The position value.

4.218.32 wxTextCtrl::operator <<

wxTextCtrl& operator <<(const wxString& s)

wxTextCtrl& operator <<(int i)

wxTextCtrl& operator <<(long l)

wxTextCtrl& operator <<(float f)

wxTextCtrl& operator <<(double d)

wxTextCtrl& operator <<(char c)

Operator definitions for appending to a text control, for example:

```
wxTextCtrl *wnd = new wxTextCtrl(my_frame);  
  
(*wnd) << "Welcome to text control number " << 1 << ".\n";
```

4.219

wxTextDataObject

`wxTextDataObject` is a specialization of `wxDataObject` for text data. It can be used without change to paste data into the `wxClipboard` (p. 82) or a `wxDropSource` (p. 216). A user may wish to derive a new class from this class for providing text on-demand in order to minimize memory consumption when offering data in several formats, such as plain text and RTF.

In order to offer text data on-demand `GetSize` (p. 724) and `WriteData` (p. 724) will have to be overridden.

Derived from

`wxDataObject` (p. 138)

Include files

<wx/dataobj.h>

See also

`wxDataObject` (p. 138)

4.219.1 wxTextDataObject::wxTextDataObject

wxTextDataObject()

Default constructor. Call *SetText* (p. 724) later or override *WriteData* (p. 724) and *GetSize* (p. 724) for providing data on-demand.

wxTextDataObject(const wxString& strText)

Constructor, passing text.

4.219.2 wxTextDataObject::GetSize**virtual size_t GetSize() const**

Returns the data size. By default, returns the size of the text data set in the constructor or using *SetText* (p. 724). This can be overridden to provide text size data on-demand. It is recommended to return the text length plus 1 for a trailing zero, but this is not strictly required.

4.219.3 wxTextDataObject::GetText**virtual wxString GetText() const**

Returns the text associated with the data object. You may wish to override this method when offering data on-demand, but this is not required by wxWindows' internals. Use this method to get data in text form from the *wxClipboard* (p. 82).

4.219.4 wxTextDataObject::SetText**virtual void SetText(const wxString& strText)**

Sets the text associated with the data object. This method is called internally when retrieving data from the *wxClipboard* (p. 82) and may be used to paste data to the clipboard directly (instead of on-demand).

4.219.5 wxTextDataObject::WriteData**virtual void WriteData(void*dest) const**

Write the data owned by this class to *dest*. By default, this calls *WriteString* (p. 724) with the string set in the constructor or using *SetText* (p. 724). This can be overridden to provide text data on-demand; in this case *WriteString* (p. 724) must be called from within the overriding *WriteData*() method.

4.219.6 wxTextDataObject::WriteString

void WriteString(const wxString& strvoid*dest) const

Writes the the string *str* to *dest*. This method must be called from *WriteData* (p. 724).

4.220 wxTextEntryDialog

This class represents a dialog that requests a one-line text string from the user. It is implemented as a generic wxWindows dialog.

Derived from

wxDialog (p. 178)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/textdlg.h>

See also

wxTextEntryDialog overview (p. 918)

4.220.1 wxTextEntryDialog::wxTextEntryDialog

wxTextEntryDialog(wxWindow* parent, const wxString& message, const wxString& caption = "Please enter text", const wxString& defaultValue = "", long style = wxOK | wxCANCEL | wxCENTRE, const wxPoint& pos = wxDefaultPosition)

Constructor. Use *wxTextEntryDialog::ShowModal* (p. 726) to show the dialog.

Parameters

parent
Parent window.

message
Message to show on the dialog.

defaultValue
The default value, which may be the empty string.

style
A dialog style, specifying the buttons (wxOK, wxCANCEL) and an optional wxCENTRE style.

pos

Dialog position.

4.220.2 wxTextEntryDialog::~wxTextEntryDialog

~wxTextEntryDialog()

Destructor.

4.220.3 wxTextEntryDialog::GetValue

wxString GetValue() const

Returns the text that the user has entered if the user has pressed OK, or the original value if the user has pressed Cancel.

4.220.4 wxTextEntryDialog::SetValue

void SetValue(const wxString& value)

Sets the default text value.

4.220.5 wxTextEntryDialog::ShowModal

int ShowModal()

Shows the dialog, returning wxID_OK if the user pressed OK, and wxOK_CANCEL otherwise.

4.221 wxTextDropTarget

A predefined drop target for dealing with text data.

Derived from

wxDropTarget (p. 218)

Include files

<wx/dnd.h>

See also

Drag and drop overview (p. 975), *wxDropSource* (p. 216), *wxDropTarget* (p. 218), *wxFileDropTarget* (p. 252)

4.221.1 wxTextDropTarget::wxTextDropTarget**wxTextDropTarget()**

Constructor.

4.221.2 wxTextDropTarget::GetFormatCount**virtual size_t GetFormatCount() const**

See *wxDropTarget::GetFormatCount* (p. 219). This function is implemented appropriately for text.

4.221.3 wxTextDropTarget::GetFormat**virtual wxDataFormat GetFormat(size_t n) const**

See *wxDropTarget::GetFormat* (p. 219). This function is implemented appropriately for text.

4.221.4 wxTextDropTarget::OnDrop**virtual bool OnDrop(long x, long y, const void *data, size_t size)**

See *wxDropTarget::OnDrop* (p. 219). This function is implemented appropriately for text, and calls *wxTextDropTarget::OnDropText* (p. 727).

4.221.5 wxTextDropTarget::OnDropText**virtual bool OnDropText(long x, long y, const char *data)**

Override this function to receive dropped text.

Parameters

x
The x coordinate of the mouse.

y
The y coordinate of the mouse.

data
The data being dropped: a NULL-terminated string.

Return value

Return TRUE to accept the data, FALSE to veto the operation.

4.222 wxTextValidator

wxTextValidator validates text controls, providing a variety of filtering behaviours.

For more information, please see *Validator overview* (p. 969).

Derived from

wxValidator (p. 781)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/valtext.h>

See also

Validator overview (p. 969), *wxValidator* (p. 781), *wxGenericValidator* (p. 295)

4.222.1 wxTextValidator::wxTextValidator

wxTextValidator(const wxTextValidator& *validator*)

Copy constructor.

wxTextValidator(long *style* = wxFILTER_NONE, wxString* *valPtr* = NULL)

Constructor, taking a style and optional pointer to a wxString variable.

Parameters

style

A bitlist of flags, which can be:

wxFILTER_NONE	No filtering takes place.
wxFILTER_ASCII	Non-ASCII characters are filtered out.
wxFILTER_ALPHA	Non-alpha characters are filtered out.
wxFILTER_ALPHANUMERIC	Non-alphanumeric characters are filtered out.
wxFILTER_NUMERIC	Non-numeric characters are filtered out.
wxFILTER_INCLUDE_LIST	Use an include list. The validator checks if the user input is on the list, complaining if not.
wxFILTER_EXCLUDE_LIST	Use an exclude list. The validator checks if the user input is on the list, complaining if it is.

valPtr

A pointer to a `wxString` variable that contains the value. This variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window). If `NULL`, the validator uses its own internal storage for the value.

4.222.2 `wxTextValidator::~~wxTextValidator`

`~wxTextValidator()`

Destructor.

4.222.3 `wxTextValidator::Clone`

`virtual wxValidator* Clone() const`

Clones the text validator using the copy constructor.

4.222.4 `wxTextValidator::GetExcludeList`

`wxStringList& GetExcludeList() const`

Returns a reference to the exclude list (the list of invalid values).

4.222.5 `wxTextValidator::GetIncludeList`

`wxStringList& GetIncludeList() const`

Returns a reference to the include list (the list of valid values).

4.222.6 `wxTextValidator::GetStyle`

`long GetStyle() const`

Returns the validator style.

4.222.7 `wxTextValidator::OnChar`

`void OnChar(wxKeyEvent& event)`

Receives character input from the window and filters it according to the current validator style.

4.222.8 wxTextValidator::SetExcludeList**void SetExcludeList(const wxStringList& *stringList*)**

Sets the exclude list (invalid values for the user input).

4.222.9 wxTextValidator::SetIncludeList**void SetIncludeList(const wxStringList& *stringList*)**

Sets the include list (valid values for the user input).

4.222.10 wxTextValidator::SetStyle**void SetStyle(long *style*)**

Sets the validator style.

4.222.11 wxTextValidator::TransferFromWindow**virtual bool TransferToWindow(wxWindow* *parent*)**

Transfers the string value to the window.

4.222.12 wxTextValidator::TransferToWindow**virtual bool TransferToWindow(wxWindow* *parent*)**

Transfers the window value to the string.

4.222.13 wxTextValidator::Validate**virtual bool Validate(wxWindow* *parent*)**

Validates the window contents against the include or exclude lists, depending on the validator style.

4.223 wxTextFile

The wxTextFile is a simple class which allows to work with text files on line by line basis. It also understands the differences in line termination characters under different platforms and will not do anything bad to files with "non native" line termination sequences - in fact, it can be also used to modify the text files and change the line

termination characters from one type (say DOS) to another (say Unix).

One word of warning: the class is not at all optimized for big files and so it will load the file entirely into memory when opened. Of course, you should not work in this way with large files (as an estimation, anything over 1 Megabyte is surely too big for this class). On the other hand, it is not a serious limitation for the small files like configuration files or programs sources which are well handled by `wxTextFile`.

The typical things you may do with `wxTextFile` in order are:

- Create and open it: this is done with *Open* (p. 732) function which opens the file (name may be specified either as *Open* argument or in the constructor), reads its contents in memory and closes it. If all of these operations are successful, *Open()* will return `TRUE` and `FALSE` on error.
- Work with the lines in the file: this may be done either with "direct access" functions like *GetLineCount* (p. 733) and *GetLine* (p. 733) (*operator[]* does exactly the same but looks more like array addressing) or with "sequential access" functions which include *GetFirstLine* (p. 734)/*GetNextLine* (p. 734) and also *GetLastLine* (p. 734)/*GetPrevLine* (p. 734). For the sequential access functions the current line number is maintained: it is returned by *GetCurrentLine* (p. 733) and may be changed with *GoToLine* (p. 733).
- Add/remove lines to the file: *AddLine* (p. 735) and *InsertLine* (p. 735) add new lines while *RemoveLine* (p. 735) deletes the existing ones.
- Save your changes: notice that the changes you make to the file will **not** be saved automatically; calling *Close* (p. 732) or doing nothing discards them! To save the changes you must explicitly call *Write* (p. 735) - here, you may also change the line termination type if you wish.

Derived from

No base class

Include files

<wx/textfile.h>

Data structures

The following constants identify the line termination type:

```
enum wxTextFileType
{
    wxTextFileType_None,    // incomplete (the last line of the file only)
    wxTextFileType_Unix,    // line is terminated with 'LF' = 0xA = 10 =
    '\n'
    wxTextFileType_Dos,     // 'CR' 'LF'
    wxTextFileType_Mac      // 'CR' = 0xD = 13 =
    '\r'
};
```

See also

wxFile (p. 241)

4.223.1 wxTextFile::wxTextFile**wxTextFile() const**

Default constructor, use `Open(string)` to initialize the object.

4.223.2 wxTextFile::wxTextFile**wxTextFile(const wxString& strFile) const**

Constructor does not load the file into memory, use `Open()` to do it.

4.223.3 wxTextFile::Exists**bool Exists() const**

Return TRUE if file exists - the name of the file should have been specified in the constructor before calling `Exists()`.

4.223.4 wxTextFile::Open**bool Open() const**

`Open()` opens the file with the name which was given in the *constructor* (p. 732) and also loads file in memory on success.

4.223.5 wxTextFile::Open**bool Open(const wxString& strFile) const**

Same as *Open()* (p. 732) but allows to specify the file name (must be used if the default constructor was used to create the object).

4.223.6 wxTextFile::Close**bool Close() const**

Closes the file and frees memory, **losing all changes**. Use *Write()* (p. 735) if you want to save them.

4.223.7 wxTextFile::IsOpened**bool IsOpened() const**

Returns TRUE if the file is currently opened.

4.223.8 wxTextFile::GetLineCount**size_t GetLineCount() const**

Get the number of lines in the file.

4.223.9 wxTextFile::GetLine**wxString& GetLine(size_t n) const**

Retrieves the line number *n* from the file. The returned line may be modified but you shouldn't add line terminator at the end - this will be done by `wxTextFile`.

4.223.10 wxTextFile::operator[]**wxString& operator[](size_t n) const**

The same as *GetLine* (p. 733).

4.223.11 wxTextFile::GetCurrentLine**size_t GetCurrentLine() const**

Returns the current line: it has meaning only when you're using `GetFirstLine()/GetNextLine()` functions, it doesn't get updated when you're using "direct access" functions like `GetLine()`. `GetFirstLine()` and `GetLastLine()` also change the value of the current line, as well as `GoToLine()`.

4.223.12 wxTextFile::GoToLine**void GoToLine(size_t n) const**

Changes the value returned by *GetCurrentLine* (p. 733) and used by *GetFirstLine()* (p. 734)/*GetNextLine()* (p. 734).

4.223.13 wxTextFile::Eof**bool Eof() const**

Returns TRUE if the current line is the last one.

4.223.14 wxTextFile::GetFirstLine

wxString& GetFirstLine() const

This method together with *GetNextLine()* (p. 734) allows more "iterator-like" traversal of the list of lines, i.e. you may write something like:

```
for ( str = GetFirstLine(); !Eof(); str = GetNextLine() )
{
    // do something with the current line in str
}
```

4.223.15 wxTextFile::GetNextLine

wxString& GetNextLine()

Gets the next line (see *GetFirstLine* (p. 734) for the example).

4.223.16 wxTextFile::GetPrevLine

wxString& GetPrevLine()

Gets the previous line in the file.

4.223.17 wxTextFile::GetLastLine

wxString& GetLastLine()

Gets the last line of the file.

4.223.18 wxTextFile::GetLineType

wxTextFileType GetLineType(size_t n) const

Get the type of the line (see also *GetEOL* (p. 735))

4.223.19 wxTextFile::GuessType

wxTextFileType GuessType() const

Guess the type of file (which is supposed to be opened). If sufficiently many lines of the file are in DOS/Unix/Mac format, the corresponding value will be returned. If the

detection mechanism fails `wxTextFileType_None` is returned.

4.223.20 `wxTextFile::GetName`

const char* GetName() const

Get the name of the file.

4.223.21 `wxTextFile::AddLine`

void AddLine(const wxString& str, wxTextFileType type = typeDefault) const

Adds a line to the end of file.

4.223.22 `wxTextFile::InsertLine`

void InsertLine(const wxString& str, size_t n, wxTextFileType type = typeDefault) const

Insert a line before the line number *n*.

4.223.23 `wxTextFile::RemoveLine`

void RemoveLine(size_t n) const

Delete line number *n* from the file.

4.223.24 `wxTextFile::Write`

bool Write(wxTextFileType typeNew = wxTextFileType_None) const

Change the file on disk. The *typeNew* parameter allows you to change the file format (default argument means "don't change type") and may be used to convert, for example, DOS files to Unix.

Returns TRUE if operation succeeded, FALSE if it failed.

4.223.25 `wxTextFile::GetEOL`

static const char* GetEOL(wxTextFileType type = typeDefault) const

Get the line termination string corresponding to given constant. *typeDefault* is the value defined during the compilation and corresponds to the native format of the platform, i.e. it will be `wxTextFileType_Dos` under Windows, `wxTextFileType_Unix` under Unix and

wxTextFileType_Mac under Mac.

4.223.26 wxTextFile::~wxTextFile

~wxTextFile() const

Destructor does nothing.

4.224 wxThread

A thread is basically a path of execution through a program. Threads are also sometimes called *light-weight processes*, but the fundamental difference between threads and processes is that memory spaces of different processes are separated while all threads share the same address space. While it makes it much easier to share common data between several threads, it also makes much easier to shoot oneself in the foot, so careful use of synchronization objects such as *mutexes* (p. 459) and/or *critical sections* (p. 126) is recommended.

Derived from

None.

Include files

<wx/thread.h>

See also

wxMutex (p. 459), *wxCondition* (p. 110), *wxCriticalSection* (p. 126)

4.224.1 wxThread::wxThread

wxThread()

Default constructor: it doesn't create nor starts the thread.

4.224.2 wxThread::~~wxThread

~wxThread()

wxThread destructor is private, so you can not call it directly - i.e., deleting wxThread objects is forbidden. Instead, you should use *Delete* (p. 737) or *Kill* (p. 738) methods. This also means that thread objects should be **always** allocated on the heap (i.e. with *new*) because the functions mentioned above will try to reclaim the storage from the heap.

4.224.3 wxThread::Create

wxThreadError Create()

Creates a new thread. The thread object is created in the suspended state, you should call *Run* (p. 739) to start running it.

Return value

One of:

wxTHREAD_NO_ERROR	There was no error.
wxTHREAD_NO_RESOURCE	There were insufficient resources to create a new thread.
wxTHREAD_RUNNING	The thread is already running.

4.224.4 wxThread::Delete

Delete()

This function should be called to terminate this thread. Unlike *Kill* (p. 738), it gives the target thread the time to terminate gracefully. Because of this, however, this function may not return immediately and if the thread is "hung" won't return at all. Also, message processing is not stopped during this function execution, so the message handlers may be called from inside it.

Delete() may be called for thread in any state: running, paused or even not yet created. Moreover, it must be called if *Create* (p. 737) or *Run* (p. 739) fail to free the memory occupied by the thread object.

4.224.5 wxThread::GetID

unsigned long GetID() const

Gets the thread identifier: this is a platform dependent number which uniquely identifies the thread throughout the system during its existence (i.e. the thread identifiers may be reused).

4.224.6 wxThread::GetPriority

int GetPriority() const

Gets the priority of the thread, between zero and 100.

The following priorities are already defined:

WXTHREAD_MIN_PRIORITY	0
WXTHREAD_DEFAULT_PRIORITY	50
WXTHREAD_MAX_PRIORITY	100

4.224.7 wxThread::IsAlive

bool IsAlive() const

Returns TRUE if the thread is alive (i.e. started and not terminating).

4.224.8 wxThread::IsMain

bool IsMain() const

Returns TRUE if the calling thread is the main application thread.

4.224.9 wxThread::IsPaused

bool IsPaused() const

Returns TRUE if the thread is paused.

4.224.10 wxThread::IsRunning

bool IsRunning() const

Returns TRUE if the thread is running.

4.224.11 wxThread::Kill

wxThreadError Kill()

Immediately terminates the target thread. **This function is dangerous and should be used with extreme care (and not used at all whenever possible)!** The resources allocated to the thread will not be freed and the state of the C runtime library may become inconsistent. Use *Delete()* (p. 737) instead.

4.224.12 wxThread::OnExit

void OnExit()

Called when the thread exits. This function is called in the context of the thread associated with the `wxThread` object, not in the context of the main thread.

4.224.13 `wxThread::Run`

`wxThreadError Run()`

Runs the thread.

4.224.14 `wxThread::SetPriority`

`void SetPriority(int priority)`

Sets the priority of the thread, between zero and 100. This must be set before the thread is created.

The following priorities are already defined:

<code>WXTHREAD_MIN_PRIORITY</code>	0
<code>WXTHREAD_DEFAULT_PRIORITY</code>	50
<code>WXTHREAD_MAX_PRIORITY</code>	100

4.224.15 `wxThread::Sleep`

`Sleep(unsigned long milliseconds)`

Pauses the thread execution for the given amount of time.

This function should be used instead of `wxSleep` (p. 873) by all worker (i.e. all except the main one) threads.

4.224.16 `wxThread::This`

`wxThread * This()`

Return the thread object for the calling thread. NULL is returned if the calling thread is the main (GUI) thread, but *IsMain* (p. 738) should be used to test whether the thread is really the main one because NULL may also be returned for the thread not created with `wxThread` class. Generally speaking, the return value for such thread is undefined.

4.224.17 `wxThread::Yield`

`Yield()`

Give the rest of the thread time slice to the system allowing the other threads to run. See also *Sleep()* (p. 739).

4.225 wxTime

Representation of time and date.

NOTE: this class should be used with caution, since it is not fully tested. It will be replaced with a new wxDateTime class in the near future.

Derived from

wxObject (p. 471)

Include files

<wx/time.h>

Data structures

```
typedef unsigned short hourTy;
```

```
typedef unsigned short minuteTy;
```

```
typedef unsigned short secondTy;
```

```
typedef unsigned long clockTy;
```

```
enum tFormat    wx12h, wx24h ;
```

```
enum tPrecision  wxStdMinSec, wxStdMin ;
```

See also

wxDate (p. 143)

4.225.1 wxTime::wxTime**wxTime()**

Initialize the object using the current time.

wxTime(clockTy s)

Initialize the object using the number of seconds that have elapsed since ???.

wxTime(const wxTime& time)

Copy constructor.

wxTime(hourTy h, minuteTy m, secondTy s = 0, bool dst = FALSE)

Initialize using hours, minutes, seconds, and whether DST time.

wxTime(const wxDate& date, hourTy h = 0, minuteTy m = 0, secondTy s = 0, bool dst = FALSE)

Initialize using a *wxDate* (p. 143) object, hours, minutes, seconds, and whether DST time.

4.225.2 wxTime::GetDay**int GetDay() const**

Returns the day of the month.

4.225.3 wxTime::GetDayOfWeek**int GetDayOfWeek() const**

Returns the day of the week, a number from 0 to 6 where 0 is Sunday and 6 is Saturday.

4.225.4 wxTime::GetHour**hourTy GetHour() const**

Returns the hour in local time.

4.225.5 wxTime::GetHourGMT

hourTy GetHourGMT() const

Returns the hour in GMT.

4.225.6 wxTime::GetMinute**minuteTy GetMinute() const**

Returns the minute in local time.

4.225.7 wxTime::GetMinuteGMT**minuteTy GetMinuteGMT() const**

Returns the minute in GMT.

4.225.8 wxTime::GetMonth**int GetMonth() const**

Returns the month.

4.225.9 wxTime::GetSecond**secondTy GetSecond() const**

Returns the second in local time or GMT.

4.225.10 wxTime::GetSecondGMT**secondTy GetSecondGMT() const**

Returns the second in GMT.

4.225.11 wxTime::GetSeconds**clockTy GetSeconds() const**

Returns the number of seconds since ???.

4.225.12 wxTime::GetYear

int GetYear() const

Returns the year.

4.225.13 wxTime::FormatTime

char* FormatTime() const

Formats the time according to the current formatting options: see *wxTime::SetFormat* (p. 743).

4.225.14 wxTime::IsBetween

bool IsBetween(const wxTime& a, const wxTime& b) const

Returns TRUE if this time is between the two given times.

4.225.15 wxTime::Max

wxTime Max(const wxTime& time) const

Returns the maximum of the two times.

4.225.16 wxTime::Min

wxTime Min(const wxTime& time) const

Returns the minimum of the two times.

4.225.17 wxTime::SetFormat

static void SetFormat(const tFormat format = wx12h, const tPrecision precision = wxStdMinSec)

Sets the format and precision.

4.225.18 wxTime::operator char*

operator char*()

Returns a pointer to a static char* containing the formatted time.

4.225.19 wxTime::operator wxDate

operator wxDate() const

Converts the wxTime into a wxDate.

4.225.20 wxTime::operator =**void operator =(const wxTime& t)**

Assignment operator.

4.225.21 wxTime::operator <**bool operator <(const wxTime& t) const**

Less than operator.

4.225.22 wxTime::operator <=**bool operator <=(const wxTime& t) const**

Less than or equal to operator.

4.225.23 wxTime::operator >**bool operator >(const wxTime& t) const**

Greater than operator.

4.225.24 wxTime::operator >=**bool operator >=(const wxTime& t) const**

Greater than or equal to operator.

4.225.25 wxTime::operator ==**bool operator ==(const wxTime& t) const**

Equality operator.

4.225.26 wxTime::operator !=

bool operator !=(const wxTime& t) const

Inequality operator.

4.225.27 wxTime::operator +

bool operator +(long sec) const

Addition operator.

4.225.28 wxTime::operator -

bool operator -(long sec) const

Subtraction operator.

4.225.29 wxTime::operator +=

bool operator +=(long sec) const

Increment operator.

4.225.30 wxTime::operator -=

bool operator -=(long sec) const

Decrement operator.

4.226 wxTimer

The `wxTimer` class allows you to execute code at specified intervals. To use it, derive a new class and override the **Notify** member to perform the required action. Start with **Start**, stop with **Stop**, it's as simple as that.

Derived from

wxObject (p. 471)

Include files

<wx/timer.h>

See also

::wxStartTimer (p. 874), *::wxGetElapsedTime* (p. 868)

4.226.1 wxTimer::wxTimer**wxTimer()**

Constructor.

4.226.2 wxTimer::~~wxTimer**~wxTimer()**

Destructor. Stops the timer if activated.

4.226.3 wxTimer::Interval**int Interval()**

Returns the current interval for the timer.

4.226.4 wxTimer::Notify**void Notify()**

This member should be overridden by the user. It is called on timeout.

4.226.5 wxTimer::Start**bool Start(int milliseconds = -1, bool oneShot=FALSE)**

(Re)starts the timer. If *milliseconds* is absent or -1, the previous value is used. Returns FALSE if the timer could not be started, TRUE otherwise (in MS Windows timers are a limited resource).

If *oneShot* is FALSE (the default), the Notify function will be repeatedly called. If TRUE, Notify will be called only once.

4.226.6 wxTimer::Stop**void Stop()**

Stops the timer.

4.227 wxToolBar

The name `wxToolBar` is defined to be a synonym for one of the following classes:

- **wxToolBar95** The native Windows 95 toolbar. Used on Windows 95, NT 4 and above.
- **wxToolBarMSW** A Windows implementation. Used on 16-bit Windows.
- **wxToolBarGTK** The GTK toolbar.
- **wxToolBarSimple** A simple implementation, with scrolling. Used on platforms with no native toolbar control, or where scrolling is required.

Note that the base class **wxToolBarBase** defines automatic scrolling management functionality which is identical to *wxScrolledWindow* (p. 586), so please refer to this class also. Not all toolbars support scrolling, but `wxToolBarSimple` does.

Derived from

`wxToolBarBase`
wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

`<wx/toolbar.h>` (to allow `wxWindows` to select an appropriate toolbar class)
`<wx/tbarbase.h>` (the base class)
`<wx/tbarmsw.h>` (the non-Windows 95 Windows toolbar class)
`<wx/tbar95.h>` (the Windows 95/98 toolbar class)
`<wx/tbarsmpl.h>` (the generic simple toolbar class)

Remarks

You may also create a toolbar that is managed by the frame, by calling *wxFrame::CreateToolBar* (p. 279).

wxToolBar95: Note that this toolbar paints tools to reflect user-selected colours. The toolbar orientation must always be **wxVERTICAL**.

Window styles

wxTB_FLAT	Gives the toolbar a flat look ('coolbar' or 'flatbar' style). Windows 95 only.
wxTB_HORIZONTAL	Specifies horizontal layout.
wxTB_VERTICAL	Specifies vertical layout (not available for the Windows 95 toolbar).
wxTB_3DBUTTONS	Gives <code>wxToolBarSimple</code> a mild 3D look to its buttons.

See also *window styles overview* (p. 959).

Event handling

The toolbar class emits menu commands in the same way that a frame menubar does, so you can use one `EVT_MENU` macro for both a menu item and a toolbar button. The event handler functions take a `wxCommandEvent` argument. For most event macros, the identifier of the tool is passed, but for `EVT_TOOL_ENTER` the toolbar window is passed and the tool id is retrieved from the `wxCommandEvent`. This is because the id may be -1 when the mouse moves off a tool, and -1 is not allowed as an identifier in the event system.

<code>EVT_TOOL(id, func)</code>	Process a <code>wxEVT_COMMAND_TOOL_CLICKED</code> event (a synonym for <code>wxEVT_COMMAND_MENU_SELECTED</code>). Pass the id of the tool.
<code>EVT_MENU(id, func)</code>	The same as <code>EVT_TOOL</code> .
<code>EVT_TOOL_RANGE(id1, id2, func)</code>	Process a <code>wxEVT_COMMAND_TOOL_CLICKED</code> event for a range of id identifiers. Pass the ids of the tools.
<code>EVT_MENU_RANGE(id1, id2, func)</code>	The same as <code>EVT_TOOL_RANGE</code> .
<code>EVT_TOOL_RCLICKED(id, func)</code>	Process a <code>wxEVT_COMMAND_TOOL_RCLICKED</code> event. Pass the id of the tool.
<code>EVT_TOOL_RCLICKED_RANGE(id1, id2, func)</code>	Process a <code>wxEVT_COMMAND_TOOL_RCLICKED</code> event for a range of ids. Pass the ids of the tools.
<code>EVT_TOOL_ENTER(id, func)</code>	Process a <code>wxEVT_COMMAND_TOOL_ENTER</code> event. Pass the id of the toolbar itself. The value of <code>wxCommandEvent::GetSelection</code> is the tool id, or -1 if the mouse cursor has moved off a tool.

See also

Toolbar overview (p. 963), *wxScrolledWindow* (p. 586)

4.227.1 `wxToolBar::wxToolBar`

`wxToolBar()`

Default constructor.

`wxToolBar(wxWindow* parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxTB_HORIZONTAL | wxNO_BORDER, const wxString& name = wxPanelNameStr)`

Constructs a toolbar.

Parameters

parent

Pointer to a parent window.

id

Window identifier. If -1, will automatically create an identifier.

pos

Window position. `wxDefaultPosition` is (-1, -1) which indicates that `wxWindows` should generate a default position for the window. If using the `wxWindow` class directly, supply an actual position.

size

Window size. `wxDefaultSize` is (-1, -1) which indicates that `wxWindows` should generate a default size for the window.

style

Window style. See `wxToolBar` (p. 746) for details.

name

Window name.

Remarks

After a toolbar is created, you use `wxToolBar::AddTool` (p. 750) and perhaps `wxToolBar::AddSeparator` (p. 749), and then you must call `wxToolBar::Realize` (p. 757) to construct and display the toolbar tools.

You may also create a toolbar that is managed by the frame, by calling `wxFrame::CreateToolBar` (p. 279).

4.227.2 `wxToolBar::~~wxToolBar`

void `~wxToolBar()`

Toolbar destructor.

4.227.3 `wxToolBar::AddSeparator`

void `AddSeparator()`

Adds a separator for spacing groups of tools.

See also

wxToolBar::AddTool (p. 750), *wxToolBar::SetToolSeparation* (p. 760)

4.227.4 wxToolBar::AddTool

wxToolBarTool* AddTool(int *toolIndex*, const **wxBitmap&** *bitmap1*, const **wxBitmap&** *bitmap2* = *wxNullBitmap*, **bool** *isToggle* = *FALSE*, **long** *xPos* = -1, **long** *yPos* = -1, **wxObject*** *clientData* = *NULL*, const **wxString&** *shortHelpString* = "", const **wxString&** *longHelpString* = "")

Adds a tool to the toolbar.

Parameters

toolIndex

An integer by which the tool may be identified in subsequent operations.

isToggle

Specifies whether the tool is a toggle or not: a toggle tool may be in two states, whereas a non-toggle tool is just a button.

bitmap1

The primary tool bitmap for toggle and button tools.

bitmap2

The second bitmap specifies the on-state bitmap for a toggle tool. If this is *NULL*, either an inverted version of the primary bitmap is used for the on-state of a toggle tool (monochrome displays) or a black border is drawn around the tool (colour displays). Note that to pass a *NULL* value, you need to cast it to (*wxBitmap **) so that C++ can construct an appropriate temporary *wxBitmap* object.

xPos

Specifies the x position of the tool if automatic layout is not suitable.

yPos

Specifies the y position of the tool if automatic layout is not suitable.

clientData

An optional pointer to client data which can be retrieved later using *wxToolBar::GetToolClientData* (p. 753).

shortHelpString

Used for displaying a tooltip for the tool in the Windows 95 implementation of *wxButtonBar*. Pass the empty string if this is not required.

longHelpString

Used to display longer help, such as status line help. Pass the empty string if this is not required.

Remarks

After you have added tools to a toolbar, you must call `wxToolBar::Realize` (p. 757) in order to have the tools appear.

See also

`wxToolBar::AddSeparator` (p. 749), `wxToolBar::Realize` (p. 757),

4.227.5 `wxToolBar::CreateTools`

bool CreateTools()

This function is implemented for some toolbar classes to create the tools and display them. The portable way of calling it is to call `wxToolBar::Realize` (p. 757) after you have added tools and separators.

See also

`wxToolBar::AddTool` (p. 750), `wxToolBar::Realize` (p. 757)

4.227.6 `wxToolBar::DrawTool`

void DrawTool(wxMemoryDC& memDC, wxToolBarTool* tool)

Draws the specified tool onto the window using the given memory device context.

Parameters

memDC

A memory DC to be used for drawing the tool.

tool

Tool to be drawn.

Remarks

For internal use only.

4.227.7 `wxToolBar::EnableTool`

void EnableTool(int toolIndex, const bool enable)

Enables or disables the tool.

Parameters

toolIndex

Tool to enable or disable.

enable

If TRUE, enables the tool, otherwise disables it.

Remarks

For `wxToolBarSimple`, does nothing. Some other implementations will change the visible state of the tool to indicate that it is disabled.

See also

`wxToolBar::GetToolEnabled` (p. 754), `wxToolBar::ToggleTool` (p. 760)

4.227.8 `wxToolBar::FindToolForPosition`

`wxToolBarTool* FindToolForPosition(const float x, const float y) const`

Finds a tool for the given mouse position.

Parameters

x
X position.

y
Y position.

Return value

A pointer to a tool if a tool is found, or NULL otherwise.

Remarks

Used internally, and should not need to be used by the programmer.

4.227.9 `wxToolBar::GetToolSize`

`wxSize GetToolSize()`

Returns the size of a whole button, which is usually larger than a tool bitmap because of added 3D effects.

See also

`wxToolBar::SetToolBitmapSize` (p. 758), `wxToolBar::GetToolBitmapSize` (p. 752)

4.227.10 `wxToolBar::GetToolBitmapSize`

wxSize GetToolBitmapSize()

Returns the size of bitmap that the toolbar expects to have. The default bitmap size is 16 by 15 pixels.

Remarks

Note that this is the size of the bitmap you pass to *wxToolBar::AddTool* (p. 750), and not the eventual size of the tool button.

See also

wxToolBar::SetToolBitmapSize (p. 758), *wxToolBar::GetToolSize* (p. 752)

4.227.11 wxToolBar::GetMargins**wxSize GetMargins() const**

Returns the left/right and top/bottom margins, which are also used for inter-toolspacing.

See also

wxToolBar::SetMargins (p. 758)

4.227.12 wxToolBar::GetMaxSize**void GetMaxSize(float* w, float* h) const**

Gets the maximum size taken up by the tools after layout, including margins. This can be used to size a frame around the toolbar window.

Parameters

w
Receives the maximum horizontal size.

h
Receives the maximum vertical size.

4.227.13 wxToolBar::GetToolClientData**wxObject* GetToolClientData(int toolIndex) const**

Get any client data associated with the tool.

Parameters

toolIndex

Index of the tool, as passed to *wxToolBar::AddTool* (p. 750).

Return value

Client data, or NULL if there is none.

4.227.14 **wxToolBar::GetToolEnabled**

bool GetToolEnabled(int *toolIndex*) const

Called to determine whether a tool is enabled (responds to user input).

Parameters

toolIndex

Index of the tool in question.

Return value

TRUE if the tool is enabled, FALSE otherwise.

4.227.15 **wxToolBar::GetToolLongHelp**

wxString GetToolLongHelp(int *toolIndex*) const

Returns the long help for the given tool.

Parameters

toolIndex

The tool in question.

See also

wxToolBar::SetToolLongHelp (p. 759), *wxToolBar::SetToolShortHelp* (p. 759)

4.227.16 **wxToolBar::GetToolPacking**

int GetToolPacking() const

Returns the value used for packing tools.

See also

wxToolBar::SetToolPacking (p. 759)

4.227.17 wxToolBar::GetToolSeparation**int GetToolSeparation() const**

Returns the default separator size.

See also

wxToolBar::SetToolSeparation (p. 760)

4.227.18 wxToolBar::GetToolShortHelp**wxString GetToolShortHelp(int *toolIndex*) const**

Returns the short help for the given tool.

Returns the long help for the given tool.

Parameters

toolIndex

The tool in question.

See also

wxToolBar::GetToolLongHelp (p. 754), *wxToolBar::SetToolShortHelp* (p. 759)

4.227.19 wxToolBar::GetToolState**bool GetToolState(int *toolIndex*) const**

Gets the on/off state of a toggle tool.

Parameters

toolIndex

The tool in question.

Return value

TRUE if the tool is toggled on, FALSE otherwise.

4.227.20 wxToolBar::Layout**void Layout()**

Called by the application after the tools have been added to automatically lay the tools out on the window. If you have given absolute positions when adding the tools, do not call this.

This function is only implemented for some toolbar classes. The portable way of calling it is to call *wxToolBar::Realize* (p. 757) after you have added tools and separators.

See also

wxToolBar::AddTool (p. 750), *wxToolBar::Realize* (p. 757)

4.227.21 **wxToolBar::OnLeftClick**

bool OnLeftClick(int *toolIndex*, bool *toggleDown*)

Called when the user clicks on a tool with the left mouse button.

This is the old way of detecting tool clicks; although it will still work, you should use the EVT_MENU or EVT_TOOL macro instead.

Parameters

toolIndex

The identifier passed to *wxToolBar::AddTool* (p. 750).

toggleDown

TRUE if the tool is a toggle and the toggle is down, otherwise is FALSE.

Return value

If the tool is a toggle and this function returns FALSE, the toggle toggle state (internal and visual) will not be changed. This provides a way of specifying that toggle operations are not permitted in some circumstances.

See also

wxToolBar::OnMouseEnter (p. 756), *wxToolBar::OnRightClick* (p. 757)

4.227.22 **wxToolBar::OnMouseEnter**

void OnMouseEnter(int *toolIndex*)

This is called when the mouse cursor moves into a tool or out of the toolbar.

This is the old way of detecting mouse enter events; although it will still work, you should use the EVT_TOOL_ENTER macro instead.

Parameters

toolIndex

Greater than -1 if the mouse cursor has moved into the tool, or -1 if the mouse cursor has moved. The programmer can override this to provide extra information about the tool, such as a short description on the status line.

Remarks

With some derived toolbar classes, if the mouse moves quickly out of the toolbar, wxWindows may not be able to detect it. Therefore this function may not always be called when expected.

4.227.23 **wxToolBar::OnRightClick**

void OnRightClick(int toolIndex, float x, float y)

Called when the user clicks on a tool with the right mouse button. The programmer should override this function to detect right tool clicks.

This is the old way of detecting tool right clicks; although it will still work, you should use the EVT_TOOL_RCLICKED macro instead.

Parameters

toolIndex

The identifier passed to *wxToolBar::AddTool* (p. 750).

x

The x position of the mouse cursor.

y

The y position of the mouse cursor.

Remarks

A typical use of this member might be to pop up a menu.

See also

wxToolBar::OnMouseEnter (p. 756), *wxToolBar::OnLeftClick* (p. 756)

4.227.24 **wxToolBar::Realize**

bool Realize()

This function should be called after you have added tools. It calls, according to the implementation, either *wxToolBar::CreateTools* (p. 751) or *wxToolBar::Layout* (p. 755).

If you are using absolute positions for your tools when using a *wxToolBarSimple* object,

do not call this function. You must call it at all other times.

4.227.25 **wxToolBar::SetToolBitmapSize**

void SetToolBitmapSize(const wxSize& size)

Sets the default size of each tool bitmap. The default bitmap size is 16 by 15 pixels.

Parameters

size

The size of the bitmaps in the toolbar.

Remarks

This should be called to tell the toolbar what the tool bitmap size is. Call it before you add tools.

Note that this is the size of the bitmap you pass to *wxToolBar::AddTool* (p. 750), and not the eventual size of the tool button.

See also

wxToolBar::GetToolBitmapSize (p. 752), *wxToolBar::GetToolSize* (p. 752)

4.227.26 **wxToolBar::SetMargins**

void SetMargins(const wxSize& size)

void SetMargins(int x, int y)

Set the values to be used as margins for the toolbar.

Parameters

size

Margin size.

x

Left margin, right margin and inter-tool separation value.

y

Top margin, bottom margin and inter-tool separation value.

Remarks

This must be called before the tools are added if absolute positioning is to be used, and the default (zero-size) margins are to be overridden.

See also

wxToolBar::GetMargins (p. 753), *wxSize* (p. 595)

4.227.27 wxToolBar::SetToolLongHelp

void SetToolLongHelp(int *toolIndex*, const wxString& *helpString*)

Sets the long help for the given tool.

Parameters

toolIndex

The tool in question.

helpString

A string for the long help.

Remarks

You might use the long help for displaying the tool purpose on the status line.

See also

wxToolBar::GetToolLongHelp (p. 754), *wxToolBar::SetToolShortHelp* (p. 759),

4.227.28 wxToolBar::SetToolPacking

void SetToolPacking(int *packing*)

Sets the value used for spacing tools. The default value is 1.

Parameters

packing

The value for packing.

Remarks

The packing is used for spacing in the vertical direction if the toolbar is horizontal, and for spacing in the horizontal direction if the toolbar is vertical.

See also

wxToolBar::GetToolPacking (p. 754)

4.227.29 wxToolBar::SetToolShortHelp

void SetToolShortHelp(int *toolIndex*, const wxString& *helpString*)

Sets the short help for the given tool.

Parameters

toolIndex

The tool in question.

helpString

The string for the short help.

Remarks

An application might use short help for identifying the tool purpose in a tooltip.

See also

wxToolBar::GetToolShortHelp (p. 755), *wxToolBar::SetToolLongHelp* (p. 759)

4.227.30 wxToolBar::SetToolSeparation

void SetToolSeparation(int *separation*)

Sets the default separator size. The default value is 5.

Parameters

separation

The separator size.

See also

wxToolBar::AddSeparator (p. 749)

4.227.31 wxToolBar::ToggleTool

void ToggleTool(int *toolIndex*, const bool *toggle*)

Toggles a tool on or off.

Parameters

toolIndex

Tool in question.

toggle

If TRUE, toggles the tool on, otherwise toggles it off.

Remarks

Only applies to a tool that has been specified as a toggle tool.

See also

wxToolBar::GetToolState (p. 755)

4.228 wxTreeCtrl

A tree control presents information as a hierarchy, with items that may be expanded to show further items. Items in a tree control are referenced by *wxTreeItemId* handles.

To intercept events from a tree control, use the event table macros described in *wxTreeEvent* (p. 774).

Derived from

wxControl (p. 125)
wxWindow (p. 798)
wxEvtHandler (p. 224)
wxObject (p. 471)

Include files

<wx/treectrl.h>

Window styles

wxTR_HAS_BUTTONS	Use this style to show + and - buttons to the left of parent items.
wxTR_EDIT_LABELS	Use this style if you wish the user to be able to edit labels in the tree control.

See also *window styles overview* (p. 959).

Event handling

To process input from a tree control, use these event handler macros to direct input to member functions that take a *wxTreeEvent* (p. 774) argument.

EVT_TREE_BEGIN_DRAG(id, func)	Begin dragging with the left mouse button.
EVT_TREE_BEGIN_RDRAG(id, func)	Begin dragging with the right mouse button.
EVT_TREE_BEGIN_LABEL_EDIT(id, func)	Begin editing a label.
EVT_TREE_END_LABEL_EDIT(id, func)	Finish editing a label.
EVT_TREE_DELETE_ITEM(id, func)	Delete an item.
EVT_TREE_GET_INFO(id, func)	Request information from the application.
EVT_TREE_SET_INFO(id, func)	Information is being supplied.
EVT_TREE_ITEM_EXPANDED(id, func)	Parent has been expanded.

EVT_TREE_ITEM_EXPANDING(id, func) Parent is being expanded.
EVT_TREE_SEL_CHANGED(id, func) Selection has changed.
EVT_TREE_SEL_CHANGING(id, func) Selection is changing.
EVT_TREE_KEY_DOWN(id, func) A key has been pressed.

See also

wxTreeItemData (p. 773), *wxTreeCtrl* overview (p. 913), *wxListBox* (p. 373), *wxListCtrl* (p. 381), *wxImageList* (p. 339), *wxTreeEvent* (p. 774)

4.228.1 wxTreeCtrl::wxTreeCtrl

wxTreeCtrl()

Default constructor.

wxTreeCtrl(wxWindow* parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxTR_HAS_BUTTONS, const wxValidator& validator = wxDefaultValidator, const wxString& name = "listCtrl")

Constructor, creating and showing a tree control.

Parameters

parent

Parent window. Must not be NULL.

id

Window identifier. A value of -1 indicates a default value.

pos

Window position.

size

Window size. If the default size (-1, -1) is specified then the window is sized appropriately.

style

Window style. See *wxTreeCtrl* (p. 761).

validator

Window validator.

name

Window name.

See also

wxTreeCtrl::Create (p. 763), *wxValidator* (p. 781)

4.228.2 wxTreeCtrl::~~wxTreeCtrl

void ~wxTreeCtrl()

Destructor, destroying the list control.

4.228.3 wxTreeCtrl::AddRoot

wxTreeItemId AddRoot(const wxString& text, int image = -1, int selImage = -1, wxTreeItemData* data = NULL)

Adds the root node to the tree, returning the new item.

If *image* > -1 and *selImage* is -1, the same image is used for both selected and unselected items.

4.228.4 wxTreeCtrl::AppendItem

wxTreeItemId AppendItem(const wxTreeItemId& parent, const wxString& text, int image = -1, int selImage = -1, wxTreeItemData* data = NULL)

Appends an item to the end of the branch identified by *parent*, return a new item id.

If *image* > -1 and *selImage* is -1, the same image is used for both selected and unselected items.

4.228.5 wxTreeCtrl::Collapse

void Collapse(const wxTreeItemId& item)

Collapses the given item.

4.228.6 wxTreeCtrl::CollapseAndReset

void CollapseAndReset(const wxTreeItemId& item)

Collapses the given item and removes all children.

4.228.7 wxTreeCtrl::Create

```
bool wxTreeCtrl(wxWindow* parent, wxWindowID id, const wxPoint& pos =  
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxTR_HAS_BUTTONS, const wxValidator& validator = wxDefaultValidator, const  
wxString& name = "listCtrl")
```

Creates the tree control. See *wxTreeCtrl::wxTreeCtrl* (p. 762) for further details.

4.228.8 wxTreeCtrl::Delete

```
void Delete(const wxTreeItemId& item)
```

Deletes the specified item.

4.228.9 wxTreeCtrl::DeleteAllItems

```
void DeleteAllItems()
```

Deletes all the items in the control.

4.228.10 wxTreeCtrl::EditLabel

```
wxTextCtrl* EditLabel(const wxTreeItemId& item, wxClassInfo* textControlClass =  
CLASSINFO(wxTextCtrl))
```

Starts editing the label of the given item, returning the text control that the tree control uses for editing.

Pass another *textControlClass* if a derived class is required. It usually will be, in order for the application to detect when editing has finished and to call *wxTreeCtrl::EndEditLabel* (p. 764).

Do not delete the text control yourself.

This function is currently supported under Windows only.

[See also](#)

wxTreeCtrl::EndEditLabel (p. 764)

4.228.11 wxTreeCtrl::EndEditLabel

```
void EndEditLabel(bool cancelEdit)
```

Ends label editing. If *cancelEdit* is TRUE, the edit will be cancelled.

This function is currently supported under Windows only.

See also

wxTreeCtrl::EditLabel (p. 764)

4.228.12 wxTreeCtrl::EnsureVisible

void EnsureVisible(const wxTreeItemId& *item*)

Scrolls and/or expands items to ensure that the given item is visible.

4.228.13 wxTreeCtrl::Expand

void Expand(const wxTreeItemId& *item*)

Expands the given item.

4.228.14 wxTreeCtrl::GetBoundingRect

bool GetBoundingRect(const wxTreeItemId& *item*, wxRect& *rect*, bool *textOnly* = FALSE) const

Retrieves the rectangle bounding the *item*. If *textOnly* is TRUE, only the rectangle around the items label will be returned, otherwise the items image is also taken into account.

The return value is TRUE if the rectangle was successfully retrieved or FALSE if it was not (in this case *rect* is not changed) - for example, if the item is currently invisible.

4.228.15 wxTreeCtrl::GetChildrenCount

size_t GetChildrenCount(const wxTreeItemId& *item*, bool *recursively* = TRUE) const

Returns the number of items in the branch. If *recursively* is TRUE, returns the total number of descendants, otherwise only one level of children is counted.

4.228.16 wxTreeCtrl::GetCount

int GetCount() const

Returns the number of items in the control.

4.228.17 wxTreeCtrl::GetEditControl

wxTextCtrl& GetEditControl() const

Returns the edit control used to edit a label.

4.228.18 wxTreeCtrl::GetFirstChild**wxTreeItemId GetFirstChild(const wxTreeItemId& item, long& cookie) const**

Returns the first child; call *wxTreeCtrl::GetNextChild* (p. 767) for the next child.

For this enumeration function you must pass in a 'cookie' parameter which is opaque for the application but is necessary for the library to make these functions reentrant (i.e. allow more than one enumeration on one and the same object simultaneously). The cookie passed to *GetFirstChild* and *GetNextChild* should be the same.

Returns an invalid tree item if there are no further children.

[See also](#)

wxTreeCtrl::GetNextChild (p. 767)

4.228.19 wxTreeCtrl::GetFirstVisibleItem**wxTreeItemId GetFirstVisibleItem() const**

Returns the first visible item.

4.228.20 wxTreeCtrl::GetImageList**wxImageList* GetImageList(int which = wxIMAGE_LIST_NORMAL) const**

Returns the specified image list. *which* may be one of:

wxIMAGE_LIST_NORMAL The normal (large icon) image list.

wxIMAGE_LIST_SMALL The small icon image list.

wxIMAGE_LIST_STATE The user-defined state image list (unimplemented).

4.228.21 wxTreeCtrl::GetIndent**int GetIndent() const**

Returns the current tree control indentation.

4.228.22 wxTreeCtrl::GetItemData

wxTreeItemData* GetItemData(const wxTreeItemId& item) const

Returns the tree item data associated with the item.

[See also](#)

wxTreeItemData (p. 773)

4.228.23 wxTreeCtrl::GetItemImage

int GetItemImage(const wxTreeItemId& item) const

Gets the normal item image.

4.228.24 wxTreeCtrl::GetItemText

wxString GetItemText(const wxTreeItemId& item) const

Returns the item label.

4.228.25 wxTreeCtrl::GetLastChild

wxTreeItemId GetLastChild(const wxTreeItemId& item) const

Returns the last child of the item (or an invalid tree item if this item has no children).

[See also](#)

GetFirstChild (p. 766), *GetLastChild* (p. 767)

4.228.26 wxTreeCtrl::GetNextChild

wxTreeItemId GetNextChild(const wxTreeItemId& item, long& cookie) const

Returns the next child; call *wxTreeCtrl::GetFirstChild* (p. 766) for the first child.

For this enumeration function you must pass in a 'cookie' parameter which is opaque for the application but is necessary for the library to make these functions reentrant (i.e. allow more than one enumeration on one and the same object simultaneously). The cookie passed to *GetFirstChild* and *GetNextChild* should be the same.

Returns an invalid tree item if there are no further children.

[See also](#)

wxTreeCtrl::GetFirstChild (p. 766)

4.228.27 **wxTreeCtrl::GetNextSibling**

wxTreeItemId GetNextSibling(const wxTreeItemId& item) const

Returns the next sibling of the specified item; call *wxTreeCtrl::GetPrevSibling* (p. 768) for the previous sibling.

Returns an invalid tree item if there are no further siblings.

[See also](#)

wxTreeCtrl::GetPrevSibling (p. 768)

4.228.28 **wxTreeCtrl::GetNextVisible**

wxTreeItemId GetNextVisible(const wxTreeItemId& item) const

Returns the next visible item.

4.228.29 **wxTreeCtrl::GetParent**

wxTreeItemId GetParent(const wxTreeItemId& item) const

Returns the item's parent.

4.228.30 **wxTreeCtrl::GetPrevSibling**

wxTreeItemId GetPrevSibling(const wxTreeItemId& item) const

Returns the previous sibling of the specified item; call *wxTreeCtrl::GetNextSibling* (p. 768) for the next sibling.

Returns an invalid tree item if there are no further children.

[See also](#)

wxTreeCtrl::GetNextSibling (p. 768)

4.228.31 **wxTreeCtrl::GetPrevVisible**

wxTreeItemId GetPrevVisible(const wxTreeItemId& item) const

Returns the previous visible item.

4.228.32 wxTreeCtrl::GetRootItem**wxTreeItemId GetRootItem() const**

Returns the root item for the tree control.

4.228.33 wxTreeCtrl::GetItemSelectedImage**int GetItemSelectedImage(const wxTreeItemId& *item*) const**

Gets the selected item image.

4.228.34 wxTreeCtrl::GetSelection**wxTreeItemId GetSelection() const**

Returns the selection, or an invalid item if there is no selection.

4.228.35 wxTreeCtrl::HitTest**long HitTest(const wxPoint& *point*, int& *flags*)**

Calculates which (if any) item is under the given point, returning extra information in *flags*. *flags* is a bitlist of the following:

wxTREE_HITTEST_ABOVE Above the client area.

wxTREE_HITTEST_BELOW Below the client area.

wxTREE_HITTEST_NOWHERE In the client area but below the last item.

wxTREE_HITTEST_ONITEMBUTTON On the button associated with an item.

wxTREE_HITTEST_ONITEMICON On the bitmap associated with an item.

wxTREE_HITTEST_ONITEMINDENT In the indentation associated with an item.

wxTREE_HITTEST_ONITEMLABEL On the label (string) associated with an item.

wxTREE_HITTEST_ONITEMRIGHT In the area to the right of an item.

wxTREE_HITTEST_ONITEMSTATEICON On the state icon for a tree view item that is in a user-defined state.

wxTREE_HITTEST_TOLEFT To the right of the client area.

wxTREE_HITTEST_TORIGHT To the left of the client area.

4.228.36 wxTreeCtrl::InsertItem**wxTreeItemId InsertItem(const wxTreeItemId& *parent*, const wxTreeItemId& *previous*, const wxString& *text*, int *image* = -1, int *selImage* = -1, wxTreeItemData* *data* = NULL)**

Inserts an item after a given one.

If *image* > -1 and *sellImage* is -1, the same image is used for both selected and unselected items.

4.228.37 wxTreeCtrl::IsBold

bool IsBold(const wxTreeItemId& item) const

Returns TRUE if the given item is in bold state.

See also: *SetItemBold* (p. 771)

4.228.38 wxTreeCtrl::IsExpanded

bool IsExpanded(const wxTreeItemId& item) const

Returns TRUE if the item is expanded (only makes sense if it has children).

4.228.39 wxTreeCtrl::IsSelected

bool IsSelected(const wxTreeItemId& item) const

Returns TRUE if the item is selected.

4.228.40 wxTreeCtrl::IsVisible

bool IsVisible(const wxTreeItemId& item) const

Returns TRUE if the item is visible (it might be outside the view, or not expanded).

4.228.41 wxTreeCtrl::ItemHasChildren

bool ItemHasChildren(const wxTreeItemId& item) const

Returns TRUE if the item has children.

4.228.42 wxTreeCtrl::OnCompareItems

int OnCompareItems(const wxTreeItemId& item1, const wxTreeItemId& item2)

Override this function in the derived class to change the sort order of the items in the tree control. The function should return a negative, zero or positive value if the first item

is less than, equal to or greater than the second one.

The base class version compares items alphabetically.

See also: *SortChildren* (p. 772)

4.228.43 wxTreeCtrl::PrependItem

wxTreeItemId PrependItem(const wxTreeItemId& parent, const wxString& text, int image = -1, int selImage = -1, wxTreeItemData* data = NULL)

Appends an item as the first child of *parent*, return a new item id.

If *image* > -1 and *selImage* is -1, the same image is used for both selected and unselected items.

4.228.44 wxTreeCtrl::ScrollTo

void ScrollTo(const wxTreeItemId& item)

Scrolls the specified item into view.

4.228.45 wxTreeCtrl::SelectItem

bool SelectItem(const wxTreeItemId& item)

Selects the given item.

4.228.46 wxTreeCtrl::SetIndent

void SetIndent(int indent)

Sets the indentation for the tree control.

4.228.47 wxTreeCtrl::SetImageList

void SetImageList(wxImageList* imageList, int which = wxIMAGE_LIST_NORMAL)

Sets the image list. *which* should be one of `wxIMAGE_LIST_NORMAL`, `wxIMAGE_LIST_SMALL` and `wxIMAGE_LIST_STATE`.

4.228.48 wxTreeCtrl::SetItemBold

void SetItemBold(const wxTreeItemId& item, bool bold = TRUE)

Makes item appear in bold font if *bold* parameter is TRUE or resets it to the normal state.

See also: *IsBold* (p. 770)

4.228.49 wxTreeCtrl::SetItemData

void SetItemData(const wxTreeItemId& *item*, wxTreeItemData* *data*)

Sets the item client data.

4.228.50 wxTreeCtrl::SetItemHasChildren

void SetItemHasChildren(const wxTreeItemId& *item*, bool *hasChildren* = TRUE)

Force appearance of the button next to the item. This is useful to allow the user to expand the items which don't have any children now, but instead adding them only when needed, thus minimizing memory usage and loading time.

4.228.51 wxTreeCtrl::SetItemImage

void SetItemImage(const wxTreeItemId& *item*, int *image*)

Sets the normal item image. This is an index into the associated image list.

4.228.52 wxTreeCtrl::SetItemSelectedImage

void SetItemSelectedImage(const wxTreeItemId& *item*, int *selImage*)

Sets the item selected image. This is an index into the associated image list.

4.228.53 wxTreeCtrl::SetItemText

void SetItemText(const wxTreeItemId& *item*, const wxString& *text*)

Sets the item label.

4.228.54 wxTreeCtrl::SortChildren

void SortChildren(const wxTreeItemId& *item*)

Sorts the children of the given item using *OnCompareItems* (p. 770) method of wxTreeCtrl. You should override that method to change the sort order (default is ascending alphabetical order).

See also

wxTreeItemData (p. 773), *OnCompareItems* (p. 770)

4.228.55 wxTreeCtrl::Toggle

void Toggle(const wxTreeItemId& item)

Toggles the given item between collapsed and expanded states.

4.228.56 wxTreeCtrl::Unselect

void Unselect()

Removes the selection from the currently selected item (if any).

4.229 wxTreeItemData

wxTreeItemData is some (arbitrary) user class associated with some item. The main advantage of having this class (compared to the old untyped interface) is that *wxTreeItemData*'s are destroyed automatically by the tree and, as this class has virtual dtor, it means that the memory will be automatically freed. We don't just use *wxObject* instead of *wxTreeItemData* because the size of this class is critical: in any real application, each tree leaf will have *wxTreeItemData* associated with it and number of leaves may be quite big.

Because the objects of this class are deleted by the tree, they should always be allocated on the heap.

Derived from

wxTreeItemId

Include files

<wx/treectrl.h>

See also

wxTreeCtrl (p. 761)

4.229.1 wxTreeItemData::wxTreeItemData

wxTreeItemData()

Default constructor.

4.229.2 **wxTreeItemData::~~wxTreeItemData**

void ~wxTreeItemData()

Virtual destructor.

4.229.3 **wxTreeItemData::GetId**

const wxTreeItem& GetId()

Returns the item associated with this node.

4.229.4 **wxTreeItemData::SetId**

void SetId(const wxTreeItemId& id)

Sets the item associated with this node.

4.230 **wxTreeEvent**

A tree event holds information about events associated with wxTreeCtrl objects.

Derived from

wxCommandEvent (p. 103)

wxEvent (p. 221)

wxObject (p. 471)

Include files

<wx/treectrl.h>

Event table macros

To process input from a tree control, use these event handler macros to direct input to member functions that take a wxTreeEvent argument.

EVT_TREE_BEGIN_DRAG(id, func)	Begin dragging with the left mouse button.
EVT_TREE_BEGIN_RDRAG(id, func)	Begin dragging with the right mouse button.
EVT_TREE_BEGIN_LABEL_EDIT(id, func)	Begin editing a label.
EVT_TREE_END_LABEL_EDIT(id, func)	Finish editing a label.
EVT_TREE_DELETE_ITEM(id, func)	Delete an item.
EVT_TREE_GET_INFO(id, func)	Request information from the application.
EVT_TREE_SET_INFO(id, func)	Information is being supplied.

EVT_TREE_ITEM_EXPANDED(id, func) Parent has been expanded.
EVT_TREE_ITEM_EXPANDING(id, func) Parent is being expanded.
EVT_TREE_SEL_CHANGED(id, func) Selection has changed.
EVT_TREE_SEL_CHANGING(id, func) Selection is changing.
EVT_TREE_KEY_DOWN(id, func) A key has been pressed.

See also

wxTreeCtrl (p. 761)

4.230.1 wxTreeEvent::wxTreeEvent

wxTreeEvent(WXTYPE *commandType* = 0, int *id* = 0)

Constructor.

4.230.2 wxTreeEvent::m_code

int m_code

Key code if the event is a keypress event.

4.230.3 wxTreeEvent::m_itemIndex

wxTreeItem m_item

The item.

4.230.4 wxTreeEvent::m_oldItem

long m_oldItem

The old item index.

4.230.5 wxTreeEvent::m_pointDrag

wxPoint m_pointDrag

The position of the mouse pointer if the event is a drag event.

4.231 wxUpdateUIEvent

This class is used for pseudo-events which are called by wxWindows to give an application the chance to update various user interface elements.

Derived from

wxEvent (p. 221)
wxObject (p. 471)

Include files

<wx/event.h>

Event table macros

To process an update event, use these event handler macros to direct input to member functions that take a *wxUpdateUIEvent* argument.

EVT_UPDATE_UI(id, func) Process a *wxEVT_UPDATE_UI* event.

Remarks

Without update UI events, an application has to work hard to check/uncheck, enable/disable, and set the text for elements such as menu items and toolbar buttons. The code for doing this has to be mixed up with the code that is invoked when an action is invoked for a menu item or button.

With update UI events, you define an event handler to look at the state of the application and change UI elements accordingly. wxWindows will call your member functions in idle time, so you don't have to worry where to call this code. In addition to being a clearer and more declarative method, it also means you don't have to worry whether you're updating a toolbar or menubar identifier. The same handler can update a menu item and toolbar button, if the identifier is the same.

Instead of directly manipulating the menu or button, you call functions in the event object, such as *wxUpdateUIEvent::Check* (p. 777). wxWindows will determine whether such a call has been made, and which UI element to update.

These events will work for popup menus as well as menubars. Just before a menu is popped up, *wxMenu::UpdateUI* (p. 426) is called to process any UI events for the window that owns the menu.

See also

Event handling overview (p. 939)

4.231.1 **wxUpdateUIEvent::wxUpdateUIEvent**

wxUpdateUIEvent(wxWindowID *commandId* = 0)

Constructor.

4.231.2 wxUpdateUIEvent::m_checked

bool m_checked

TRUE if the element should be checked, FALSE otherwise.

4.231.3 wxUpdateUIEvent::m_enabled

bool m_checked

TRUE if the element should be enabled, FALSE otherwise.

4.231.4 wxUpdateUIEvent::m_setChecked

bool m_setChecked

TRUE if the application has set the **m_checked** member.

4.231.5 wxUpdateUIEvent::m_setEnabled

bool m_setEnabled

TRUE if the application has set the **m_enabled** member.

4.231.6 wxUpdateUIEvent::m_setText

bool m_setText

TRUE if the application has set the **m_text** member.

4.231.7 wxUpdateUIEvent::m_text

wxString m_text

Holds the text with which the the application wishes to update the UI element.

4.231.8 wxUpdateUIEvent::Check

void Check(bool *check*)

Check or uncheck the UI element.

4.231.9 wxUpdateUIEvent::Enable

void Enable(bool *enable*)

Enable or disable the UI element.

4.231.10 wxUpdateUIEvent::GetChecked

bool GetChecked() const

Returns TRUE if the UI element should be checked.

4.231.11 wxUpdateUIEvent::GetEnabled

bool GetEnabled() const

Returns TRUE if the UI element should be enabled.

4.231.12 wxUpdateUIEvent::GetSetChecked

bool GetSetChecked() const

Returns TRUE if the application has called **SetChecked**. For wxWindows internal use only.

4.231.13 wxUpdateUIEvent::GetSetEnabled

bool GetSetEnabled() const

Returns TRUE if the application has called **SetEnabled**. For wxWindows internal use only.

4.231.14 wxUpdateUIEvent::GetSetText

bool GetSetText() const

Returns TRUE if the application has called **SetText**. For wxWindows internal use only.

4.231.15 wxUpdateUIEvent::GetText

wxString GetText() const

Returns the text that should be set for the UI element.

4.231.16 wxUpdateUIEvent::SetText**void SetText(const wxString& text)**

Sets the text for this UI element.

4.232 wxURL**Derived from**

wxObject (p. 471)

Include files

<wx/url.h>

See also

wxSocketBase (p. 607), *wxProtocol* (p. 528)

Example

```
wxURL url("http://a.host/a.dir/a.file");
wxInputStream *in_stream;

in_stream = url.GetInputStream();
// Then, you can use all IO calls of in_stream (See wxStream)
```

4.232.1 wxURL::wxURL**wxURL(const wxString& url)**

Constructs an URL object from the string.

Parameters

url
Url string to parse.

4.232.2 wxURL::~~wxURL**~wxURL()**

Destroys the URL object.

4.232.3 wxURL::GetProtocolName

wxString GetProtocolName() const

Returns the name of the protocol which will be used to get the URL.

4.232.4 wxURL::GetProtocol

wxProtocol& GetProtocol()

Returns a reference to the protocol which will be used to get the URL.

4.232.5 wxURL::GetPath

wxString GetPath()

Returns the path of the file to fetch. This path was encoded in the URL.

4.232.6 wxURL::GetError

wxURLError GetError() const

Returns the last error. This error refers to the URL parsing or to the protocol. It can be one of these errors:

wxURL_NOERR	No error.
wxURL_SNTAXERR	Syntax error in the URL string.
wxURL_NOPROTO	Found no protocol which can get this URL.
wxURL_NOHOST	An host name is required for this protocol.
wxURL_NOPATH	A path is required for this protocol.
wxURL_CONNERR	Connection error.
wxURL_PROTOERR	An error occurred during negotiation.

4.232.7 wxURL::GetInputStream

wxInputStream * GetInputStream()

Creates a new input stream on the the specified URL. You can use all but seek functionality of wxStream. Seek isn't available on all stream. For example, http or ftp streams doesn't deal with it.

Return value

Returns the initialized stream. You will have to delete it yourself.

See also

`wxInputStream`

4.232.8 `wxURL::SetDefaultProxy`

static void SetDefaultProxy(const wxString& url_proxy)

Sets the default proxy server to use to get the URL. The string specifies the proxy like this: <hostname>:<port number>.

Parameters

url_proxy
Specifies the proxy to use

See also

`wxURL::SetProxy` (p. 781)

4.232.9 `wxURL::SetProxy`

void SetProxy(const wxString& url_proxy)

Sets the proxy to use for this URL.

See also

`wxURL::SetDefaultProxy` (p. 781)

4.233 `wxValidator`

`wxValidator` is the base class for a family of validator classes that mediate between a class of control, and application data.

A validator has three major roles:

1. to transfer data from a C++ variable or own storage to and from a control;
2. to validate data in a control, and show an appropriate error message;
3. to filter events (such as keystrokes), thereby changing the behaviour of the associated control.

Validators can be plugged into controls dynamically.

To specify a default, 'null' validator, use the symbol **wxDefaultValidator**.

For more information, please see *Validator overview* (p. 969).

Derived from

wxEvtHandler (p. 224)

wxObject (p. 471)

Include files

<wx/validate.h>

See also

Validator overview (p. 969), *wxTextValidator* (p. 728)

4.233.1 wxValidator::wxValidator

wxValidator()

Constructor.

4.233.2 wxValidator::~~wxValidator

~wxValidator()

Destructor.

4.233.3 wxValidator::Clone

virtual wxValidator* Clone() const

All validator classes must implement the **Clone** function, which returns an identical copy of itself. This is because validators are passed to control constructors as references which must be copied. Unlike objects such as pens and brushes, it does not make sense to have a reference counting scheme to do this cloning, because all validators should have separate data.

This base function returns NULL.

4.233.4 wxValidator::GetWindow

wxWindow* GetWindow() const

Returns the window associated with the validator.

4.233.5 wxValidator::SetBellOnError

wxvalidatorsetbellonerror

void SetBellOnError(bool *dolt* = *TRUE*)

This functions switches on or turns off the error sound produced by the validators if an invalid key is pressed.

4.233.6 wxValidator::SetWindow

void SetWindow(wxWindow* *window*)

Associates a window with the validator.

4.233.7 wxValidator::TransferFromWindow

virtual bool TransferToWindow(wxWindow* *parent*)

This overridable function is called when the value in the window must be transferred to the validator. Return FALSE if there is a problem.

4.233.8 wxValidator::TransferToWindow

virtual bool TransferToWindow(wxWindow* *parent*)

This overridable function is called when the value associated with the validator must be transferred to the window. Return FALSE if there is a problem.

4.233.9 wxValidator::Validate

virtual bool Validate(wxWindow* *parent*)

This overridable function is called when the value in the associated window must be validated. Return FALSE if the value in the window is not valid; you may pop up an error dialog.

4.234 wxVariant

The **wxVariant** class represents a container for any type. A variant's value can be changed at run time, possibly to a different type of value.

As standard, `wxVariant` can store values of type `bool`, `char`, `double`, `long`, `string`, `string list`, `time`, `date`, `void pointer`, `list of strings`, and `list of variants`. However, an application can extend `wxVariant`'s capabilities by deriving from the class `wxVariantData` (p. 792) and using the `wxVariantData` form of the `wxVariant` constructor or assignment operator to assign this data to a variant. Actual values for user-defined types will need to be accessed via the `wxVariantData` object, unlike the case for basic data types where convenience functions such as `GetLong` can be used.

This class is useful for reducing the programming for certain tasks, such as an editor for different data types, or a remote procedure call protocol.

An optional name member is associated with a `wxVariant`. This might be used, for example, in CORBA or OLE automation classes, where named parameters are required.

`wxVariant` is similar to `wxExpr` and also to `wxPropertyValue`. However, `wxExpr` is efficiency-optimized for a restricted range of data types, whereas `wxVariant` is less efficient but more extensible. `wxPropertyValue` may be replaced by `wxVariant` eventually.

Derived from

`wxObject` (p. 471)

Include files

`<wx/variant.h>`

See also

`wxVariantData` (p. 792)

4.234.1 `wxVariant::wxVariant`

`wxVariant()`

Default constructor.

`wxVariant(const wxVariant& variant)`

Copy constructor.

**`wxVariant(const char* value, const wxString& name = "")`
`wxVariant(const wxString& value, const wxString& name = "")`**

Construction from a string value.

`wxVariant(char value, const wxString& name = "")`

Construction from a character value.

wxVariant(long value, const wxString& name = "")

Construction from an integer value. You may need to cast to (long) to avoid confusion with other constructors (such as the bool constructor).

wxVariant(bool value, const wxString& name = "")

Construction from a boolean value.

wxVariant(double value, const wxString& name = "")

Construction from a double-precision floating point value.

wxVariant(const wxList& value, const wxString& name = "")

Construction from a list of wxVariant objects. This constructor copies *value*, the application is still responsible for deleting *value* and its contents.

wxVariant(const wxStringList& value, const wxString& name = "")

Construction from a list of strings. This constructor copies *value*, the application is still responsible for deleting *value* and its contents.

wxVariant(const wxTime& value, const wxString& name = "")

Construction from a time.

wxVariant(const wxDate& value, const wxString& name = "")

Construction from a date.

wxVariant(void* value, const wxString& name = "")

Construction from a void pointer.

wxVariant(wxVariantData* data, const wxString& name = "")

Construction from user-defined data. The variant holds on to the *data* pointer.

4.234.2 wxVariant::~~wxVariant

~wxVariant()

Destructor.

4.234.3 wxVariant::Append

void Append(const wxVariant& value)

Appends a value to the list.

4.234.4 wxVariant::ClearList

void ClearList()

Deletes the contents of the list.

4.234.5 wxVariant::GetCount

int GetCount() const

Returns the number of elements in the list.

4.234.6 wxVariant::Delete

bool Delete(int item)

Deletes the zero-based *item* from the list.

4.234.7 wxVariant::GetBool

bool GetBool() const

Returns the boolean value.

4.234.8 wxVariant::GetChar

char GetChar() const

Returns the character value.

4.234.9 wxVariant::GetData

wxVariantData* GetData() const

Returns a pointer to the internal variant data.

4.234.10 wxVariant::GetDate

wxDate GetDate() const

Gets the date value.

4.234.11 wxVariant::GetDouble

double GetDouble() const

Returns the floating point value.

4.234.12 wxVariant::GetLong

long GetLong() const

Returns the integer value.

4.234.13 wxVariant::GetName

const wxString& GetName() const

Returns a constant reference to the variant name.

4.234.14 wxVariant::GetString

wxString GetString() const

Gets the string value.

4.234.15 wxVariant::GetTime

wxTime GetTime() const

Gets the time value.

4.234.16 wxVariant::GetType

wxString GetType() const

Returns the value type as a string. The built-in types are: bool, char, date, double, list, long, string, stringlist, time, void*.

If the variant is null, the value type returned is the string "null" (not the empty string).

4.234.17 wxVariant::GetVoidPtr

void* GetVoidPtr() const

Gets the void pointer value.

4.234.18 wxVariant::Insert

void Insert(const wxVariant& value)

Inserts a value at the front of the list.

4.234.19 wxVariant::IsNull

bool IsNull() const

Returns TRUE if there is no data associated with this variant, FALSE if there is data.

4.234.20 wxVariant::IsType

bool IsType(const wxString& type) const

Returns TRUE if *type* matches the type of the variant, FALSE otherwise.

4.234.21 wxVariant::MakeNull

void MakeNull()

Makes the variant null by deleting the internal data.

4.234.22 wxVariant::MakeString

wxString MakeString() const

Makes a string representation of the variant value (for any type).

4.234.23 wxVariant::Member

bool Member(const wxVariant& value) const

Returns TRUE if *value* matches an element in the list.

4.234.24 wxVariant::NullList

void NullList()

Makes an empty list. This differs from a null variant which has no data; a null list is of type list, but the number of elements in the list is zero.

4.234.25 wxVariant::SetData**void SetData(wxVariantData* data)**

Sets the internal variant data, deleting the existing data if there is any.

4.234.26 wxVariant::operator =

void operator =(const wxVariant& value)

void operator =(wxVariantData* value)

void operator =(const wxString& value)

void operator =(const char* value)

void operator =(char value)

void operator =(const long value)

void operator =(const bool value)

void operator =(const double value)

void operator =(const wxDate& value)

void operator =(const wxTime& value)

void operator =(void* value)

void operator =(const wxList& value)

void operator =(const wxStringList& value)

Assignment operators.

4.234.27 wxVariant::operator ==

bool operator ==(const wxVariant& value)

bool operator ==(const wxString& value)

bool operator ==(const char* value)

bool operator ==(char value)

bool operator ==(const long value)

bool operator ==(const bool value)

bool operator ==(const double value)

bool operator ==(const wxDate& value)

bool operator ==(const wxTime& value)

bool operator ==(void* value)

bool operator ==(const wxList& value)

bool operator ==(const wxStringList& value)

Equality test operators.

4.234.28 wxVariant::operator !=

bool operator !=(const wxVariant& value)

bool operator !=(const wxString& value)

bool operator !=(const char* value)

bool operator !=(char value)

bool operator !=(const long value)

bool operator !=(const bool value)

bool operator !=(const double value)

bool operator !=(const wxDate& value)

bool operator !=(const wxTime& value)

bool operator !=(void* value)

bool operator !=(const wxList& value)

bool operator !=(const wxStringList& value)

Inequality test operators.

4.234.29 wxVariant::operator []**wxVariant operator [](size_t idx) const**

Returns the value at *idx* (zero-based).

wxVariant& operator [](size_t idx)

Returns a reference to the value at *idx* (zero-based). This can be used to change the value at this index.

4.234.30 wxVariant::operator char**char operator char() const**

Operator for implicit conversion to a char, using *wxVariant::GetChar* (p. 786).

4.234.31 wxVariant::operator double**double operator double() const**

Operator for implicit conversion to a double, using *wxVariant::GetDouble* (p. 787).

long operator long() const

Operator for implicit conversion to a long, using *wxVariant::GetLong* (p. 787).

4.234.32 wxVariant::operator wxDate**wxDate operator wxDate() const**

Operator for implicit conversion to a wxDate, using *wxVariant::GetDate* (p. 786).

4.234.33 wxVariant::operator wxString**wxString operator wxString() const**

Operator for implicit conversion to a string, using *wxVariant::MakeString* (p. 788).

4.234.34 wxVariant::operator wxTime**wxTime operator wxTime() const**

Operator for implicit conversion to a wxTime, using *wxVariant::GetTime* (p. 787).

4.234.35 wxVariant::operator void***void* operator void*() const**

Operator for implicit conversion to a pointer to a void, using *wxVariant::GetVoidPtr* (p. 787).

4.235 wxVariantData

The **wxVariantData** is used to implement a new type for *wxVariant*. Derive from *wxVariantData*, and override the pure virtual functions.

Derived from

wxObject (p. 471)

Include files

<wx/variant.h>

See also

wxVariant (p. 783)

4.235.1 wxVariantData::wxVariantData**wxVariantData()**

Default constructor.

4.235.2 wxVariantData::Copy**void Copy(wxVariantData& data)**

Copy the data from 'this' object to *data*.

4.235.3 wxVariantData::Eq**bool Eq(wxVariantData& data) const**

Returns TRUE if this object is equal to *data*.

4.235.4 wxVariantData::GetType**wxString GetType() const**

Returns the string type of the data.

4.235.5 wxVariantData::Read**bool Read(ostream& *stream*)****bool Read(wxString& *string*)**

Reads the data from *stream* or *string*.

4.235.6 wxVariantData::Write**bool Write(ostream& *stream*) const****bool Write(wxString& *string*) const**

Writes the data to *stream* or *string*.

4.236 wxView

The view class can be used to model the viewing and editing component of an application's file-based data. It is part of the document/view framework supported by `wxWindows`, and cooperates with the `wxDocument` (p. 207), `wxDocTemplate` (p. 202) and `wxDocManager` (p. 189) classes.

Derived from

`wxEvtHandler` (p. 224)

`wxObject` (p. 471)

Include files

<wx/docview.h>

See also

`wxView` overview (p. 935), `wxDocument` (p. 207), `wxDocTemplate` (p. 202), `wxDocManager` (p. 189)

4.236.1 wxView::m_viewDocument

wxDocument* m_viewDocument

The document associated with this view. There may be more than one view per document, but there can never be more than one document for one view.

4.236.2 wxView::m_viewFrame**wxFrame* m_viewFrame**

Frame associated with the view, if any.

4.236.3 wxView::m_viewTypeName**wxString m_viewTypeName**

The view type name given to the wxDocTemplate constructor, copied to this variable when the view is created. Not currently used by the framework.

4.236.4 wxView::wxView**wxView()**

Constructor. Define your own default constructor to initialize application-specific data.

4.236.5 wxView::~~wxView**~wxView()**

Destructor. Removes itself from the document's list of views.

4.236.6 wxView::Activate**virtual void Activate(bool activate)**

Call this from your view frame's OnActivate member to tell the framework which view is currently active. If your windowing system doesn't call OnActivate, you may need to call this function from OnMenuCommand or any place where you know the view must be active, and the framework will need to get the current view.

The prepackaged view frame wxDocChildFrame calls wxView::Activate from its OnActivate member and from its OnMenuCommand member.

This function calls wxView::OnActivateView.

4.236.7 wxView::Close**virtual bool Close**(bool *deleteWindow* = TRUE)

Closes the view by calling OnClose. If *deleteWindow* is TRUE, this function should delete the window associated with the view.

4.236.8 wxView::GetDocument**wxDocument* GetDocument()** const

Gets a pointer to the document associated with the view.

4.236.9 wxView::GetDocumentManager**wxDocumentManager* GetDocumentManager()** const

Returns a pointer to the document manager instance associated with this view.

4.236.10 wxView::GetFrame**wxFrame * GetFrame()**

Gets the frame associated with the view (if any).

4.236.11 wxView::GetViewName**wxString GetViewName()** const

Gets the name associated with the view (passed to the wxDocTemplate constructor). Not currently used by the framework.

4.236.12 wxView::OnActivateView**virtual void OnActivateView**(bool *activate*, wxView **activeView*, wxView **deactiveView*)

Called when a view is activated by means of wxView::Activate. The default implementation does nothing.

4.236.13 wxView::OnChangeFilename**virtual void OnChangeFilename()**

Called when the filename has changed. The default implementation constructs a suitable title and sets the title of the view frame (if any).

4.236.14 **wxView::OnClose**

virtual bool OnClose(bool *deleteWindow*)

Implements closing behaviour. The default implementation calls `wxDocument::Close` to close the associated document. Does not delete the view. The application may wish to do some cleaning up operations in this function, *if* a call to `wxDocument::Close` succeeded. For example, if your application's all share the same window, you need to disassociate the window from the view and perhaps clear the window. If *deleteWindow* is TRUE, delete the frame associated with the view.

4.236.15 **wxView::OnCreate**

virtual bool OnCreate(wxDocument* *doc*, long *flags*)

Called just after view construction to give the view a chance to initialize itself based on the passed document and flags (unused). By default, simply returns TRUE. If the function returns FALSE, the view will be deleted.

The predefined document child frame, `wxDocChildFrame`, calls this function automatically.

4.236.16 **wxView::OnCreatePrintout**

virtual wxPrintout* OnCreatePrintout()

If the printing framework is enabled in the library, this function returns a *wxPrintout* (p. 516) object for the purposes of printing. It should create a new object everytime it is called; the framework will delete objects it creates.

By default, this function returns an instance of `wxDocPrintout`, which prints and previews one page by calling `wxView::OnDraw`.

Override to return an instance of a class other than `wxDocPrintout`.

4.236.17 **wxView::OnUpdate**

virtual void OnUpdate(wxView* *sender*, wxObject* *hint*)

Called when the view should be updated. *sender* is a pointer to the view that sent the update request, or NULL if no single view requested the update (for instance, when the document is opened). *hint* is as yet unused but may in future contain application-specific information for making updating more efficient.

4.236.18 wxView::SetDocument**void SetDocument(wxDocument* doc)**

Associates the given document with the view. Normally called by the framework.

4.236.19 wxView::SetFrame**void SetFrame(wxFrame* frame)**

Sets the frame associated with this view. The application should call this if possible, to tell the view about the frame.

4.236.20 wxView::SetViewName**void SetViewName(const wxString& name)**

Sets the view type name. Should only be called by the framework.

4.237 wxWave

This class represents a short wave file, in Windows WAV format, that can be stored in memory and played. Currently this class is for Windows only.

Derived from

wxObject (p. 471)

Include files

<wx/wave.h>

4.237.1 wxWave::wxWave**wxWave()**

Default constructor.

wxWave(const wxString& fileName, bool isResource = FALSE)

Constructs a wave object from a file or resource. Call *wxWave::IsOk* (p. 798) to determine whether this succeeded.

Parameters

fileName

The filename or Windows resource.

isResource

TRUE if *fileName* is a resource, FALSE if it is a filename.

4.237.2 **wxWave::~~wxWave**

~wxWave()

Destroys the wxWave object.

4.237.3 **wxWave::Create**

bool Create(const wxString& fileName, bool isResource = FALSE)

Constructs a wave object from a file or resource.

Parameters

fileName

The filename or Windows resource.

isResource

TRUE if *fileName* is a resource, FALSE if it is a filename.

Return value

TRUE if the call was successful, FALSE otherwise.

4.237.4 **wxWave::IsOk**

bool IsOk() const

Returns TRUE if the object contains a successfully loaded file or resource, FALSE otherwise.

4.237.5 **wxWave::Play**

bool Play(bool async = TRUE, bool looped = FALSE) const

Plays the wave file synchronously or asynchronously, looped or single-shot.

4.238 **wxWindow**

`wxWindow` is the base class for all windows. Any children of the window will be deleted automatically by the destructor before the window itself is deleted.

Derived from

`wxEvtHandler` (p. 224)

`wxObject` (p. 471)

Include files

`<wx/window.h>`

Window styles

The following styles can apply to all windows, although they will not always make sense for a particular window class.

wxSIMPLE_BORDER	Displays a thin border around the window. <code>wxBORDER</code> is the old name for this style.
wxDOUBLE_BORDER	Displays a double border. Windows only.
wxSUNKEN_BORDER	Displays a sunken border.
wxRAISED_BORDER	Displays a raised border.
wxSTATIC_BORDER	Displays a border suitable for a static control.
wxTRANSPARENT_WINDOW	The window is transparent, that is, it will not receive paint events. Windows only.
wxNO_3D	Prevents the children of this window taking on 3D styles, even though the application-wide policy is for 3D controls. Windows only.
wxTAB_TRAVERSAL	Use this to enable tab traversal for non-dialog windows.
wxVSCROLL	Use this style to enable a vertical scrollbar.
wxHSCROLL	Use this style to enable a horizontal scrollbar.
wxCLIP_CHILDREN	Use this style to eliminate flicker caused by the background being repainted, then children being painted over them. Windows-only.

See also *window styles overview* (p. 959).

See also

Event handling overview (p. 939)

4.238.1 `wxWindow::wxWindow`

`wxWindow()`

Default constructor.

wxWindow(**wxWindow*** *parent*, **wxWindowID** *id*, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = 0, **const wxString&** *name* = *wxPanelNameStr*)

Constructs a window, which can be a child of a frame, dialog or any other non-control window.

Parameters

parent

Pointer to a parent window.

id

Window identifier. If -1, will automatically create an identifier.

pos

Window position. *wxDefaultPosition* is (-1, -1) which indicates that *wxWindows* should generate a default position for the window. If using the *wxWindow* class directly, supply an actual position.

size

Window size. *wxDefaultSize* is (-1, -1) which indicates that *wxWindows* should generate a default size for the window.

style

Window style. For generic window styles, please see *wxWindow* (p. 798).

name

Window name.

4.238.2 **wxWindow::~~wxWindow**

~wxWindow()

Destructor. Deletes all subwindows, then deletes itself. Instead of using the **delete** operator explicitly, you should normally use *wxWindow::Destroy* (p. 805) so that *wxWindows* can delete a window only when it is safe to do so, in idle time.

See also

Window deletion overview (p. 929), *wxWindow::OnCloseWindow* (p. 819), *wxWindow::Destroy* (p. 805), *wxCloseEvent* (p. 84)

4.238.3 **wxWindow::AddChild**

virtual void AddChild(**wxWindow*** *child*)

Adds a child window. This is called automatically by window creation functions so

should not be required by the application programmer.

Parameters

child

Child window to add.

4.238.4 **wxWindow::CaptureMouse**

virtual void CaptureMouse()

Directs all mouse input to this window. Call *wxWindow::ReleaseMouse* (p. 830) to release the capture.

See also

wxWindow::ReleaseMouse (p. 830)

4.238.5 **wxWindow::Center**

void Center(int direction)

A synonym for *Centre* (p. 801).

4.238.6 **wxWindow::Centre**

virtual void Centre(int direction = wxHORIZONTAL)

Centres the window.

Parameters

direction

Specifies the direction for the centering. May be `wxHORIZONTAL`, `wxVERTICAL` or `wxBOTH`.

Remarks

The actual behaviour depends on the derived window. For a frame or dialog box, centring is relative to the whole display. For a panel item, centring is relative to the panel.

See also

wxWindow::Center (p. 801)

4.238.7 wxWindow::Clear**void Clear()**

Clears the window by filling it with the current background colour. Does not cause an erase background event to be generated.

4.238.8 wxWindow::ClientToScreen**virtual void ClientToScreen(int* x, int* y) const****virtual wxPoint ClientToScreen(const wxPoint& pt) const**

Converts to screen coordinates from coordinates relative to this window.

x

A pointer to a integer value for the x coordinate. Pass the client coordinate in, and a screen coordinate will be passed out.

y

A pointer to a integer value for the y coordinate. Pass the client coordinate in, and a screen coordinate will be passed out.

pt

The client position for the second form of the function.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

ClientToScreen(point)	Accepts and returns a wxPoint
ClientToScreenXY(x, y)	Returns a 2-tuple, (x, y)

4.238.9 wxWindow::Close**virtual bool Close(const bool force = FALSE)**

The purpose of this call is to provide a safer way of destroying a window than using the *delete* operator.

Parameters*force*

FALSE if the window's close handler should be able to veto the destruction of this window, TRUE if it cannot.

Remarks

Close calls the *close handler* (p. 84) for the window, providing an opportunity for the window to choose whether to destroy the window.

The close handler should check whether the window is being deleted forcibly, using *wxCloseEvent::GetForce* (p. 85), in which case it should destroy the window using *wxWindow::Destroy* (p. 805).

Applies to managed windows (*wxFrame* and *wxDIALOG* classes) only.

Note that calling Close does not guarantee that the window will be destroyed; but it provides a way to simulate a manual close of a window, which may or may not be implemented by destroying the window. The default implementation of *wxDIALOG::OnCloseWindow* does not necessarily delete the dialog, since it will simply simulate an *wxID_CANCEL* event which itself only hides the dialog.

To guarantee that the window will be destroyed, call *wxWindow::Destroy* (p. 805) instead.

See also

Window deletion overview (p. 929), *wxWindow::OnCloseWindow* (p. 819), *wxWindow::Destroy* (p. 805), *wxCloseEvent* (p. 84)

4.238.10 **wxWindow::ConvertDialogToPixels**

wxPoint ConvertDialogToPixels(const wxPoint& pt)

wxSize ConvertDialogToPixels(const wxSize& sz)

Converts a point or size from dialog units to pixels.

For the x dimension, the dialog units are multiplied by the average character width and then divided by 4.

For the y dimension, the dialog units are multiplied by the average character height and then divided by 8.

Remarks

Dialog units are used for maintaining a dialog's proportions even if the font changes. Dialogs created using Dialog Editor optionally use dialog units.

You can also use these functions programmatically. A convenience macro is defined:

```
#define wxDLG_UNIT(parent, pt) parent->ConvertDialogToPixels(pt)
```

See also

wxWindow::ConvertPixelsToDialog (p. 804)

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

ConvertDialogPointToPixels(point)	Accepts and returns a wxPoint
ConvertDialogSizeToPixels(size)	Accepts and returns a wxSize

Additionally, the following helper functions are defined:

wxDLG_PNT(win, point)	Converts a wxPoint from dialog units to pixels
wxDLG_SZE(win, size)	Converts a wxSize from dialog units to pixels

4.238.11 wxWindow::ConvertPixelsToDialog

wxPoint **ConvertPixelsToDialog(const wxPoint& pt)**

wxSize **ConvertPixelsToDialog(const wxSize& sz)**

Converts a point or size from pixels to dialog units.

For the x dimension, the pixels are multiplied by 4 and then divided by the average character width.

For the y dimension, the pixels are multiplied by 8 and then divided by the average character height.

Remarks

Dialog units are used for maintaining a dialog's proportions even if the font changes. Dialogs created using Dialog Editor optionally use dialog units.

See also

wxWindow::ConvertDialogToPixels (p. 803)

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

ConvertDialogPointToPixels(point)	Accepts and returns a wxPoint
ConvertDialogSizeToPixels(size)	Accepts and returns a wxSize

4.238.12 wxWindow::Destroy**virtual bool Destroy()**

Destroys the window safely. Use this function instead of the delete operator, since different window classes can be destroyed differently. Frames and dialogs are not destroyed immediately when this function is called - they are added to a list of windows to be deleted on idle time, when all the window's events have been processed. This prevents problems with events being sent to non-existent windows.

Return value

TRUE if the window has either been successfully deleted, or it has been added to the list of windows pending real deletion.

4.238.13 wxWindow::DestroyChildren**virtual void DestroyChildren()**

Destroys all children of a window. Called automatically by the destructor.

4.238.14 wxWindow::DragAcceptFiles**virtual void DragAcceptFiles(const bool *accept*)**

Enables or disables eligibility for drop file events (OnDropFiles).

Parameters*accept*

If TRUE, the window is eligible for drop file events. If FALSE, the window will not accept drop file events.

Remarks

Windows only.

See also

wxWindow::OnDropFiles (p. 820)

4.238.15 wxWindow::Enable**virtual void Enable(const bool *enable*)**

Enable or disable the window for user input.

Parameters

enable

If TRUE, enables the window for input. If FALSE, disables the window.

See also

wxWindow::IsEnabled (p. 814)

4.238.16 **wxWindow::FindFocus**

static wxWindow* FindFocus()

Finds the window or control which currently has the keyboard focus.

Remarks

Note that this is a static function, so it can be called without needing a wxWindow pointer.

See also

wxWindow::SetFocus (p. 835)

4.238.17 **wxWindow::FindWindow**

wxWindow* FindWindow(long id)

Find a child of this window, by identifier.

wxWindow* FindWindow(const wxString& name)

Find a child of this window, by name.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

FindWindowById(id)	Accepts an integer
FindWindowByName(name)	Accepts a string

4.238.18 **wxWindow::Fit**

virtual void Fit()

Sizes the window so that it fits around its subwindows.

4.238.19 wxWindow::GetBackgroundColour**virtual wxColour GetBackgroundColour() const**

Returns the background colour of the window.

See also

wxWindow::SetBackgroundColour (p. 832), *wxWindow::SetForegroundColour* (p. 836),
wxWindow::GetForegroundColour (p. 809), *wxWindow::OnEraseBackground* (p. 821)

4.238.20 wxWindow::GetCharHeight**virtual int GetCharHeight() const**

Returns the character height for this window.

4.238.21 wxWindow::GetCharWidth**virtual int GetCharWidth() const**

Returns the average character width for this window.

4.238.22 wxWindow::GetChildren**wxList& GetChildren()**

Returns a reference to the list of the window's children.

4.238.23 wxWindow::GetClientSize**virtual void GetClientSize(int* width, int* height) const****virtual wxSize GetClientSize() const**

This gets the size of the window 'client area' in pixels. The client area is the area which may be drawn on by the programmer, excluding title bar, border etc.

Parameters*width*

Receives the client width in pixels.

height

Receives the client height in pixels.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

wxGetClientSizeTuple()	Returns a 2-tuple of (width, height)
wxGetClientSize()	Returns a wxSize object

4.238.24 **wxWindow::GetConstraints**

wxLayoutConstraints* GetConstraints() const

Returns a pointer to the window's layout constraints, or NULL if there are none.

4.238.25 **wxWindow::GetDefaultItem**

wxButton* GetDefaultItem() const

Returns a pointer to the button which is the default for this window, or NULL.

4.238.26 **wxWindow::GetDropTarget**

wxDropTarget* GetDropTarget() const

Returns the associated drop target, which may be NULL.

See also

wxWindow::SetDropTarget (p. 835), *Drag and drop overview* (p. 975)

4.238.27 **wxWindow::GetEventHandler**

wxEvtHandler* GetEventHandler() const

Returns the event handler for this window. By default, the window is its own event handler.

See also

wxWindow::SetEventHandler (p. 834), *wxWindow::PushEventHandler* (p. 829), *wxWindow::PopEventHandler* (p. 829), *wxEvtHandler::ProcessEvent* (p. 227), *wxEvtHandler* (p. 224)

4.238.28 wxWindow::GetFont**wxFont& GetFont() const**

Returns a reference to the font for this window.

See also

wxWindow::SetFont (p. 835)

4.238.29 wxWindow::GetForegroundColour**virtual wxColour GetForegroundColour()**

Returns the foreground colour of the window.

Remarks

The interpretation of foreground colour is open to interpretation according to the window class; it may be the text colour or other colour, or it may not be used at all.

See also

wxWindow::SetForegroundColour (p. 836), *wxWindow::SetBackgroundColour* (p. 832), *wxWindow::GetBackgroundColour* (p. 807)

4.238.30 wxWindow::GetGrandParent**wxWindow* GetGrandParent() const**

Returns the grandparent of a window, or NULL if there isn't one.

4.238.31 wxWindow::GetHandle**void* GetHandle() const**

Returns the platform-specific handle of the physical window. Cast it to an appropriate handle, such as **HWND** for Windows or **Widget** for Motif.

4.238.32 wxWindow::GetId**int GetId() const**

Returns the identifier of the window.

Remarks

Each window has an integer identifier. If the application has not provided one, an identifier will be generated.

See also

wxWindow::SetId (p. 836) *Window identifiers* (p. 943)

4.238.33 wxWindow::GetPosition

virtual void GetPosition(int* x, int* y) const

This gets the position of the window in pixels, relative to the parent window or if no parent, relative to the whole display.

Parameters

x
Receives the x position of the window.

y
Receives the y position of the window.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

GetPosition()	Returns a wxPoint
GetPositionTuple()	Returns a tuple (x, y)

4.238.34 wxWindow::GetLabel

virtual wxString& GetLabel() const

Generic way of getting a label from any window, for identification purposes.

Remarks

The interpretation of this function differs from class to class. For frames and dialogs, the value returned is the title. For buttons or static text controls, it is the button text. This function can be useful for meta-programs (such as testing tools or special-needs access programs) which need to identify windows by name.

4.238.35 wxWindow::GetName

virtual wxString& GetName() const

Returns the window's name.

Remarks

This name is not guaranteed to be unique; it is up to the programmer to supply an appropriate name in the window constructor or via *wxWindow::SetName* (p. 836).

See also

wxWindow::SetName (p. 836)

4.238.36 **wxWindow::GetParent**

virtual wxWindow* GetParent() const

Returns the parent of the window, or NULL if there is no parent.

4.238.37 **wxWindow::GetRect**

virtual wxRect GetRect() const

Returns the size and position of the window as a *wxRect* (p. 546) object.

4.238.38 **wxWindow::GetReturnCode**

int GetReturnCode()

Gets the return code for this window.

Remarks

A return code is normally associated with a modal dialog, where *wxDialog::ShowModal* (p. 185) returns a code to the application.

See also

wxWindow::SetReturnCode (p. 837), *wxDialog::ShowModal* (p. 185),
wxDialog::EndModal (p. 181)

4.238.39 **wxWindow::GetScrollThumb**

virtual int GetScrollThumb(int orientation)

Returns the built-in scrollbar thumb size.

See also

wxWindow::SetScrollbar (p. 837)

4.238.40 wxWindow::GetScrollPos

virtual int GetScrollPos(int orientation)

Returns the built-in scrollbar position.

See also

See *wxWindow::SetScrollbar* (p. 837)

4.238.41 wxWindow::GetScrollRange

virtual int GetScrollRange(int orientation)

Returns the built-in scrollbar range.

See also

wxWindow::SetScrollbar (p. 837)

4.238.42 wxWindow::GetSize

virtual void GetSize(int* width, int* height) const

virtual wxSize GetSize() const

This gets the size of the entire window in pixels.

Parameters

width

Receives the window width.

height

Receives the window height.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

GetSize()	Returns a wxSize
GetSizeTuple()	Returns a 2-tuple (width, height)

4.238.43 wxWindow::GetTextExtent

virtual void GetTextExtent(const wxString& string, int* x, int* y, int* descent = NULL, int* externalLeading = NULL, const wxFont* font = NULL, const bool use16 = FALSE) const

Gets the dimensions of the string as it would be drawn on the window with the currently selected font.

Parameters

string

String whose extent is to be measured.

x

Return value for width.

y

Return value for height.

descent

Return value for descent (optional).

externalLeading

Return value for external leading (optional).

font

Font to use instead of the current window font (optional).

use16

If TRUE, *string* contains 16-bit characters. The default is FALSE.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

GetTextExtent(string)	Returns a 2-tuple, (width, height)
GetFullTextExtent(string, font=NULL)	Returns a 4-tuple, (width, height, descent, externalLeading)

4.238.44 wxWindow::GetTitle

virtual wxString GetTitle()

Gets the window's title. Applicable only to frames and dialogs.

See also

wxWindow::SetTitle (p. 841)

4.238.45 wxWindow::GetUpdateRegion

virtual wxRegion GetUpdateRegion() const

Returns the region specifying which parts of the window have been damaged. Should only be called within an *OnPaint* (p. 825) event handler.

[See also](#)

wxRegion (p. 562), *wxRegionIterator* (p. 566), *wxWindow::OnPaint* (p. 825)

4.238.46 wxWindow::GetWindowStyleFlag

long GetWindowStyleFlag() const

Gets the window style that was passed to the constructor or **Create** member.

4.238.47 wxWindow::InitDialog

void InitDialog()

Sends an *wxWindow::OnInitDialog* (p. 823) event, which in turn transfers data to the dialog via validators.

[See also](#)

wxWindow::OnInitDialog (p. 823)

4.238.48 wxWindow::IsEnabled

virtual bool IsEnabled() const

Returns TRUE if the window is enabled for input, FALSE otherwise.

[See also](#)

wxWindow::Enable (p. 805)

4.238.49 wxWindow::IsRetained

virtual bool IsRetained() const

Returns TRUE if the window is retained, FALSE otherwise.

Remarks

Retained windows are only available on X platforms.

4.238.50 **wxWindow::IsShown**

virtual bool IsShown() const

Returns TRUE if the window is shown, FALSE if it has been hidden.

4.238.51 **wxWindow::Layout**

void Layout()

Invokes the constraint-based layout algorithm for this window. It is called automatically by the default **wxWindow::OnSize** member.

4.238.52 **wxWindow::LoadFromResource**

virtual bool LoadFromResource(wxWindow* parent, const wxString& resourceName, const wxResourceTable* resourceTable = NULL)

Loads a panel or dialog from a resource file.

Parameters

parent

Parent window.

resourceName

The name of the resource to load.

resourceTable

The resource table to load it from. If this is NULL, the default resource table will be used.

Return value

TRUE if the operation succeeded, otherwise FALSE.

4.238.53 **wxWindow::Lower**

void Lower()

Lowers the window to the bottom of the window hierarchy if it is a managed window (dialog or frame).

4.238.54 **wxWindow::MakeModal**

virtual void MakeModal(const bool *flag*)

Disables all other windows in the application so that the user can only interact with this window.

Parameters

flag

If TRUE, this call disables all other windows in the application so that the user can only interact with this window. If FALSE, the effect is reversed.

4.238.55 **wxWindow::Move**

void Move(int *x*, int *y*)

void Move(const wxPoint& *pt*)

Moves the window to the given position.

Parameters

x

Required x position.

y

Required y position.

pt

wxPoint (p. 503) object representing the position.

Remarks

Implementations of `SetSize` can also implicitly implement the `wxWindow::Move` function, which is defined in the base `wxWindow` class as the call:

```
SetSize(x, y, -1, -1, wxSIZE_USE_EXISTING);
```

See also

wxWindow::SetSize (p. 839)

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

Move(point)
MoveXY(x, y)

Accepts a wxPoint
Accepts a pair of integers

4.238.56 **wxWindow::OnActivate**

void OnActivate(wxActivateEvent& event)

Called when a window is activated or deactivated.

Parameters

event
Object containing activation information.

Remarks

If the window is being activated, *wxActivateEvent::GetActive* (p. 6) returns TRUE, otherwise it returns FALSE (it is being deactivated).

See also

wxActivateEvent (p. 5), *Event handling overview* (p. 939)

4.238.57 **wxWindow::OnChar**

void OnChar(wxKeyEvent& event)

Called when the user has pressed a key that is not a modifier (SHIFT, CONTROL or ALT).

Parameters

event
Object containing keypress information. See *wxKeyEvent* (p. 359) for details about this class.

Remarks

This member function is called in response to a keypress. To intercept this event, use the EVT_CHAR macro in an event table definition. Your **OnChar** handler may call this default function to achieve default keypress functionality.

Note that the ASCII values do not have explicit key codes: they are passed as ASCII values.

Note that not all keypresses can be intercepted this way. If you wish to intercept modifier keypresses, then you will need to use *wxWindow::OnKeyDown* (p. 821)

or `wxWindow::OnKeyUp` (p. 822).

Most, but not all, windows allow keypresses to be intercepted.

See also

`wxWindow::OnKeyDown` (p. 821), `wxWindow::OnKeyUp` (p. 822), `wxKeyEvent` (p. 359), `wxWindow::OnCharHook` (p. 818), *Event handling overview* (p. 939)

4.238.58 `wxWindow::OnCharHook`

void OnCharHook(wxKeyEvent& event)

This member is called to allow the window to intercept keyboard events before they are processed by child windows.

Parameters

event

Object containing keypress information. See `wxKeyEvent` (p. 359) for details about this class.

Remarks

This member function is called in response to a keypress, if the window is active. To intercept this event, use the `EVT_CHAR_HOOK` macro in an event table definition. If you do not process a particular keypress, call `wxEvent::Skip` (p. 224) to allow default processing.

An example of using this function is in the implementation of escape-character processing for `wxDialog`, where pressing ESC dismisses the dialog by **OnCharHook** 'forging' a cancel button press event.

Note that the ASCII values do not have explicit key codes: they are passed as ASCII values.

This function is only relevant to top-level windows (frames and dialogs), and under Windows only.

See also

`wxKeyEvent` (p. 359), `wxWindow::OnCharHook` (p. 818), `wxApp::OnCharHook` (p. 10), *Event handling overview* (p. 939)

4.238.59 `wxWindow::OnCommand`

virtual void OnCommand(wxEvtHandler& object, wxCommandEvent& event)

This virtual member function is called if the control does not handle the command event.

Parameters

object
Object receiving the command event.

event
Command event

Remarks

This virtual function is provided mainly for backward compatibility. You can also intercept commands from child controls by using an event table, with identifiers or identifier ranges to identify the control(s) in question.

See also

wxCommandEvent (p. 103), *Event handling overview* (p. 939)

4.238.60 **wxWindow::OnClose**

virtual bool OnClose()

Called when the user has tried to close a a frame or dialog box using the window manager (X) or system menu (Windows).

Note: This is an obsolete function. It is superceded by the *wxWindow::OnCloseWindow* (p. 819) event handler.

Return value

If TRUE is returned by OnClose, the window will be deleted by the system, otherwise the attempt will be ignored. Do not delete the window from within this handler, although you may delete other windows.

See also

Window deletion overview (p. 929), *wxWindow::Close* (p. 802), *wxWindow::OnCloseWindow* (p. 819), *wxCloseEvent* (p. 84)

4.238.61 **wxWindow::OnCloseWindow**

void OnCloseWindow(wxCloseEvent& event)

This is an event handler function called when the user has tried to close a a frame or dialog box using the window manager (X) or system menu (Windows). It is called via the *wxWindow::Close* (p. 802) function, so that the application can also invoke the handler programmatically.

Use the `EVT_CLOSE` event table macro to handle close events.

You should check whether the application is forcing the deletion of the window using `wxCloseEvent::GetForce` (p. 85). If this is `TRUE`, destroy the window using `wxWindow::Destroy` (p. 805). If not, it is up to you whether you respond by destroying the window.

(Note: `GetForce` is now superseded by `CanVeto`. So to test whether forced destruction of the window is required, test for the negative of `CanVeto`. If `CanVeto` returns `FALSE`, it is not possible to skip window deletion.)

If you don't destroy the window, you should call `wxCloseEvent::Veto` (p. 86) to let the calling code know that you did not destroy the window. This allows the `wxWindow::Close` (p. 802) function to return `TRUE` or `FALSE` depending on whether the close instruction was honoured or not.

Remarks

The `wxWindow::OnClose` (p. 819) virtual function remains for backward compatibility with earlier versions of wxWindows. The default **OnCloseWindow** handler for `wxFrame` and `wxDialog` will call **OnClose**, destroying the window if it returns `TRUE` or if the close is being forced.

See also

Window deletion overview (p. 929), `wxWindow::Close` (p. 802), `wxWindow::OnClose` (p. 819), `wxWindow::Destroy` (p. 805), `wxCloseEvent` (p. 84), `wxApp::OnQueryEndSession` (p. 12), `wxApp::OnEndSession` (p. 11)

4.238.62 wxWindow::OnDropFiles

void OnDropFiles(wxDropFilesEvent& event)

Called when files have been dragged from the file manager to the window.

Parameters

event

Drop files event. For more information, see `wxDropFilesEvent` (p. 214).

Remarks

The window must have previously been enabled for dropping by calling `wxWindow::DragAcceptFiles` (p. 805).

This event is only generated under Windows.

To intercept this event, use the `EVT_DROP_FILES` macro in an event table definition.

See also

wxDropFilesEvent (p. 214), *wxWindow::DragAcceptFiles* (p. 805), *Event handling overview* (p. 939)

4.238.63 **wxWindow::OnEraseBackground**

void OnEraseBackground(wxEraseEvent& event)

Called when the background of the window needs to be erased.

Parameters

event

Erase background event. For more information, see *wxEraseEvent* (p. 220).

Remarks

This event is only generated under Windows.

To intercept this event, use the `EVT_ERASE_BACKGROUND` macro in an event table definition.

See also

wxEraseEvent (p. 220), *Event handling overview* (p. 939)

4.238.64 **wxWindow::OnKeyDown**

void OnKeyDown(wxKeyEvent& event)

Called when the user has pressed a key, before it is translated into an ASCII value using other modifier keys that might be pressed at the same time.

Parameters

event

Object containing keypress information. See *wxKeyEvent* (p. 359) for details about this class.

Remarks

This member function is called in response to a key down event. To intercept this event, use the `EVT_KEY_DOWN` macro in an event table definition. Your **OnKeyDown** handler may call this default function to achieve default keypress functionality.

Note that not all keypresses can be intercepted this way. If you wish to intercept special keys, such as shift, control, and function keys, then you will need to use *wxWindow::OnKeyDown* (p. 821) or *wxWindow::OnKeyUp* (p. 822).

Most, but not all, windows allow keypresses to be intercepted.

See also

wxWindow::OnChar (p. 817), *wxWindow::OnKeyUp* (p. 822), *wxKeyEvent* (p. 359), *wxWindow::OnCharHook* (p. 818), *Event handling overview* (p. 939)

4.238.65 **wxWindow::OnKeyUp**

void OnKeyUp(wxKeyEvent& event)

Called when the user has released a key.

Parameters

event

Object containing keypress information. See *wxKeyEvent* (p. 359) for details about this class.

Remarks

This member function is called in response to a key up event. To intercept this event, use the `EVT_KEY_UP` macro in an event table definition. Your **OnKeyUp** handler may call this default function to achieve default keypress functionality.

Note that not all keypresses can be intercepted this way. If you wish to intercept special keys, such as shift, control, and function keys, then you will need to use *wxWindow::OnKeyDown* (p. 821) or *wxWindow::OnKeyUp* (p. 822).

Most, but not all, windows allow key up events to be intercepted.

See also

wxWindow::OnChar (p. 817), *wxWindow::OnKeyDown* (p. 821), *wxKeyEvent* (p. 359), *wxWindow::OnCharHook* (p. 818), *Event handling overview* (p. 939)

4.238.66 **wxWindow::OnKillFocus**

void OnKillFocus(wxFocusEvent& event)

Called when a window's focus is being killed.

Parameters

event

The focus event. For more information, see *wxFocusEvent* (p. 263).

Remarks

To intercept this event, use the macro `EVT_KILL_FOCUS` in an event table definition.

Most, but not all, windows respond to this event.

See also

wxFocusEvent (p. 263), *wxWindow::OnSetFocus* (p. 827), *Event handling overview* (p. 939)

4.238.67 **wxWindow::OnIdle**

void OnIdle(wxIdleEvent& event)

Provide this member function for any processing which needs to be done when the application is idle.

See also

wxApp::OnIdle (p. 11), *wxIdleEvent* (p. 317)

4.238.68 **wxWindow::OnInitDialog**

void OnInitDialog(wxInitDialogEvent& event)

Default handler for the `wxEVT_INIT_DIALOG` event. Calls *wxWindow::TransferDataToWindow* (p. 842).

Parameters

event
Dialog initialisation event.

Remarks

Gives the window the default behaviour of transferring data to child controls via the validator that each control has.

See also

wxValidator (p. 781), *wxWindow::TransferDataToWindow* (p. 842)

4.238.69 **wxWindow::OnMenuCommand**

void OnMenuCommand(wxCommandEvent& event)

Called when a menu command is received from a menu bar.

Parameters

event

The menu command event. For more information, see *wxCommandEvent* (p. 103).

Remarks

A function with this name doesn't actually exist; you can choose any member function to receive menu command events, using the `EVT_COMMAND` macro for individual commands or `EVT_COMMAND_RANGE` for a range of commands.

See also

wxCommandEvent (p. 103), *wxWindow::OnMenuHighlight* (p. 824), *Event handling overview* (p. 939)

4.238.70 **wxWindow::OnMenuHighlight**

void OnMenuHighlight(wxMenuEvent& *event*)

Called when a menu select is received from a menu bar: that is, the mouse cursor is over a menu item, but the left mouse button has not been pressed.

Parameters

event

The menu highlight event. For more information, see *wxMenuEvent* (p. 438).

Remarks

You can choose any member function to receive menu select events, using the `EVT_MENU_HIGHLIGHT` macro for individual menu items or `EVT_MENU_HIGHLIGHT_ALL` macro for all menu items.

The default implementation for *wxFrame::OnMenuHighlight* (p. 283) displays help text in the first field of the status bar.

This function was known as **OnMenuSelect** in earlier versions of wxWindows, but this was confusing since a selection is normally a left-click action.

See also

wxMenuEvent (p. 438), *wxWindow::OnMenuCommand* (p. 823), *Event handling overview* (p. 939)

4.238.71 **wxWindow::OnMouseEvent**

void OnMouseEvent(wxMouseEvent& *event*)

Called when the user has initiated an event with the mouse.

Parameters

event

The mouse event. See *wxMouseEvent* (p. 450) for more details.

Remarks

Most, but not all, windows respond to this event.

To intercept this event, use the `EVT_MOUSE_EVENTS` macro in an event table definition, or individual mouse event macros such as `EVT_LEFT_DOWN`.

See also

wxMouseEvent (p. 450), *Event handling overview* (p. 939)

4.238.72 **wxWindow::OnMove**

void OnMove(wxMoveEvent& *event*)

Called when a window is moved.

Parameters

event

The move event. For more information, see *wxMoveEvent* (p. 458).

Remarks

Use the `EVT_MOVE` macro to intercept move events.

Remarks

Not currently implemented.

See also

wxMoveEvent (p. 458), *wxFrame::OnSize* (p. 283), *Event handling overview* (p. 939)

4.238.73 **wxWindow::OnPaint**

void OnPaint(wxPaintEvent& *event*)

Sent to the event handler when the window must be refreshed.

Parameters

event

Paint event. For more information, see *wxPaintEvent* (p. 484).

Remarks

Use the `EVT_PAINT` macro in an event table definition to intercept paint events.

In a paint event handler, the application should always create a *wxPaintDC* (p. 483) object.

For example:

```
void MyWindow::OnPaint(wxPaintEvent& event)
{
    wxPaintDC dc(this);

    DrawMyDocument(dc);
}
```

You can optimize painting by retrieving the rectangles that have been damaged and only repainting these. The rectangles are in terms of the client area, and are unscrolled, so you will need to do some calculations using the current view position to obtain logical, scrolled units.

Here is an example of using the *wxRegionIterator* (p. 566) class:

```
// Called when window needs to be repainted.
void MyWindow::OnPaint(wxPaintEvent& event)
{
    wxPaintDC dc(this);

    // Find Out where the window is scrolled to
    int vbX,vbY; // Top left corner of client
    ViewStart(&vbX,&vbY);

    int vX,vY,vW,vH; // Dimensions of client area in
pixels
    wxRegionIterator upd(GetUpdateRegion()); // get the update rect list

    while (upd)
    {
        vX = upd.GetX();
        vY = upd.GetY();
        vW = upd.GetW();
        vH = upd.GetH();

        // Alternatively we can do this:
        // wxRect rect;
        // upd.GetRect(&rect);

        // Repaint this rectangle
        ...some code...

        upd ++ ;
    }
}
```

```
}
```

See also

wxPaintEvent (p. 484), *wxPaintDC* (p. 483), *Event handling overview* (p. 939)

4.238.74 wxWindow::OnScroll**void OnScroll(wxScrollEvent& event)**

Called when a scroll event is received from one of the window's built-in scrollbars.

Parameters

event

Command event. Retrieve the new scroll position by calling *wxScrollEvent::GetPosition* (p. 586), and the scrollbar orientation by calling *wxScrollEvent::GetOrientation* (p. 586).

Remarks

Note that it is not possible to distinguish between horizontal and vertical scrollbars until the function is executing (you can't have one function for vertical, another for horizontal events).

See also

wxScrollEvent (p. 584), *Event handling overview* (p. 939)

4.238.75 wxWindow::OnSetFocus**void OnSetFocus(wxFocusEvent& event)**

Called when a window's focus is being set.

Parameters

event

The focus event. For more information, see *wxFocusEvent* (p. 263).

Remarks

To intercept this event, use the macro `EVT_SET_FOCUS` in an event table definition.

Most, but not all, windows respond to this event.

See also

wxFocusEvent (p. 263), *wxWindow::OnKillFocus* (p. 822), *Event handling overview* (p.

939)

4.238.76 **wxWindow::OnSize**

void OnSize(wxSizeEvent& event)

Called when the window has been resized.

Parameters

event

Size event. For more information, see *wxSizeEvent* (p. 596).

Remarks

You may wish to use this for frames to resize their child windows as appropriate.

Note that the size passed is of the whole window: call *wxWindow::GetClientSize* (p. 807) for the area which may be used by the application.

See also

wxSizeEvent (p. 596), *Event handling overview* (p. 939)

4.238.77 **wxWindow::OnSysColourChanged**

void OnSysColourChanged(wxOnSysColourChangedEvent& event)

Called when the user has changed the system colours.

Parameters

event

System colour change event. For more information, see *wxSysColourChangedEvent* (p. 678).

See also

wxSysColourChangedEvent (p. 678), *Event handling overview* (p. 939)

4.238.78 **wxWindow::PopEventHandler**

wxEvtHandler* PopEventHandler(bool deleteHandler = FALSE) const

Removes and returns the top-most event handler on the event handler stack.

Parameters

deleteHandler

If this is TRUE, the handler will be deleted after it is removed. The default value is FALSE.

See also

wxWindow::SetEventHandler (p. 834), *wxWindow::GetEventHandler* (p. 808),
wxWindow::PushEventHandler (p. 829), *wxEvtHandler::ProcessEvent* (p. 227),
wxEvtHandler (p. 224)

4.238.79 wxWindow::PopupMenu

virtual bool PopupMenu(wxMenu* menu, int x, int y)

Pops up the given menu at the specified coordinates, relative to this window, and returns control when the user has dismissed the menu. If a menu item is selected, the callback defined for the menu is called with wxMenu and wxCommandEvent reference arguments. The callback should access the commandInt member of the event to check the selected menu identifier.

Parameters

menu

Menu to pop up.

x

Required x position for the menu to appear.

y

Required y position for the menu to appear.

See also

wxMenu (p. 418)

Remarks

Just before the menu is popped up, *wxMenu::UpdateUI* (p. 426) is called to ensure that the menu items are in the correct state.

4.238.80 wxWindow::PushEventHandler

void PushEventHandler(wxEvtHandler* handler)

Pushes this event handler onto the event stack for the window.

Parameters

handler

Specifies the handler to be pushed.

Remarks

An event handler is an object that is capable of processing the events sent to a window. By default, the window is its own event handler, but an application may wish to substitute another, for example to allow central implementation of event-handling for a variety of different window classes.

wxWindow::PushEventHandler (p. 829) allows an application to set up a chain of event handlers, where an event not handled by one event handler is handed to the next one in the chain. Use *wxWindow::PopEventHandler* (p. 828) to remove the event handler.

See also

wxWindow::SetEventHandler (p. 834), *wxWindow::GetEventHandler* (p. 808), *wxWindow::PopEventHandler* (p. 829), *wxEvtHandler::ProcessEvent* (p. 227), *wxEvtHandler* (p. 224)

4.238.81 **wxWindow::Raise**

void Raise()

Raises the window to the top of the window hierarchy if it is a managed window (dialog or frame).

4.238.82 **wxWindow::Refresh**

virtual void Refresh(const bool eraseBackground = TRUE, const wxRect* rect = NULL)

Causes a message or event to be generated to repaint the window.

Parameters

eraseBackground

If TRUE, the background will be erased.

rect

If non-NULL, only the given rectangle will be treated as damaged.

4.238.83 **wxWindow::ReleaseMouse**

virtual void ReleaseMouse()

Releases mouse input captured with *wxWindow::CaptureMouse* (p. 801).

See also

wxWindow::CaptureMouse (p. 801)

4.238.84 wxWindow::RemoveChild

virtual void RemoveChild(*wxWindow child)**

Removes a child window. This is called automatically by window deletion functions so should not be required by the application programmer.

Parameters

child

Child window to remove.

4.238.85 wxWindow::ScreenToClient

virtual void ScreenToClient(*int x, *int** y) const**

virtual wxPoint ScreenToClient(const wxPoint& pt) const

Converts from screen to client window coordinates.

Parameters

x

Stores the screen x coordinate and receives the client x coordinate.

y

Stores the screen x coordinate and receives the client x coordinate.

pt

The screen position for the second form of the function.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

ScreenToClient(point)	Accepts and returns a wxPoint
ScreenToClientXY(x, y)	Returns a 2-tuple, (x, y)

4.238.86 wxWindow::ScrollWindow

virtual void ScrollWindow(*int dx*, *int dy*, const wxRect* rect = NULL)

Physically scrolls the pixels in the window.

Parameters

dx

Amount to scroll horizontally.

dy

Amount to scroll vertically.

rect

Rectangle to invalidate. If this is NULL, the whole window is invalidated. If you pass a rectangle corresponding to the area of the window exposed by the scroll, your painting handler can optimise painting by checking for the invalidated region.

Remarks

Available only under Windows.

Use this function to optimise your scrolling implementations, to minimise the area that must be redrawn.

4.238.87 **wxWindow::SetAcceleratorTable**

virtual void SetAcceleratorTable(const wxAcceleratorTable& *accel*)

Sets the accelerator table for this window. See *wxAcceleratorTable* (p. 2).

4.238.88 **wxWindow::SetAutoLayout**

void SetAutoLayout(const bool *autoLayout*)

Determines whether the *wxWindow::Layout* (p. 815) function will be called automatically when the window is resized.

Parameters

autoLayout

Set this to TRUE if you wish the Layout function to be called from within *wxWindow::OnSize* functions.

See also

wxWindow::SetConstraints (p. 835)

4.238.89 **wxWindow::SetBackgroundColour**

virtual void SetBackgroundColour(const wxColour& colour)

Sets the background colour of the window.

Parameters

colour

The colour to be used as the background colour.

Remarks

The background colour is usually painted by the default *wxWindow::OnEraseBackground* (p. 821) event handler function.

Note that setting the background colour does not cause an immediate refresh, so you may wish to call *wxWindow::Clear* (p. 802) or *wxWindow::Refresh* (p. 830) after calling this function.

See also

wxWindow::GetBackgroundColour (p. 807), *wxWindow::SetForegroundColour* (p. 836), *wxWindow::GetForegroundColour* (p. 809), *wxWindow::Clear* (p. 802), *wxWindow::Refresh* (p. 830), *wxWindow::OnEraseBackground* (p. 821)

4.238.90 **wxWindow::SetClientSize**

virtual void SetClientSize(int width, int height)

virtual void SetClientSize(const wxSize& size)

This sets the size of the window client area in pixels. Using this function to size a window tends to be more device-independent than *wxWindow::SetSize* (p. 839), since the application need not worry about what dimensions the border or title bar have when trying to fit the window around panel items, for example.

Parameters

width

The required client area width.

height

The required client area height.

size

The required client size.

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

SetClientSize(size) Accepts a `wxSize`
SetClientSizeWH(width, height)

4.238.91 **wxWindow::SetCursor**

virtual void SetCursor(const wxCursor&cursor)

Sets the window's cursor. Notice that setting the cursor for this window does not set it for its children so you'll need to explicitly call `SetCursor()` for them too if you need it.

Parameters

cursor

Specifies the cursor that the window should normally display.

See also

`::wxSetCursor` (p. 857), `wxCursor` (p. 128)

4.238.92 **wxWindow::SetEventHandler**

void SetEventHandler(wxEvtHandler* handler)

Sets the event handler for this window.

Parameters

handler

Specifies the handler to be set.

Remarks

An event handler is an object that is capable of processing the events sent to a window. By default, the window is its own event handler, but an application may wish to substitute another, for example to allow central implementation of event-handling for a variety of different window classes.

It is usually better to use `wxWindow::PushEventHandler` (p. 829) since this sets up a chain of event handlers, where an event not handled by one event handler is handed to the next one in the chain.

See also

`wxWindow::GetEventHandler` (p. 808), `wxWindow::PushEventHandler` (p. 829),
`wxWindow::PopEventHandler` (p. 829), `wxEvtHandler::ProcessEvent` (p. 227),
`wxEvtHandler` (p. 224)

4.238.93 wxWindow::SetConstraints**void SetConstraints(wxLayoutConstraints* constraints)**

Sets the window to have the given layout constraints. The window will then own the object, and will take care of its deletion. If an existing layout constraints object is already owned by the window, it will be deleted.

Parameters*constraints*

The constraints to set. Pass NULL to disassociate and delete the window's constraints.

Remarks

You must call *wxWindow::SetAutoLayout* (p. 832) to tell a window to use the constraints automatically in *OnSize*; otherwise, you must override *OnSize* and call *Layout* explicitly.

4.238.94 wxWindow::SetDropTarget**void SetDropTarget(wxDropTarget* target)**

Associates a drop target with this window.

If the window already has a drop target, it is deleted.

See also

wxWindow::GetDropTarget (p. 808), *Drag and drop overview* (p. 975)

4.238.95 wxWindow::SetFocus**virtual void SetFocus()**

This sets the window to receive keyboard input.

4.238.96 wxWindow::SetFont**void SetFont(const wxFont& font)**

Sets the font for this window.

Parameters*font*

Font to associate with this window.

See also

wxWindow::GetFont (p. 809)

4.238.97 wxWindow::SetForegroundColour

virtual void SetForegroundColour(const wxColour& colour)

Sets the foreground colour of the window.

Parameters

colour

The colour to be used as the foreground colour.

Remarks

The interpretation of foreground colour is open to interpretation according to the window class; it may be the text colour or other colour, or it may not be used at all.

See also

wxWindow::GetForegroundColour (p. 809), *wxWindow::SetBackgroundColour* (p. 832), *wxWindow::GetBackgroundColour* (p. 807)

4.238.98 wxWindow::SetId

void SetId(int id)

Sets the identifier of the window.

Remarks

Each window has an integer identifier. If the application has not provided one, an identifier will be generated. Normally, the identifier should be provided on creation and should not be modified subsequently.

See also

wxWindow::GetId (p. 809), *Window identifiers* (p. 943)

4.238.99 wxWindow::SetName

virtual void SetName(const wxString& name)

Sets the window's name.

Parameters

name

A name to set for the window.

See also

wxWindow::GetName (p. 810)

4.238.100 **wxWindow::SetPalette**

virtual void SetPalette(*wxPalette* palette*)

Obsolete - use *wxDC::SetPalette* (p. 163) instead.

4.238.101 **wxWindow::SetReturnCode**

void SetReturnCode(*int retCode*)

Sets the return code for this window.

Parameters

retCode

The integer return code, usually a control identifier.

Remarks

A return code is normally associated with a modal dialog, where *wxDialog::ShowModal* (p. 185) returns a code to the application. The function *wxDialog::EndModal* (p. 181) calls **SetReturnCode**.

See also

wxWindow::GetReturnCode (p. 811), *wxDialog::ShowModal* (p. 185),
wxDialog::EndModal (p. 181)

4.238.102 **wxWindow::SetScrollbar**

virtual void SetScrollbar(*int orientation, int position, int thumbSize, int range, const bool refresh = TRUE*)

Sets the scrollbar properties of a built-in scrollbar.

Parameters

orientation

Determines the scrollbar whose page size is to be set. May be `wxHORIZONTAL` or `wxVERTICAL`.

position

The position of the scrollbar in scroll units.

thumbSize

The size of the thumb, or visible portion of the scrollbar, in scroll units.

range

The maximum position of the scrollbar.

refresh

TRUE to redraw the scrollbar, FALSE otherwise.

Remarks

Let's say you wish to display 50 lines of text, using the same font. The window is sized so that you can only see 16 lines at a time.

You would use:

```
SetScrollbar(wxVERTICAL, 0, 16, 50);
```

Note that with the window at this size, the thumb position can never go above 50 minus 16, or 34.

You can determine how many lines are currently visible by dividing the current view size by the character height in pixels.

When defining your own scrollbar behaviour, you will always need to recalculate the scrollbar settings when the window size changes. You could therefore put your scrollbar calculations and `SetScrollbar` call into a function named `AdjustScrollbars`, which can be called initially and also from your `wxWindow::OnSize` (p. 828) event handler function.

See also

Scrolling overview (p. 932), *wxScrollBar* (p. 579), *wxScrolledWindow* (p. 586)

4.238.103 wxWindow::SetScrollPos

virtual void SetScrollPos(int *orientation*, int *pos*, **const bool** *refresh* = TRUE)

Sets the position of one of the built-in scrollbars.

Parameters

orientation

Determines the scrollbar whose position is to be set. May be `wxHORIZONTAL` or `wxVERTICAL`.

pos

Position in scroll units.

refresh

TRUE to redraw the scrollbar, FALSE otherwise.

Remarks

This function does not directly affect the contents of the window: it is up to the application to take note of scrollbar attributes and redraw contents accordingly.

See also

wxWindow::SetScrollbar (p. 837), *wxWindow::GetScrollPos* (p. 838),
wxWindow::GetScrollThumb (p. 811), *wxScrollBar* (p. 579), *wxScrolledWindow* (p. 586)

4.238.104 **wxWindow::SetSize**

virtual void SetSize(int x, int y, int width, int height, int sizeFlags = wxSIZE_AUTO)

virtual void SetSize(const wxRect& rect)

Sets the size and position of the window in pixels.

virtual void SetSize(int width, int height)

virtual void SetSize(const wxSize& size)

Sets the size of the window in pixels.

Parameters

x

Required x position in pixels, or -1 to indicate that the existing value should be used.

y

Required y position in pixels, or -1 to indicate that the existing value should be used.

width

Required width in pixels, or -1 to indicate that the existing value should be used.

height

Required height position in pixels, or -1 to indicate that the existing value should be used.

size

wxSize (p. 595) object for setting the size.

rect

wxRect (p. 546) object for setting the position and size.

sizeFlags

Indicates the interpretation of other parameters. It is a bit list of the following:

wxSIZE_AUTO_WIDTH: a -1 width value is taken to indicate a wxWindows-supplied default width.

wxSIZE_AUTO_HEIGHT: a -1 height value is taken to indicate a wxWindows-supplied default width.

wxSIZE_AUTO: -1 size values are taken to indicate a wxWindows-supplied default size.

wxSIZE_USE_EXISTING: existing dimensions should be used if -1 values are supplied.

wxSIZE_ALLOW_MINUS_ONE: allow dimensions of -1 and less to be interpreted as real dimensions, not default values.

Remarks

The second form is a convenience for calling the first form with default x and y parameters, and must be used with non-default width and height values.

The first form sets the position and optionally size, of the window. Parameters may be -1 to indicate either that a default should be supplied by wxWindows, or that the current value of the dimension should be used.

See also

wxWindow::Move (p. 816)

wxPython note:

In place of a single overloaded method name, wxPython implements the following methods:

```
SetDimensions(x, y, width, height, sizeFlags=wxSIZE_AUTO)
SetSize(size)
SetPosition(point)
```

4.238.105 wxWindow::SetSizeHints

```
virtual void SetSizeHints(int minW=-1, int minH=-1, int maxW=-1, int maxH=-1, int
incW=-1, int incH=-1)
```


Allows specification of minimum and maximum window sizes, and window size increments. If a pair of values is not set (or set to -1), the default values will be used.

Parameters

minW

Specifies the minimum width allowable.

minH

Specifies the minimum height allowable.

maxW

Specifies the maximum width allowable.

maxH

Specifies the maximum height allowable.

incW

Specifies the increment for sizing the width (Motif/Xt only).

incH

Specifies the increment for sizing the height (Motif/Xt only).

Remarks

If this function is called, the user will not be able to size the window outside the given bounds.

The resizing increments are only significant under Motif or Xt.

4.238.106 **wxWindow::SetTitle**

virtual void SetTitle(const wxString& title)

Sets the window's title. Applicable only to frames and dialogs.

Parameters

title

The window's title.

See also

wxWindow::GetTitle (p. 813)

4.238.107 **wxWindow::Show**

virtual bool Show(const bool show)

Shows or hides the window.

Parameters

show

If TRUE, displays the window and brings it to the front. Otherwise, hides the window.

See also

wxWindow::IsShown (p. 815)

4.238.108 **wxWindow::TransferDataFromWindow**

virtual bool TransferDataFromWindow()

Transfers values from child controls to data areas specified by their validators. Returns FALSE if a transfer failed.

See also

wxWindow::TransferDataToWindow (p. 842), *wxValidator* (p. 781), *wxWindow::Validate* (p. 842)

4.238.109 **wxWindow::TransferDataToWindow**

virtual bool TransferDataToWindow()

Transfers values to child controls from data areas specified by their validators.

Return value

Returns FALSE if a transfer failed.

See also

wxWindow::TransferDataFromWindow (p. 842), *wxValidator* (p. 781), *wxWindow::Validate* (p. 842)

4.238.110 **wxWindow::Validate**

virtual bool Validate()

Validates the current values of the child controls using their validators.

Return value

Returns FALSE if any of the validations failed.

See also

wxWindow::TransferDataFromWindow (p. 842), *wxWindow::TransferDataFromWindow* (p. 842), *wxValidator* (p. 781)

4.238.111 **wxWindow::WarpPointer**

void WarpPointer(int x, int y)

Moves the pointer to the given position on the window.

Parameters

x
The new x position for the cursor.

y
The new y position for the cursor.

4.239 **wxWindowDC**

A *wxWindowDC* must be constructed if an application wishes to paint on the whole area of a window (client and decorations). This should normally be constructed as a temporary stack object; don't store a *wxWindowDC* object.

To draw on a window from inside **OnPaint**, construct a *wxPaintDC* (p. 483) object.

To draw on the client area of a window from outside **OnPaint**, construct a *wxClientDC* (p. 81) object.

To draw on the whole window including decorations, construct a *wxWindowDC* (p. 843) object (Windows only).

Derived from

wxDC (p. 151)

Include files

<wx/dcclient.h>

See also

wxDC (p. 151), *wxMemoryDC* (p. 415), *wxPaintDC* (p. 483), *wxClientDC* (p. 81), *wxScreenDC* (p. 578)

4.239.1 wxWindowDC::wxWindowDC

wxWindowDC(wxWindow* *window*)

Constructor. Pass a pointer to the window on which you wish to paint.

4.240 wxZlibInputStream

Derived from

wxFilterInputStream (p. 262)

Include files

<wx/zstream.h>

See also

wxStreamBuffer (p. 648), *wxInputStream* (p. 346)

Short description

This stream uncompresses all data read from it. It uses the "filtered" stream to get new compressed data.

4.241 wxZlibOutputStream

Derived from

wxFilterOutputStream (p. 262)

Include files

<wx/zstream.h>

See also

wxStreamBuffer (p. 648), *wxOutputStream* (p. 476)

Short description

This stream compresses all data written to it, and passes the compressed data to the "filtered" stream.

5 Functions

The functions defined in `wxWindows` are described here.

5.1 File functions

Include files

`<wx/utils.h>`

See also

wxPathList (p. 492)

5.1.1 `::wxDirExists`

`bool wxDirExists(const wxString& dirname)`

Returns TRUE if the directory exists.

5.1.2 `::wxDos2UnixFilename`

`void Dos2UnixFilename(const wxString& s)`

Converts a DOS to a Unix filename by replacing backslashes with forward slashes.

5.1.3 `::wxFileExists`

`bool wxFileExists(const wxString& filename)`

Returns TRUE if the file exists.

5.1.4 `::wxFileNameFromPath`

`wxString wxFileNameFromPath(const wxString& path)`

`char* wxFileNameFromPath(char* path)`

Returns the filename for a full path. The second form returns a pointer to temporary storage that should not be deallocated.

5.1.5 `::wxFindFirstFile`

wxString wxFindFirstFile(const char*spec, int flags = 0)

This function does directory searching; returns the first file that matches the path *spec*, or the empty string. Use *wxFindNextFile* (p. 846) to get the next matching file.

spec may contain wildcards.

flags is reserved for future use.

For example:

```
wxString f = wxFindFirstFile("/home/project/*.");
while ( !f.IsEmpty() )
{
    ...
    f = wxFindNextFile();
}
```

5.1.6 ::wxFindNextFile**wxString wxFindFirstFile()**

Returns the next file that matches the path passed to *wxFindFirstFile* (p. 845).

5.1.7 ::wxGetOSDirectory**wxString wxGetOSDirectory()**

Returns the Windows directory under Windows; on other platforms returns the empty string.

5.1.8 ::wxIsAbsolutePath**bool wxIsAbsolutePath(const wxString& filename)**

Returns TRUE if the argument is an absolute filename, i.e. with a slash or drive name at the beginning.

5.1.9 ::wxPathOnly**wxString wxPathOnly(const wxString& path)**

Returns the directory part of the filename.

5.1.10 ::wxUnix2DosFilename

void wxUnix2DosFilename(const wxString& s)

Converts a Unix to a DOS filename by replacing forward slashes with backslashes.

5.1.11 ::wxConcatFiles

bool wxConcatFiles(const wxString& file1, const wxString& file2, const wxString& file3)

Concatenates *file1* and *file2* to *file3*, returning TRUE if successful.

5.1.12 ::wxCopyFile

bool wxCopyFile(const wxString& file1, const wxString& file2)

Copies *file1* to *file2*, returning TRUE if successful.

5.1.13 ::wxGetCwd

wxString wxGetCwd()

Returns a string containing the current (or working) directory.

5.1.14 ::wxGetHostName

bool wxGetHostName(const wxString& buf, int sz)

Copies the current host machine's name into the supplied buffer.

Under Windows or NT, this function first looks in the environment variable `SYSTEM_NAME`; if this is not found, the entry **HostName** in the **wxWindows** section of the WIN.INI file is tried.

Returns TRUE if successful, FALSE otherwise.

5.1.15 ::wxGetEmailAddress

bool wxGetEmailAddress(const wxString& buf, int sz)

Copies the user's email address into the supplied buffer, by concatenating the values returned by *wxGetHostName* (p. 847) and *wxGetUserId* (p. 848).

Returns TRUE if successful, FALSE otherwise.

5.1.16 ::wxGetUserId

bool wxGetUserId(const wxString& buf, int sz)

Copies the current user id into the supplied buffer.

Under Windows or NT, this function first looks in the environment variables **USER** and **LOGNAME**; if neither of these is found, the entry **UserId** in the **wxWindows** section of the WIN.INI file is tried.

Returns TRUE if successful, FALSE otherwise.

5.1.17 ::wxGetUserName

bool wxGetUserName(const wxString& buf, int sz)

Copies the current user name into the supplied buffer.

Under Windows or NT, this function looks for the entry **UserName** in the **wxWindows** section of the WIN.INI file. If PenWindows is running, the entry **Current** in the section **User** of the PENWIN.INI file is used.

Returns TRUE if successful, FALSE otherwise.

5.1.18 ::wxGetWorkingDirectory

wxString wxGetWorkingDirectory(char* buf=NULL, int sz=1000)

This function is obsolete: use *wxCwd* (p. 847) instead.

Copies the current working directory into the buffer if supplied, or copies the working directory into new storage (which you must delete yourself) if the buffer is NULL.

sz is the size of the buffer if supplied.

5.1.19 ::wxGetTempFileName

char* wxGetTempFileName(const wxString& prefix, char* buf=NULL)

Makes a temporary filename based on *prefix*, opens and closes the file, and places the name in *buf*. If *buf* is NULL, new store is allocated for the temporary filename using *new*.

Under Windows, the filename will include the drive and name of the directory allocated for temporary files (usually the contents of the TEMP variable). Under Unix, the `/tmp` directory is used.

It is the application's responsibility to create and delete the file.

5.1.20 ::wxIsWild

bool wxIsWild(const wxString& *pattern*)

Returns TRUE if the pattern contains wildcards. See *wxMatchWild* (p. 849).

5.1.21 ::wxMatchWild

bool wxMatchWild(const wxString& *pattern*, const wxString& *text*, bool *dot_special*)

Returns TRUE if the *pattern* matches the *text*; if *dot_special* is TRUE, filenames beginning with a dot are not matched with wildcard characters. See *wxIsWild* (p. 849).

5.1.22 ::wxMkdir

bool wxMkdir(const wxString& *dir*)

Makes the directory *dir*, returning TRUE if successful.

5.1.23 ::wxRemoveFile

bool wxRemoveFile(const wxString& *file*)

Removes *file*, returning TRUE if successful.

5.1.24 ::wxRenameFile

bool wxRenameFile(const wxString& *file1*, const wxString& *file2*)

Renames *file1* to *file2*, returning TRUE if successful.

5.1.25 ::wxRmdir

bool wxRmdir(const wxString& *dir*, int *flags*=0)

Removes the directory *dir*, returning TRUE if successful. Does not work under VMS.

The *flags* parameter is reserved for future use.

5.1.26 ::wxSetWorkingDirectory

bool wxSetWorkingDirectory(const wxString& *dir*)

Sets the current working directory, returning TRUE if the operation succeeded. Under MS Windows, the current drive is also changed if *dir* contains a drive specification.

5.1.27 ::wxSplitPath

void wxSplitPath(const char * fullname, const wxString * path, const wxString * name, const wxString * ext)

This function splits a full file name into components: the path (including possible disk/drive specification under Windows), the base name and the extension. Any of the output parameters (*path*, *name* or *ext*) may be NULL if you are not interested in the value of a particular component.

wxSplitPath() will correctly handle filenames with both DOS and Unix path separators under Windows, however it will not consider backslashes as path separators under Unix (where backslash is a valid character in a filename).

On entry, *fullname* should be non NULL (it may be empty though).

On return, *path* contains the file path (without the trailing separator), *name* contains the file name and *ext* contains the file extension without leading dot. All three of them may be empty if the corresponding component is. The old contents of the strings pointed to by these parameters will be overwritten in any case (if the pointers are not NULL).

5.1.28 ::wxTransferFileToStream

bool wxTransferFileToStream(const wxString& filename, ostream& stream)

Copies the given file to *stream*. Useful when converting an old application to use streams (within the document/view framework, for example).

Use of this function requires the file wx_doc.h to be included.

5.1.29 ::wxTransferStreamToFile

bool wxTransferStreamToFile(istream& stream const wxString& filename)

Copies the given stream to the file *filename*. Useful when converting an old application to use streams (within the document/view framework, for example).

Use of this function requires the file wx_doc.h to be included.

5.2 String functions

5.2.1 ::copystring

char* copystring(const char* s)

Makes a copy of the string *s* using the C++ new operator, so it can be deleted with the *delete* operator.

5.2.2 ::wxStringMatch

**bool wxStringMatch(const wxString& s1, const wxString& s2,
bool subString = TRUE, bool exact = FALSE)**

Returns TRUE if the substring *s1* is found within *s2*, ignoring case if *exact* is FALSE. If *subString* is FALSE, no substring matching is done.

5.2.3 ::wxStringEq

bool wxStringEq(const wxString& s1, const wxString& s2)

A macro defined as:

```
#define wxStringEq(s1, s2) (s1 && s2 && (strcmp(s1, s2) == 0))
```

5.2.4 ::IsEmpty

bool IsEmpty(const char * p)

Returns TRUE if the string is empty, FALSE otherwise. It is safe to pass NULL pointer to this function and it will return TRUE for it.

5.2.5 ::Stricmp

int Stricmp(const char *p1, const char *p2)

Returns a negative value, 0, or positive value if *p1* is less than, equal to or greater than *p2*. The comparison is case-insensitive.

This function complements the standard C function *strcmp()* which performs case-sensitive comparison.

5.2.6 ::Strlen

size_t Strlen(const char * p)

This is a safe version of standard function *strlen()*: it does exactly the same thing (i.e. returns the length of the string) except that it returns 0 if *p* is the NULL pointer.

5.2.7 ::wxGetTranslation

const char * wxGetTranslation(const char * str)

This function returns the translation of string *str* in the current *locale* (p. 396). If the string is not found in any of the loaded message catalogs (see *i18n overview* (p. 978)), the original string is returned. In debug build, an error message is logged - this should help to find the strings which were not yet translated. As this function is used very often, an alternative syntax is provided: the `_()` macro is defined as `wxGetTranslation()`.

5.3 Dialog functions

Below are a number of convenience functions for getting input from the user or displaying messages. Note that in these functions the last three parameters are optional. However, it is recommended to pass a parent frame parameter, or (in MS Windows or Motif) the wrong window frame may be brought to the front when the dialog box is popped up.

5.3.1 ::wxFileSelector

**wxString wxFileSelector(const wxString& message, const wxString& default_path = NULL,
const wxString& default_filename = NULL, const wxString& default_extension = NULL,
const wxString& wildcard = ".*.*", int flags = 0, wxWindow *parent = NULL,
int x = -1, int y = -1)**

Pops up a file selector box. In Windows, this is the common file selector dialog. In X, this is a file selector box with somewhat less functionality. The path and filename are distinct elements of a full file pathname. If path is NULL, the current directory will be used. If filename is NULL, no default filename will be supplied. The wildcard determines what files are displayed in the file selector, and file extension supplies a type extension for the required filename. Flags may be a combination of `wxOPEN`, `wxSAVE`, `wxOVERWRITE_PROMPT`, `wxHIDE_READONLY`, or 0. They are only significant at present in Windows.

Both the X and Windows versions implement a wildcard filter. Typing a filename containing wildcards (*, ?) in the filename text item, and clicking on Ok, will result in only those files matching the pattern being displayed. In the X version, supplying no default name will result in the wildcard filter being inserted in the filename text item; the filter is ignored if a default name is supplied.

Under Windows (only), the wildcard may be a specification for multiple types of file with a description for each, such as:

```
"BMP files (*.bmp) | *.bmp | GIF files (*.gif) | *.gif"
```

The application must check for a NULL return value (the user pressed Cancel). For

example:

```
const wxString& s = wxFileSelector("Choose a file to open");
if (s)
{
    ...
}
```

Include files

<wx/filedlg.h>

5.3.2 ::wxGetTextFromUser

wxString wxGetTextFromUser(const wxString& message, const wxString& caption = "Input text", const wxString& default_value = "", wxWindow *parent = NULL, int x = -1, int y = -1, bool centre = TRUE)

Pop up a dialog box with title set to *caption*, message *message*, and a *default_value*. The user may type in text and press OK to return this text, or press Cancel to return the empty string.

If *centre* is TRUE, the message text (which may include new line characters) is centred; if FALSE, the message is left-justified.

Include files

<wx/textdlg.h>

5.3.3 ::wxGetMultipleChoice

int wxGetMultipleChoice(const wxString& message, const wxString& caption, int n, const wxString& choices[], int nsel, int *selection, wxWindow *parent = NULL, int x = -1, int y = -1, bool centre = TRUE, int width=150, int height=200)

Pops up a dialog box containing a message, OK/Cancel buttons and a multiple-selection listbox. The user may choose one or more item(s) and press OK or Cancel.

The number of initially selected choices, and array of the selected indices, are passed in; this array will contain the user selections on exit, with the function returning the number of selections. *selection* must be as big as the number of choices, in case all are selected.

If Cancel is pressed, -1 is returned.

choices is an array of *n* strings for the listbox.

If *centre* is *TRUE*, the message text (which may include new line characters) is centred; if *FALSE*, the message is left-justified.

Include files

<wx/choicdlg.h>

5.3.4 ::wxGetSingleChoice

```
wxString wxGetSingleChoice(const wxString& message, const wxString& caption,  
int n, const wxString& choices[],  
wxWindow *parent = NULL, int x = -1, int y = -1,  
bool centre = TRUE, int width=150, int height=200)
```

Pops up a dialog box containing a message, OK/Cancel buttons and a single-selection listbox. The user may choose an item and press OK to return a string or Cancel to return the empty string.

choices is an array of *n* strings for the listbox.

If *centre* is *TRUE*, the message text (which may include new line characters) is centred; if *FALSE*, the message is left-justified.

Include files

<wx/choicdlg.h>

5.3.5 ::wxGetSingleChoiceIndex

```
int wxGetSingleChoiceIndex(const wxString& message, const wxString& caption,  
int n, const wxString& choices[],  
wxWindow *parent = NULL, int x = -1, int y = -1,  
bool centre = TRUE, int width=150, int height=200)
```

As **wxGetSingleChoice** but returns the index representing the selected string. If the user pressed cancel, -1 is returned.

Include files

<wx/choicdlg.h>

5.3.6 ::wxGetSingleChoiceData

```
wxString wxGetSingleChoiceData(const wxString& message, const wxString&  
caption, int n, const wxString& choices[],  
const wxString& client_data[], wxWindow *parent = NULL, int x = -1,  
int y = -1, bool centre = TRUE, int width=150, int height=200)
```

As **wxGetSingleChoice** but takes an array of client data pointers corresponding to the strings, and returns one of these pointers.

Include files

<wx/choicdlg.h>

5.3.7 ::wxMessageBox

```
int wxMessageBox(const wxString& message, const wxString& caption =  
"Message", int style = wxOK | wxCENTRE,  
wxWindow *parent = NULL, int x = -1, int y = -1)
```

General purpose message dialog. *style* may be a bit list of the following identifiers:

wxYES_NO	Puts Yes and No buttons on the message box. May be combined with wxCANCEL.
wxCANCEL	Puts a Cancel button on the message box. May be combined with wxYES_NO or wxOK.
wxOK	Puts an Ok button on the message box. May be combined with wxCANCEL.
wxCENTRE	Centres the text.
wxICON_EXCLAMATION	Under Windows, displays an exclamation mark symbol.
wxICON_HAND	Under Windows, displays a hand symbol.
wxICON_QUESTION	Under Windows, displays a question mark symbol.
wxICON_INFORMATION	Under Windows, displays an information symbol.

The return value is one of: wxYES, wxNO, wxCANCEL, wxOK.

For example:

```
...  
int answer = wxMessageBox("Quit program?", "Confirm",  
                           wxYES_NO | wxCANCEL, main_frame);  
if (answer == wxYES)  
    delete main_frame;  
...
```

message may contain newline characters, in which case the message will be split into separate lines, to cater for large messages.

Under Windows, the native MessageBox function is used unless wxCENTRE is specified in the style, in which case a generic function is used. This is because the native MessageBox function cannot centre text. The symbols are not shown when the generic function is used.

Include files

<wx/msgdlg.h>

5.4 GDI functions

The following are relevant to the GDI (Graphics Device Interface).

Include files

<wx/gdicmn.h>

5.4.1 ::wxColourDisplay

bool wxColourDisplay()

Returns TRUE if the display is colour, FALSE otherwise.

5.4.2 ::wxDisplayDepth

int wxDisplayDepth()

Returns the depth of the display (a value of 1 denotes a monochrome display).

5.4.3 ::wxMakeMetafilePlaceable

bool wxMakeMetafilePlaceable(const wxString& filename, int minX, int minY, int maxX, int maxY, float scale=1.0)

Given a filename for an existing, valid metafile (as constructed using *wxMetafileDC* (p. 442)) makes it into a placeable metafile by prepending a header containing the given bounding box. The bounding box may be obtained from a device context after drawing into it, using the functions *wxDC::MinX*, *wxDC::MinY*, *wxDC::MaxX* and *wxDC::MaxY*.

In addition to adding the placeable metafile header, this function adds the equivalent of the following code to the start of the metafile data:

```
SetMapMode(dc, MM_ANISOTROPIC);
SetWindowOrg(dc, minX, minY);
SetWindowExt(dc, maxX - minX, maxY - minY);
```

This simulates the *wxMM_TEXT* mapping mode, which *wxWindows* assumes.

Placeable metafiles may be imported by many Windows applications, and can be used in RTF (Rich Text Format) files.

scale allows the specification of scale for the metafile.

This function is only available under Windows.

5.4.4 ::wxSetCursor

void wxSetCursor(wxCursor *cursor)

Globally sets the cursor; only has an effect in MS Windows. See also *wxCursor* (p. 128), *wxWindow::SetCursor* (p. 834).

5.5 Printer settings

The following functions are used to control PostScript printing. Under Windows, PostScript output can only be sent to a file.

Include files

<wx/dcps.h>

5.5.1 ::wxGetPrinterCommand

wxString wxGetPrinterCommand()

Gets the printer command used to print a file. The default is `lpr`.

5.5.2 ::wxGetPrinterFile

wxString wxGetPrinterFile()

Gets the PostScript output filename.

5.5.3 ::wxGetPrinterMode

int wxGetPrinterMode()

Gets the printing mode controlling where output is sent (`PS_PREVIEW`, `PS_FILE` or `PS_PRINTER`). The default is `PS_PREVIEW`.

5.5.4 ::wxGetPrinterOptions

wxString wxGetPrinterOptions()

Gets the additional options for the print command (e.g. specific printer). The default is nothing.

5.5.5 ::wxGetPrinterOrientation

int wxGetPrinterOrientation()

Gets the orientation (PS_PORTRAIT or PS_LANDSCAPE). The default is PS_PORTRAIT.

5.5.6 ::wxGetPrinterPreviewCommand

wxString wxGetPrinterPreviewCommand()

Gets the command used to view a PostScript file. The default depends on the platform.

5.5.7 ::wxGetPrinterScaling

void wxGetPrinterScaling(float *x, float *y)

Gets the scaling factor for PostScript output. The default is 1.0, 1.0.

5.5.8 ::wxGetPrinterTranslation

void wxGetPrinterTranslation(float *x, float *y)

Gets the translation (from the top left corner) for PostScript output. The default is 0.0, 0.0.

5.5.9 ::wxSetPrinterCommand

void wxSetPrinterCommand(const wxString& *command*)

Sets the printer command used to print a file. The default is `lpr`.

5.5.10 ::wxSetPrinterFile

void wxSetPrinterFile(const wxString& *filename*)

Sets the PostScript output filename.

5.5.11 ::wxSetPrinterMode

void wxSetPrinterMode(int *mode*)

Sets the printing mode controlling where output is sent (PS_PREVIEW, PS_FILE or PS_PRINTER). The default is PS_PREVIEW.

5.5.12 ::wxSetPrinterOptions

void wxSetPrinterOptions(const wxString& options)

Sets the additional options for the print command (e.g. specific printer). The default is nothing.

5.5.13 ::wxSetPrinterOrientation**void wxSetPrinterOrientation(int orientation)**

Sets the orientation (PS_PORTRAIT or PS_LANDSCAPE). The default is PS_PORTRAIT.

5.5.14 ::wxSetPrinterPreviewCommand**void wxSetPrinterPreviewCommand(const wxString& command)**

Sets the command used to view a PostScript file. The default depends on the platform.

5.5.15 ::wxSetPrinterScaling**void wxSetPrinterScaling(float x, float y)**

Sets the scaling factor for PostScript output. The default is 1.0, 1.0.

5.5.16 ::wxSetPrinterTranslation**void wxSetPrinterTranslation(float x, float y)**

Sets the translation (from the top left corner) for PostScript output. The default is 0.0, 0.0.

5.6 Clipboard functions

These clipboard functions are implemented for Windows only.

Include files

<wx/clipbrd.h>

5.6.1 ::wxClipboardOpen**bool wxClipboardOpen()**

Returns TRUE if this application has already opened the clipboard.

5.6.2 `::wxCloseClipboard`

bool wxCloseClipboard()

Closes the clipboard to allow other applications to use it.

5.6.3 `::wxEmptyClipboard`

bool wxEmptyClipboard()

Empties the clipboard.

5.6.4 `::wxEnumClipboardFormats`

int wxEnumClipboardFormats(int *dataFormat*)

Enumerates the formats found in a list of available formats that belong to the clipboard. Each call to this function specifies a known available format; the function returns the format that appears next in the list.

dataFormat specifies a known format. If this parameter is zero, the function returns the first format in the list.

The return value specifies the next known clipboard data format if the function is successful. It is zero if the *dataFormat* parameter specifies the last format in the list of available formats, or if the clipboard is not open.

Before it enumerates the formats function, an application must open the clipboard by using the `wxOpenClipboard` function.

5.6.5 `::wxGetClipboardData`

wxObject * wxGetClipboardData(int *dataFormat*)

Gets data from the clipboard.

dataFormat may be one of:

- `wxCF_TEXT` or `wxCF_OEMTEXT`: returns a pointer to new memory containing a null-terminated text string.
- `wxCF_BITMAP`: returns a new `wxBitmap`.

The clipboard must have previously been opened for this call to succeed.

5.6.6 `::wxGetClipboardFormatName`

bool wxGetClipboardFormatName(int *dataFormat*, const wxString& *formatName*, int *maxCount*)

Gets the name of a registered clipboard format, and puts it into the buffer *formatName* which is of maximum length *maxCount*. *dataFormat* must not specify a predefined clipboard format.

5.6.7 `::wxIsClipboardFormatAvailable`

bool wxIsClipboardFormatAvailable(int *dataFormat*)

Returns TRUE if the given data format is available on the clipboard.

5.6.8 `::wxOpenClipboard`

bool wxOpenClipboard()

Opens the clipboard for passing data to it or getting data from it.

5.6.9 `::wxRegisterClipboardFormat`

int wxRegisterClipboardFormat(const wxString& *formatName*)

Registers the clipboard data format name and returns an identifier.

5.6.10 `::wxSetClipboardData`

bool wxSetClipboardData(int *dataFormat*, wxObject **data*, int *width*, int *height*)

Passes data to the clipboard.

dataFormat may be one of:

- `wxCF_TEXT` or `wxCF_OEMTEXT`: *data* is a null-terminated text string.
- `wxCF_BITMAP`: *data* is a `wxBitmap`.
- `wxCF_DIB`: *data* is a `wxBitmap`. The bitmap is converted to a DIB (device independent bitmap).
- `wxCF_METAFILE`: *data* is a `wxMetafile`. *width* and *height* are used to give recommended dimensions.

The clipboard must have previously been opened for this call to succeed.

5.7 Miscellaneous functions

5.7.1 `::wxNewId`

long wxNewId()

Generates an integer identifier unique to this run of the program.

Include files

<wx/utils.h>

5.7.2 `::wxRegisterId`

void wxRegisterId(long id)

Ensures that ids subsequently generated by **NewId** do not clash with the given **id**.

Include files

<wx/utils.h>

5.7.3 `::wxBeginBusyCursor`

void wxBeginBusyCursor(wxCursor *cursor = wxHOURLASS_CURSOR)

Changes the cursor to the given cursor for all windows in the application. Use *wxEndBusyCursor* (p. 865) to revert the cursor back to its previous state. These two calls can be nested, and a counter ensures that only the outer calls take effect.

See also *wxIsBusy* (p. 871), *wxBusyCursor* (p. 35).

Include files

<wx/utils.h>

5.7.4 `::wxBell`

void wxBell()

Ring the system bell.

Include files

<wx/utils.h>

5.7.5 `::wxCreateDynamicObject`

wxObject * wxCreateDynamicObject(const wxString& className)

Creates and returns an object of the given class, if the class has been registered with the dynamic class system using DECLARE... and IMPLEMENT... macros.

5.7.6 ::wxDDECleanUp

void wxDDECleanUp()

Called when wxWindows exits, to clean up the DDE system. This no longer needs to be called by the application.

See also `helprefwxDDEInitializewxddeinitialize`.

Include files

<wx/dde.h>

5.7.7 ::wxDDEInitialize

void wxDDEInitialize()

Initializes the DDE system. May be called multiple times without harm.

This no longer needs to be called by the application: it will be called by wxWindows if necessary.

See also `wxDDEServer` (p. 172), `wxDDEClient` (p. 166), `wxDDEConnection` (p. 167), `wxDDECleanUp` (p. 863).

Include files

<wx/dde.h>

5.7.8 ::wxDebugMsg

void wxDebugMsg(const wxString& fmt, ...)

Display a debugging message; under Windows, this will appear on the debugger command window, and under Unix, it will be written to standard error.

The syntax is identical to **printf**: pass a format string and a variable list of arguments.

Note that under Windows, you can see the debugging messages without a debugger if you have the DBWIN debug log application that comes with Microsoft C++.

Tip: under Windows, if your application crashes before the message appears in the debugging window, put a `wxYield` call after each `wxDebugMsg` call. `wxDebugMsg` seems to be broken under WIN32s (at least for Watcom C++): preformat your messages and use `OutputDebugString` instead.

This function is now obsolete, replaced by *Log functions* (p. 884).

Include files

<wx/utils.h>

5.7.9 ::wxDisplaySize

void wxDisplaySize(int *width, int *height)

Gets the physical size of the display in pixels.

Include files

<wx/gdicmn.h>

5.7.10 ::wxEntry

This initializes `wxWindows` in a platform-dependent way. Use this if you are not using the default `wxWindows` entry code (e.g. `main` or `WinMain`). For example, you can initialize `wxWindows` from an Microsoft Foundation Classes application using this function.

void wxEntry(HANDLE hInstance, HANDLE hPrevInstance, const wxString& commandLine, int cmdShow, bool enterLoop = TRUE)

`wxWindows` initialization under Windows (non-DLL). If `enterLoop` is `FALSE`, the function will return immediately after calling `wxApp::OnInit`. Otherwise, the `wxWindows` message loop will be entered.

void wxEntry(HANDLE hInstance, HANDLE hPrevInstance, WORD wDataSegment, WORD wHeapSize, const wxString& commandLine)

`wxWindows` initialization under Windows (for applications constructed as a DLL).

int wxEntry(int argc, const wxString& *argv)

`wxWindows` initialization under Unix.

Remarks

To clean up `wxWindows`, call `wxApp::OnExit` followed by the static function `wxApp::CleanUp`. For example, if exiting from an MFC application that also uses `wxWindows`:


```
int CTheApp::ExitInstance()  
{  
    // OnExit isn't called by CleanUp so must be called explicitly.  
    wxTheApp->OnExit();  
    wxApp::CleanUp();  
  
    return CWinApp::ExitInstance();  
}
```

Include files

<wx/app.h>

5.7.11 ::wxError

void wxError(const wxString& msg, const wxString& title = "wxWindows Internal Error")

Displays *msg* and continues. This writes to standard error under Unix, and pops up a message box under Windows. Used for internal wxWindows errors. See also *wxFatalError* (p. 866).

Include files

<wx/utils.h>

5.7.12 ::wxEndBusyCursor

void wxEndBusyCursor()

Changes the cursor back to the original cursor, for all windows in the application. Use with *wxBeginBusyCursor* (p. 862).

See also *wxIsBusy* (p. 871), *wxBusyCursor* (p. 35).

Include files

<wx/utils.h>

5.7.13 ::wxExecute

long wxExecute(const wxString& command, bool sync = FALSE, wxProcess *callback = NULL)

long wxExecute(char **argv, bool sync = FALSE, wxProcess *callback = NULL)

Executes another program in Unix or Windows.

The first form takes a command string, such as "emacs file.txt".

The second form takes an array of values: a command, any number of arguments, terminated by NULL.

If *sync* is FALSE (the default), flow of control immediately returns. If TRUE, the current application waits until the other program has terminated.

In the case of synchronous execution, the return value is the exit code of the process (which terminates by the moment the function returns) and will be -1 if the process couldn't be started and typically 0 if the process terminated successfully.

For asynchronous execution, however, the return value is the process id and zero value indicates that the command could not be executed.

If callback isn't NULL and if execution is asynchronous (note that callback parameter can not be non NULL for synchronous execution), *wxProcess::OnTerminate* (p. 527) will be called when the process finishes.

See also *wxShell* (p. 873), *wxProcess* (p. 525).

Include files

<wx/utils.h>

5.7.14 ::wxExit

void wxExit()

Exits application after calling *wxApp::OnExit* (p. 10). Should only be used in an emergency: normally the top-level frame should be deleted (after deleting all other frames) to terminate the application. See *wxWindow::OnCloseWindow* (p. 819) and *wxApp* (p. 6).

Include files

<wx/app.h>

5.7.15 ::wxFatalError

void wxFatalError(const wxString& msg, const wxString& title = "wxWindows Fatal Error")

Displays *msg* and exits. This writes to standard error under Unix, and pops up a message box under Windows. Used for fatal internal wxWindows errors. See also *wxError* (p. 865).

Include files

<wx/utils.h>

5.7.16 ::wxFindMenuItemId

```
int wxFindMenuItemId(wxFrame *frame, const wxString& menuString, const
wxString& itemString)
```

Find a menu item identifier associated with the given frame's menu bar.

Include files

```
<wx/utils.h>
```

5.7.17 ::wxFindWindowByLabel

```
wxWindow * wxFindWindowByLabel(const wxString& label, wxWindow
*parent=NULL)
```

Find a window by its label. Depending on the type of window, the label may be a window title or panel item label. If *parent* is NULL, the search will start from all top-level frames and dialog boxes; if non-NULL, the search will be limited to the given window hierarchy. The search is recursive in both cases.

Include files

```
<wx/utils.h>
```

5.7.18 ::wxFindWindowByName

```
wxWindow * wxFindWindowByName(const wxString& name, wxWindow
*parent=NULL)
```

Find a window by its name (as given in a window constructor or **Create** function call). If *parent* is NULL, the search will start from all top-level frames and dialog boxes; if non-NULL, the search will be limited to the given window hierarchy. The search is recursive in both cases.

If no such named window is found, **wxFindWindowByLabel** is called.

Include files

```
<wx/utils.h>
```

5.7.19 ::wxGetActiveWindow

```
wxWindow * wxGetActiveWindow()
```

Gets the currently active window (Windows only).

Include files

<wx/windows.h>

5.7.20 ::wxGetDisplayName

wxString wxGetDisplayName()

Under X only, returns the current display name. See also *wxSetDisplayName* (p. 872).

Include files

<wx/utils.h>

5.7.21 ::wxGetHomeDir

wxString wxGetHomeDir(const wxString& buf)

Fills the buffer with a string representing the user's home directory (Unix only).

Include files

<wx/utils.h>

5.7.22 ::wxGetHostName

bool wxGetHostName(const wxString& buf, int bufSize)

Copies the host name of the machine the program is running on into the buffer *buf*, of maximum size *bufSize*, returning TRUE if successful. Under Unix, this will return a machine name. Under Windows, this returns "windows".

Include files

<wx/utils.h>

5.7.23 ::wxGetElapsedTime

long wxGetElapsedTime(bool resetTimer = TRUE)

Gets the time in milliseconds since the last *::wxStartTimer* (p. 874).

If *resetTimer* is TRUE (the default), the timer is reset to zero by this call.

See also *wxTimer* (p. 745).

Include files

<wx/timer.h>

5.7.24 ::wxGetFreeMemory

long wxGetFreeMemory()

Returns the amount of free memory in Kbytes under environments which support it, and -1 if not supported. Currently, returns a positive value under Windows, and -1 under Unix.

Include files

<wx/utils.h>

5.7.25 ::wxGetMousePosition

void wxGetMousePosition(int* x, int* y)

Returns the mouse position in screen coordinates.

Include files

<wx/utils.h>

5.7.26 ::wxGetOsVersion

int wxGetOsVersion(int *major = NULL, int *minor = NULL)

Gets operating system version information.

Platform	Return types
Macintosh	Return value is wxMACINTOSH.
GTK	Return value is wxGTK, <i>major</i> is 1, <i>minor</i> is 0. (for GTK 1.0.X)
Motif	Return value is wxMOTIF_X, <i>major</i> is X version, <i>minor</i> is X revision.
OS/2	Return value is wxOS2_PM.
Windows 3.1	Return value is wxWINDOWS, <i>major</i> is 3, <i>minor</i> is 1.
Windows NT	Return value is wxWINDOWS_NT, <i>major</i> is 3, <i>minor</i> is 1.
Windows 95	Return value is wxWIN95, <i>major</i> is 3, <i>minor</i> is 1.
Win32s (Windows 3.1)	Return value is wxWIN32S, <i>major</i> is 3, <i>minor</i> is 1.
Watcom C++ 386 supervisor mode (Windows 3.1)	Return value is wxWIN386, <i>major</i> is 3, <i>minor</i> is 1.

Include files

<wx/utils.h>

5.7.27 ::wxGetResource

bool wxGetResource(const wxString& section, const wxString& entry, const wxString& *value, const wxString& file = NULL)

bool wxGetResource(const wxString& section, const wxString& entry, float *value, const wxString& file = NULL)

bool wxGetResource(const wxString& section, const wxString& entry, long *value, const wxString& file = NULL)

bool wxGetResource(const wxString& section, const wxString& entry, int *value, const wxString& file = NULL)

Gets a resource value from the resource database (for example, WIN.INI, or .Xdefaults). If *file* is NULL, WIN.INI or .Xdefaults is used, otherwise the specified file is used.

Under X, if an application class (wxApp::GetClassName) has been defined, it is appended to the string /usr/lib/X11/app-defaults/ to try to find an applications default file when merging all resource databases.

The reason for passing the result in an argument is that it can be convenient to define a default value, which gets overridden if the value exists in the resource file. It saves a separate test for that resource's existence, and it also allows the overloading of the function for different types.

See also *wxWriteResource* (p. 875), *wxConfigBase* (p. 111).

Include files

<wx/utils.h>

5.7.28 ::wxGetUserId

bool wxGetUserId(const wxString& buf, int bufSize)

Copies the user's login identity (such as "jacs") into the buffer *buf*, of maximum size *bufSize*, returning TRUE if successful. Under Windows, this returns "user".

Include files

<wx/utils.h>

5.7.29 ::wxGetUserName

bool wxGetUserName(const wxString& buf, int bufSize)

Copies the user's name (such as "Julian Smart") into the buffer *buf*, of maximum size *bufSize*, returning TRUE if successful. Under Windows, this returns "unknown".

Include files

<wx/utils.h>

5.7.30 ::wxKill

int wxKill(long pid, int sig)

Under Unix (the only supported platform), equivalent to the Unix kill function. Returns 0 on success, -1 on failure.

Tip: sending a signal of 0 to a process returns -1 if the process does not exist. It does not raise a signal in the receiving process.

Include files

<wx/utils.h>

5.7.31 ::wxIsBusy

bool wxIsBusy()

Returns TRUE if between two *wxBeginBusyCursor* (p. 862) and *wxEndBusyCursor* (p. 865) calls.

See also *wxBusyCursor* (p. 35).

Include files

<wx/utils.h>

5.7.32 ::wxLoadUserResource

wxString wxLoadUserResource(const wxString& resourceName, const wxString& resourceType="TEXT")

Loads a user-defined Windows resource as a string. If the resource is found, the function creates a new character array and copies the data into it. A pointer to this data is returned. If unsuccessful, NULL is returned.

The resource must be defined in the `.rc` file using the following syntax:

```
myResource TEXT file.ext
```

where `file.ext` is a file that the resource compiler can find.

One use of this is to store `.wxr` files instead of including the data in the C++ file; some compilers cannot cope with the long strings in a `.wxr` file. The resource data can then be parsed using `wxResourceParseString` (p. 884).

This function is available under Windows only.

Include files

```
<wx/utils.h>
```

5.7.33 ::wxNow

wxString wxNow()

Returns a string representing the current date and time.

Include files

```
<wx/utils.h>
```

5.7.34 ::wxPostDelete

void wxPostDelete(wxObject *object)

Tells the system to delete the specified object when all other events have been processed. In some environments, it is necessary to use this instead of deleting a frame directly with the delete operator, because some GUIs will still send events to a deleted window.

Now obsolete: use `wxWindow::Close` (p. 802) instead.

Include files

```
<wx/utils.h>
```

5.7.35 ::wxSetDisplayName

void wxSetDisplayName(const wxString& displayName)

Under X only, sets the current display name. This is the X host and display name such

as "colonsay:0.0", and the function indicates which display should be used for creating windows from this point on. Setting the display within an application allows multiple displays to be used.

See also *wxGetDisplayName* (p. 868).

Include files

<wx/utils.h>

5.7.36 ::wxShell

bool wxShell(const wxString& command = NULL)

Executes a command in an interactive shell window. If no command is specified, then just the shell is spawned.

See also *wxExecute* (p. 865).

Include files

<wx/utils.h>

5.7.37 ::wxSleep

void wxSleep(int secs)

Sleeps for the specified number of seconds.

Include files

<wx/utils.h>

5.7.38 ::wxStripMenuCodes

wxString wxStripMenuCodes(const wxString& in)

void wxStripMenuCodes(char* in, char* out)

Strips any menu codes from *in* and places the result in *out* (or returns the new string, in the first form).

Menu codes include & (mark the next character with an underline as a keyboard shortcut in Windows and Motif) and \t (tab in Windows).

Include files

<wx/utils.h>

5.7.39 ::wxStartTimer

void wxStartTimer()

Starts a stopwatch; use `::wxGetElapsedTime` (p. 868) to get the elapsed time.

See also *wxTimer* (p. 745).

Include files

<wx/timer.h>

5.7.40 ::wxToLower

char wxToLower(char *ch*)

Converts the character to lower case. This is implemented as a macro for efficiency.

Include files

<wx/utils.h>

5.7.41 ::wxToUpper

char wxToUpper(char *ch*)

Converts the character to upper case. This is implemented as a macro for efficiency.

Include files

<wx/utils.h>

5.7.42 ::wxTrace

void wxTrace(const wxString& *fmt*, ...)

Takes printf-style variable argument syntax. Output is directed to the current output stream (see *wxDebugContext* (p. 928)).

This function is now obsolete, replaced by *Log functions* (p. 884).

Include files

<wx/memory.h>

5.7.43 ::wxTraceLevel

void wxTraceLevel(int level, const wxString& fmt, ...)

Takes printf-style variable argument syntax. Output is directed to the current output stream (see *wxDebugContext* (p. 928)). The first argument should be the level at which this information is appropriate. It will only be output if the level returned by *wxDebugContext::GetLevel* is equal to or greater than this value.

This function is now obsolete, replaced by *Log functions* (p. 884).

Include files

<wx/memory.h>

5.7.44 ::wxWriteResource

bool wxWriteResource(const wxString& section, const wxString& entry, const wxString& value, const wxString& file = NULL)

bool wxWriteResource(const wxString& section, const wxString& entry, float value, const wxString& file = NULL)

bool wxWriteResource(const wxString& section, const wxString& entry, long value, const wxString& file = NULL)

bool wxWriteResource(const wxString& section, const wxString& entry, int value, const wxString& file = NULL)

Writes a resource value into the resource database (for example, WIN.INI, or .Xdefaults). If *file* is NULL, WIN.INI or .Xdefaults is used, otherwise the specified file is used.

Under X, the resource databases are cached until the internal function **wxFlushResources** is called automatically on exit, when all updated resource databases are written to their files.

Note that it is considered bad manners to write to the .Xdefaults file under Unix, although the WIN.INI file is fair game under Windows.

See also *wxGetResource* (p. 870), *wxConfigBase* (p. 111).

Include files

<wx/utils.h>

5.7.45 ::wxYield

bool wxYield()

Yields control to pending messages in the windowing system. This can be useful, for example, when a time-consuming process writes to a text window. Without an occasional yield, the text window will not be updated properly, and (since Windows multitasking is cooperative) other processes will not respond.

Caution should be exercised, however, since yielding may allow the user to perform actions which are not compatible with the current task. Disabling menu items or whole menus during processing can avoid unwanted reentrance of code.

Include files

<wx/utils.h>

5.8 Macros

These macros are defined in wxWindows.

5.8.1 CLASSINFO

wxClassInfo * CLASSINFO(className)

Returns a pointer to the wxClassInfo object associated with this class.

Include files

<wx/object.h>

5.8.2 WXDEBUG_NEW

WXDEBUG_NEW(arg)

This is defined in debug mode to be call the redefined new operator with filename and line number arguments. The definition is:

```
#define WXDEBUG_NEW new(__FILE__, __LINE__)
```

In non-debug mode, this is defined as the normal new operator.

Include files

<wx/object.h>

5.8.3 DECLARE_ABSTRACT_CLASS

DECLARE_ABSTRACT_CLASS(className)

Used inside a class declaration to declare that the class should be made known to the class hierarchy, but objects of this class cannot be created dynamically. The same as DECLARE_CLASS.

Example:

```
class wxCommand: public wxObject
{
    DECLARE_ABSTRACT_CLASS(wxCommand)

    private:
        ...
    public:
        ...
};
```

Include files

<wx/object.h>

5.8.4 DECLARE_APP**DECLARE_APP(className)**

This is used in headers to create a forward declaration of the wxGetApp function implemented by IMPLEMENT_APP. It creates the declaration `className& wxGetApp(void)`.

Example:

```
DECLARE_APP(MyApp)
```

Include files

<wx/app.h>

5.8.5 DECLARE_CLASS**DECLARE_CLASS(className)**

Used inside a class declaration to declare that the class should be made known to the class hierarchy, but objects of this class cannot be created dynamically. The same as DECLARE_ABSTRACT_CLASS.

Include files

<wx/object.h>

5.8.6 DECLARE_DYNAMIC_CLASS

DECLARE_DYNAMIC_CLASS(className)

Used inside a class declaration to declare that the objects of this class should be dynamically createable from run-time type information.

Example:

```
class wxFrame: public wxWindow
{
    DECLARE_DYNAMIC_CLASS(wxFrame)

    private:
        const wxString& frameTitle;
    public:
        ...
};
```

Include files

<wx/object.h>

5.8.7 IMPLEMENT_ABSTRACT_CLASS

IMPLEMENT_ABSTRACT_CLASS(className, baseClassName)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information. The same as IMPLEMENT_CLASS.

Example:

```
IMPLEMENT_ABSTRACT_CLASS(wxCommand, wxObject)

wxCommand::wxCommand(void)
{
    ...
}
```

Include files

<wx/object.h>

5.8.8 IMPLEMENT_ABSTRACT_CLASS2

IMPLEMENT_ABSTRACT_CLASS2(className, baseClassName1, baseClassName2)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information and two base classes. The same as IMPLEMENT_CLASS2.

Include files

<wx/object.h>

5.8.9 IMPLEMENT_APP

IMPLEMENT_APP(className)

This is used in the application class implementation file to make the application class known to wxWindows for dynamic construction. You use this instead of

Old form:

```
MyApp myApp;
```

New form:

```
IMPLEMENT_APP(MyApp)
```

See also *DECLARE_APP* (p. 877).

Include files

<wx/app.h>

5.8.10 IMPLEMENT_CLASS

IMPLEMENT_CLASS(className, baseClassName)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information. The same as **IMPLEMENT_ABSTRACT_CLASS**.

Include files

<wx/object.h>

5.8.11 IMPLEMENT_CLASS2

IMPLEMENT_CLASS2(className, baseClassName1, baseClassName2)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information and two base classes. The same as **IMPLEMENT_ABSTRACT_CLASS2**.

Include files

<wx/object.h>

5.8.12 IMPLEMENT_DYNAMIC_CLASS

IMPLEMENT_DYNAMIC_CLASS(className, baseClassName)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information, and whose instances can be created dynamically.

Example:

```
IMPLEMENT_DYNAMIC_CLASS(wxFrame, wxWindow)

wxFrame::wxFrame(void)
{
    ...
}
```

Include files

<wx/object.h>

5.8.13 IMPLEMENT_DYNAMIC_CLASS2

IMPLEMENT_DYNAMIC_CLASS2(className, baseClassName1, baseClassName2)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information, and whose instances can be created dynamically. Use this for classes derived from two base classes.

Include files

<wx/object.h>

5.8.14 WXTRACE

WXTRACE(formatString, ...)

Calls wxTrace with printf-style variable argument syntax. Output is directed to the current output stream (see *wxDebugContext* (p. 928)).

This macro is now obsolete, replaced by *Log functions* (p. 884).

Include files

<wx/memory.h>

5.8.15 WXTRACELEVEL

WXTRACELEVEL(level, formatString, ...)

Calls `wxTraceLevel` with printf-style variable argument syntax. Output is directed to the current output stream (see *wxDebugContext* (p. 928)). The first argument should be the level at which this information is appropriate. It will only be output if the level returned by `wxDebugContext::GetLevel` is equal to or greater than this value.

This function is now obsolete, replaced by *Log functions* (p. 884).

Include files

<wx/memory.h>

5.9 wxWindows resource functions

wxWindows resource system (p. 951)

This section details functions for manipulating wxWindows (.WXR) resource files and loading user interface elements from resources.

Please note that this use of the word 'resource' is different from that used when talking about initialisation file resource reading and writing, using such functions as `wxWriteResource` and `wxGetResource`. It's just an unfortunate clash of terminology.

For an overview of the wxWindows resource mechanism, see *the wxWindows resource system* (p. 951).

See also `wxWindow::LoadFromResource` (p. 815) for loading from resource data.

Warning: this needs updating for wxWindows 2.

5.9.1 ::wxResourceAddIdentifier

bool `wxResourceAddIdentifier`(const **wxString&** name, **int** value)

Used for associating a name with an integer identifier (equivalent to dynamically #defining a name to an integer). Unlikely to be used by an application except perhaps for implementing resource functionality for interpreted languages.

5.9.2 ::wxResourceClear

void `wxResourceClear`()

Clears the wxWindows resource table.

5.9.3 ::wxResourceCreateBitmap

wxBitmap * wxResourceCreateBitmap(const wxString& resource)

Creates a new bitmap from a file, static data, or Windows resource, given a valid wxWindows bitmap resource identifier. For example, if the .WXR file contains the following:

```
static const wxString& aiai_resource = "bitmap(name = 'aiai_resource',\
    bitmap = ['aiai', wxBITMAP_TYPE_BMP_RESOURCE, 'WINDOWS'],\
    bitmap = ['aiai.xpm', wxBITMAP_TYPE_XPM, 'X']).";
```

then this function can be called as follows:

```
wxBitmap *bitmap = wxResourceCreateBitmap("aiai_resource");
```

5.9.4 ::wxResourceCreateIcon**wxIcon * wxResourceCreateIcon(const wxString& resource)**

Creates a new icon from a file, static data, or Windows resource, given a valid wxWindows icon resource identifier. For example, if the .WXR file contains the following:

```
static const wxString& aiai_resource = "icon(name = 'aiai_resource',\
    icon = ['aiai', wxBITMAP_TYPE_ICO_RESOURCE, 'WINDOWS'],\
    icon = ['aiai', wxBITMAP_TYPE_XBM_DATA, 'X']).";
```

then this function can be called as follows:

```
wxIcon *icon = wxResourceCreateIcon("aiai_resource");
```

5.9.5 ::wxResourceCreateMenuBar**wxMenuBar * wxResourceCreateMenuBar(const wxString& resource)**

Creates a new menu bar given a valid wxWindows menubar resource identifier. For example, if the .WXR file contains the following:

```
static const wxString& menuBar11 = "menu(name = 'menuBar11',\
    menu = \
    [\
        ['&File', 1, '', \
            ['&Open File', 2, 'Open a file'],\
            ['&Save File', 3, 'Save a file'],\
            [],\
            ['E&xit', 4, 'Exit program']\
        ],\
        ['&Help', 5, '', \
            ['&About', 6, 'About this program']\
        ]\
    ]\
    ).";
```

then this function can be called as follows:

```
wxMenuBar *menuBar = wxResourceCreateMenuBar("menuBar11");
```

5.9.6 ::wxResourceGetIdentifier

int wxResourceGetIdentifier(const wxString& name)

Used for retrieving the integer value associated with an identifier. A zero value indicates that the identifier was not found.

See *wxResourceAddIdentifier* (p. 881).

5.9.7 ::wxResourceParseData

bool wxResourceParseData(const wxString& resource, wxResourceTable *table = NULL)

Parses a string containing one or more wxWindows resource objects. If the resource objects are global static data that are included into the C++ program, then this function must be called for each variable containing the resource data, to make it known to wxWindows.

resource should contain data in the following form:

```
dialog(name = 'dialog1',
       style = 'wxCAPTION | wxDEFAULT_DIALOG_STYLE',
       title = 'Test dialog box',
       x = 312, y = 234, width = 400, height = 300,
       modal = 0,
       control = [wxGroupBox, 'Groupbox', '0', 'group6', 5, 4, 380, 262,
                  [11, 'wxSWISS', 'wxNORMAL', 'wxNORMAL', 0]],
       control = [wxMultiText, 'Multitext', 'wxVERTICAL_LABEL', 'multitext3',
                  156, 126, 200, 70, 'wxWindows is a multi-platform, GUI toolkit.',
                  [11, 'wxSWISS', 'wxNORMAL', 'wxNORMAL', 0],
                  [11, 'wxSWISS', 'wxNORMAL', 'wxNORMAL', 0]]).
```

This function will typically be used after including a .wxr file into a C++ program as follows:

```
#include "dialog1.wxr"
```

Each of the contained resources will declare a new C++ variable, and each of these variables should be passed to *wxResourceParseData*.

5.9.8 ::wxResourceParseFile

bool wxResourceParseFile(const wxString& filename, wxResourceTable *table = NULL)

Parses a file containing one or more wxWindows resource objects in C++-compatible

syntax. Use this function to dynamically load wxWindows resource data.

5.9.9 ::wxResourceParseString

bool wxResourceParseString(const wxString& resource, wxResourceTable *table = NULL)

Parses a string containing one or more wxWindows resource objects. If the resource objects are global static data that are included into the C++ program, then this function must be called for each variable containing the resource data, to make it known to wxWindows.

resource should contain data with the following form:

```
static const wxString& dialog1 = "dialog(name = 'dialog1',\
    style = 'wxCAPTION | wxDEFAULT_DIALOG_STYLE',\
    title = 'Test dialog box',\
    x = 312, y = 234, width = 400, height = 300,\
    modal = 0,\
    control = [wxGroupBox, 'Groupbox', '0', 'group6', 5, 4, 380, 262,\
        [11, 'wxSWISS', 'wxNORMAL', 'wxNORMAL', 0]],\
    control = [wxMultiText, 'Multitext', 'wxVERTICAL_LABEL',\
        'multitext3',\
        156, 126, 200, 70, 'wxWindows is a multi-platform, GUI toolkit.',\
        [11, 'wxSWISS', 'wxNORMAL', 'wxNORMAL', 0]],\
        [11, 'wxSWISS', 'wxNORMAL', 'wxNORMAL', 0]])";
```

This function will typically be used after calling *wxLoadUserResource* (p. 871) to load an entire .wxr file into a string.

5.9.10 ::wxResourceRegisterBitmapData

bool wxResourceRegisterBitmapData(const wxString& name, const wxString& xbm_data, int width, int height, wxResourceTable *table = NULL)

bool wxResourceRegisterBitmapData(const wxString& name, const wxString& *xpm_data)

Makes #included XBM or XPM bitmap data known to the wxWindows resource system. This is required if other resources will use the bitmap data, since otherwise there is no connection between names used in resources, and the global bitmap data.

5.9.11 ::wxResourceRegisterIconData

Another name for *wxResourceRegisterBitmapData* (p. 884).

5.10 Log functions

These functions provide a variety of logging functions: see *Log classes overview* (p. 904)

for further information.

Include files

<wx/log.h>

5.10.1 ::wxLogError

void wxLogError(const char* *formatString*, ...)

The function to use for error messages, i.e. the messages that must be shown to the user. The default processing is to pop up a message box to inform the user about it.

5.10.2 ::wxLogFatalError

void wxLogFatalError(const char* *formatString*, ...)

Like *wxLogError* (p. 885), but also terminates the program with the exit code 3. Using *abort()* standard function also terminates the program with this exit code.

5.10.3 ::wxLogWarning

void wxLogWarning(const char* *formatString*, ...)

For warnings - they are also normally shown to the user, but don't interrupt the program work.

5.10.4 ::wxLogMessage

void wxLogMessage(const char* *formatString*, ...)

for all normal, informational messages. They also appear in a message box by default (but it can be changed). Notice that the standard behaviour is to not show informational messages if there are any errors later - the logic being that the later error messages make the informational messages preceding them meaningless.

5.10.5 ::wxLogVerbose

void wxLogVerbose(const char* *formatString*, ...)

For verbose output. Normally, it's suppressed, but might be activated if the user wishes to know more details about the program progress (another, but possibly confusing name for the same function is **wxLogInfo**).

5.10.6 ::wxLogStatus

void wxLogStatus(const char* *formatString*, ...)

For status messages - they will go into the status bar of the active or specified (as the first argument) *wxFrame* (p. 275) if it has one.

5.10.7 ::wxLogSysError

void wxLogSysError(const char* *formatString*, ...)

Mostly used by *wxWindows* itself, but might be handy for logging errors after system call (API function) failure. It logs the specified message text as well as the last system error code (*errno* or *::GetLastError()* depending on the platform) and the corresponding error message. The second form of this function takes the error code explicitly as the first argument.

5.10.8 ::wxLogDebug

void wxLogDebug(const char* *formatString*, ...)

The right function for debug output. It only does anything at all in the debug mode (when the preprocessor symbol `__WXDEBUG__` is defined) and expands to nothing in release mode (otherwise).

5.10.9 ::wxLogTrace

void wxLogTrace(const char* *formatString*, ...)

void wxLogTrace(wxTraceMask *mask*, const char* *formatString*, ...)

As **wxLogDebug**, only does something in debug build. The reason for making it a separate function from it is that usually there are a lot of trace messages, so it might make sense to separate them from other debug messages which would be flooded in them. Moreover, the second version of this function takes a trace mask as the first argument which allows to further restrict the amount of messages generated. The value of *mask* can be:

- `wxTraceMemAlloc`: trace memory allocation (new/delete)
- `wxTraceMessages`: trace window messages/X callbacks
- `wxTraceResAlloc`: trace GDI resource allocation
- `wxTraceRefCount`: trace various ref counting operations

5.11 Debugging macros and functions

Useful macros and functions for error checking and defensive programming. `ASSERTs` are only compiled if `__WXDEBUG__` is defined, whereas `CHECK` macros stay in release

builds.

Include files

<wx/debug.h>

5.11.1 ::wxOnAssert

void wxOnAssert(const char* fileName, int lineNumber, const char* msg = NULL)

This function may be redefined to do something non trivial and is called whenever one of debugging macros fails (i.e. condition is false in an assertion).

5.11.2 wxASSERT

wxASSERT(condition)

Assert macro. An error message will be generated if the condition is FALSE.

5.11.3 wxASSERT_MSG

wxASSERT_MSG(condition, msg)

Assert macro with message. An error message will be generated if the condition is FALSE.

5.11.4 wxFAIL

wxFAIL(condition)

Will always generate an assert error if this code is reached (in debug mode).

5.11.5 wxFAIL_MSG

wxFAIL_MSG(condition, msg)

Will always generate an assert error with specified message if this code is reached (in debug mode).

5.11.6 wxCHECK

wxCHECK(condition, retValue)

Checks that the condition is true, returns with the given return value if not (FAILs in debug mode). This check is done even in release mode.

5.11.7 wxCHECK_MSG

wxCHECK_MSG(*condition, retValue, msg*)

Checks that the condition is true, returns with the given return value if not (FAILs in debug mode). This check is done even in release mode.

5.12 Keycodes

Keypresses are represented by an enumerated type, `wxKeyCode`. The possible values are the ASCII character codes, plus the following:

```
WXX_BACK      = 8
WXX_TAB       = 9
WXX_RETURN    = 13
WXX_ESCAPE    = 27
WXX_SPACE     = 32
WXX_DELETE    = 127

WXX_START     = 300
WXX_LBUTTON
WXX_RBUTTON
WXX_CANCEL
WXX_MBUTTON
WXX_CLEAR
WXX_SHIFT
WXX_CONTROL
WXX_MENU
WXX_PAUSE
WXX_CAPITAL
WXX_PRIOR
WXX_NEXT
WXX_END
WXX_HOME
WXX_LEFT
WXX_UP
WXX_RIGHT
WXX_DOWN
WXX_SELECT
WXX_PRINT
WXX_EXECUTE
WXX_SNAPSHOT
WXX_INSERT
WXX_HELP
WXX_NUMPAD0
WXX_NUMPAD1
WXX_NUMPAD2
WXX_NUMPAD3
WXX_NUMPAD4
WXX_NUMPAD5
WXX_NUMPAD6
WXX_NUMPAD7
WXX_NUMPAD8
WXX_NUMPAD9
WXX_MULTIPLY
WXX_ADD
```


WXX_SEPARATOR
WXX_SUBTRACT
WXX_DECIMAL
WXX_DIVIDE
WXX_F1
WXX_F2
WXX_F3
WXX_F4
WXX_F5
WXX_F6
WXX_F7
WXX_F8
WXX_F9
WXX_F10
WXX_F11
WXX_F12
WXX_F13
WXX_F14
WXX_F15
WXX_F16
WXX_F17
WXX_F18
WXX_F19
WXX_F20
WXX_F21
WXX_F22
WXX_F23
WXX_F24
WXX_NUMLOCK
WXX_SCROLL

6 Classes by category

A classification of wxWindows classes by category.

Managed windows

There are several types of window that are directly controlled by the window manager (such as MS Windows, or the Motif Window Manager). Frames may contain windows, and dialog boxes may directly contain controls.

<i>wxDialog</i> (p. 178)	Dialog box
<i>wxFrame</i> (p. 275)	Normal frame
<i>wxMDIParentFrame</i> (p. 409)	MDI parent frame
<i>wxMDIChildFrame</i> (p. 404)	MDI child frame
<i>wxMiniFrame</i> (p. 446)	A frame with a small title bar
<i>wxTabbedDialog</i> (p. 682)	Tabbed dialog

See also **Common dialogs**.

Miscellaneous windows

The following are a variety of windows that are derived from *wxWindow*.

<i>wxGrid</i> (p. 297)	A grid (table) window
<i>wxPanel</i> (p. 488)	A window whose colour changes according to current user settings
<i>wxSashWindow</i> (p. 573)	Window with four optional sashes that can be dragged
<i>wxSashLayoutWindow</i> (p. 570)	Window that can be involved in an IDE-like layout arrangement
<i>wxScrolledWindow</i> (p. 586)	Window with automatically managed scrollbars
<i>wxSplitterWindow</i> (p. 626)	Window which can be split vertically or horizontally
<i>wxStatusBar</i> (p. 640)	Implements the status bar on a frame
<i>wxToolBar</i> (p. 746)	Toolbar class
<i>wxNotebook</i> (p. 464)	Notebook class

Common dialogs

Overview (p. 915)

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

<i>wxDialog</i> (p. 178)	Base class for common dialogs
<i>wxColourDialog</i> (p. 93)	Colour chooser dialog
<i>wxDirDialog</i> (p. 185)	Directory selector dialog
<i>wxFileDialog</i> (p. 248)	File selector dialog
<i>wxMultipleChoiceDialog</i> (p. 459)	Dialog to get one or more selections from a list
<i>wxSingleChoiceDialog</i> (p. 593)	Dialog to get a single selection from a list and

	return the string
<i>wxTextEntryDialog</i> (p. 725)	Dialog to get a single line of text from the user
<i>wxFontDialog</i> (p. 273)	Font chooser dialog
<i>wxPageSetupDialog</i> (p. 482)	Standard page setup dialog
<i>wxPrintDialog</i> (p. 512)	Standard print dialog
<i>wxMessageDialog</i> (p. 439)	Simple message box dialog

Controls

Typically, these are small windows which provide interaction with the user. Controls that are not static can have *validators* (p. 781) associated with them.

<i>wxControl</i> (p. 125)	The base class for controls
<i>wxButton</i> (p. 36)	Push button control, displaying text
<i>wxBitmapButton</i> (p. 54)	Push button control, displaying a bitmap
<i>wxCheckBox</i> (p. 70)	Checkbox control
<i>wxCheckListBox</i> (p. 72)	A listbox with a checkbox to the left of each item
<i>wxChoice</i> (p. 75)	Choice control (a combobox without the editable area)
<i>wxComboBox</i> (p. 94)	A choice with an editable area
<i>wxGauge</i> (p. 291)	A control to represent a varying quantity, such as time remaining
<i>wxStaticBox</i> (p. 637)	A static, or group box for visually grouping related controls
<i>wxListBox</i> (p. 373)	A list of strings for single or multiple selection
<i>wxListCtrl</i> (p. 381)	A control for displaying lists of strings and/or icons, plus a multicolumn report view
<i>wxTabCtrl</i> (p. 695)	Manages several tabs
<i>wxTextCtrl</i> (p. 712)	Single or multiline text editing control
<i>wxTreeCtrl</i> (p. 761)	Tree (hierachy) control
<i>wxScrollBar</i> (p. 579)	Scrollbar control
<i>wxSpinButton</i> (p. 622)	A spin or 'up-down' control
<i>wxStaticText</i> (p. 638)	One or more lines of non-editable text
<i>wxStaticBitmap</i> (p. 634)	A control to display a bitmap
<i>wxRadioBox</i> (p. 537)	A group of radio buttons
<i>wxRadioButton</i> (p. 543)	A round button to be used with others in a mutually exclusive way
<i>wxSlider</i> (p. 597)	A slider that can be dragged by the user

Menus

<i>wxMenu</i> (p. 418)	Displays a series of menu items for selection
<i>wxMenuBar</i> (p. 426)	Contains a series of menus for use with a frame
<i>wxMenuItem</i> (p. 433)	Represents a single menu item

Window layout

Overview (p. 919)

These are the classes relevant to automated window layout.

wxIndividualLayoutConstraint (p. 342) Represents a single constraint dimension
wxLayoutConstraints (p. 365) Represents the constraints for a window class

Device contexts

Overview (p. 926)

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

<i>wxClientDC</i> (p. 81)	A device context to access the client area outside OnPaint events
<i>wxPaintDC</i> (p. 483)	A device context to access the client area inside OnPaint events
<i>wxWindowDC</i> (p. 843)	A device context to access the non-client area
<i>wxScreenDC</i> (p. 578)	A device context to access the entire screen
<i>wxDC</i> (p. 151)	The device context base class
<i>wxMemoryDC</i> (p. 415)	A device context for drawing into bitmaps
<i>wxMetafileDC</i> (p. 442)	A device context for drawing into metafiles
<i>wxPostScriptDC</i> (p. 504)	A device context for drawing into PostScript files
<i>wxPrinterDC</i> (p. 515)	A device context for drawing to printers

Graphics device interface

Bitmaps overview (p. 908)

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

<i>wxColour</i> (p. 86)	Represents the red, blue and green elements of a colour
<i>wxBitmap</i> (p. 39)	Represents a bitmap
<i>wxBrush</i> (p. 61)	Used for filling areas on a device context
<i>wxBrushList</i> (p. 67)	The list of previously-created brushes
<i>wxCursor</i> (p. 128)	A small, transparent bitmap representing the cursor
<i>wxFont</i> (p. 264)	Represents fonts
<i>wxFontList</i> (p. 274)	The list of previously-created fonts
<i>wxIcon</i> (p. 318)	A small, transparent bitmap for assigning to frames and drawing on device contexts
<i>wxImage</i> (p. 325)	A platform-independent image class
<i>wxImageList</i> (p. 339)	A list of images, used with some controls
<i>wxMask</i> (p. 403)	Represents a mask to be used with a bitmap for transparent drawing
<i>wxPen</i> (p. 494)	Used for drawing lines on a device context
<i>wxPenList</i> (p. 501)	The list of previously-created pens
<i>wxPalette</i> (p. 484)	Represents a table of indices into RGB values
<i>wxRegion</i> (p. 562)	Represents a simple or complex region on a window or device context

Events

Overview (p. 939)

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

<i>wxActivateEvent</i> (p. 5)	A window or application activation event
<i>wxCalculateLayoutEvent</i> (p. 68)	Used to calculate window layout
<i>wxCloseEvent</i> (p. 84)	A close window or end session event
<i>wxCommandEvent</i> (p. 103)	An event from a variety of standard controls
<i>wxDropFilesEvent</i> (p. 214)	A drop files event
<i>wxEraseEvent</i> (p. 220)	An erase background event
<i>wxEvent</i> (p. 221)	The event base class
<i>wxFocusEvent</i> (p. 263)	A window focus event
<i>wxKeyEvent</i> (p. 359)	A keypress event
<i>wxIdleEvent</i> (p. 317)	An idle event
<i>wxInitDialogEvent</i> (p. 345)	A dialog initialisation event
<i>wxJoystickEvent</i> (p. 356)	A joystick event
<i>wxListEvent</i> (p. 394)	A list control event
<i>wxMenuEvent</i> (p. 438)	A menu event
<i>wxMouseEvent</i> (p. 450)	A mouse event
<i>wxMoveEvent</i> (p. 458)	A move event
<i>wxNotebookEvent</i> (p. 470)	A notebook control event
<i>wxPaintEvent</i> (p. 484)	A paint event
<i>wxProcessEvent</i> (p. 527)	A process ending event
<i>wxQueryLayoutInfoEvent</i> (p. 535)	Used to query layout information
<i>wxSizeEvent</i> (p. 596)	A size event
<i>wxSocketEvent</i> (p. 617)	A socket event
<i>wxSysColourChangedEvent</i> (p. 678)	A system colour change event
<i>wxTabEvent</i> (p. 700)	A tab control event
<i>wxTreeEvent</i> (p. 774)	A tree control event
<i>wxUpdateUIEvent</i> (p. 775)	A user interface update event

Validators

Overview (p. 969)

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

<i>wxValidator</i> (p. 781)	Base validator class
<i>wxTextValidator</i> (p. 728)	Text control validator class
<i>wxGenericValidator</i> (p. 295)	Generic control validator class

Data structures

These are the data structure classes supported by wxWindows.

<i>wxExpr</i> (p. 230)	A class for flexible I/O
<i>wxExprDatabase</i> (p. 237)	A class for flexible I/O
<i>wxDate</i> (p. 143)	A class for date manipulation
<i>wxHashTable</i> (p. 310)	A simple hash table implementation
<i>wxList</i> (p. 367)	A simple linked list implementation
<i>wxNode</i> (p. 463)	Represents a node in the wxList implementation
<i>wxObject</i> (p. 471)	The root class for most wxWindows classes
<i>wxPathList</i> (p. 492)	A class to help search multiple paths
<i>wxPoint</i> (p. 503)	Representation of a point
<i>wxRect</i> (p. 546)	A class representing a rectangle
<i>wxRegion</i> (p. 562)	A class representing a region
<i>wxString</i> (p. 654)	A string class
<i>wxStringList</i> (p. 674)	A class representing a list of strings
<i>wxRealPoint</i> (p. 546)	Representation of a point using floating point numbers
<i>wxSize</i> (p. 595)	Representation of a size
<i>wxTime</i> (p. 740)	A class for time manipulation
<i>wxVariant</i> (p. 783)	A class for storing arbitrary types that may change at run-time

Run-time class information system

Overview (p. 958)

wxWindows supports run-time manipulation of class information, and dynamic creation of objects given class names.

<i>wxClassInfo</i> (p. 79)	Holds run-time class information
<i>wxObject</i> (p. 471)	Root class for classes with run-time information
<i>Macros</i> (p. 876)	Macros for manipulating run-time information

Debugging features

Overview (p. 927)

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

<i>wxDebugContext</i> (p. 173)	Provides memory-checking facilities
<i>wxLog</i> (p. 399)	Logging facility
<i>Log functions</i> (p. 884)	Error and warning logging functions
<i>Debugging macros</i> (p. 886)	Debug macros for assertion and checking
<i>WXDEBUG_NEW</i> (p. 876)	Use this macro to give further debugging information

Interprocess communication

Overview (p. 946)

wxWindows provides a simple interprocess communications facilities based on DDE.

<i>wxDDEClient</i> (p. 166)	Represents a client
<i>wxDDEConnection</i> (p. 167)	Represents the connection between a client and a server
<i>wxDDEServer</i> (p. 172)	Represents a server
<i>wxTCPClient</i> (p. 703)	Represents a client
<i>wxTCPConnection</i> (p. 705)	Represents the connection between a client and a server
<i>wxTCPServer</i> (p. 709)	Represents a server
<i>wxSocketClient</i> (p. 615)	Represents a socket client
<i>wxSocketHandler</i> (p. 618)	Represents a socket handler
<i>wxSocketServer</i> (p. 620)	Represents a socket server

Document/view framework

Overview (p. 933)

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

<i>wxDocument</i> (p. 207)	Represents a document
<i>wxView</i> (p. 793)	Represents a view
<i>wxDocTemplate</i> (p. 202)	Manages the relationship between a document class and a view class
<i>wxDocManager</i> (p. 189)	Manages the documents and views in an application
<i>wxDocChildFrame</i> (p. 187)	A child frame for showing a document view
<i>wxDocParentFrame</i> (p. 200)	A parent frame to contain views

Printing framework

Overview (p. 950)

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

<i>wxPreviewFrame</i> (p. 507)	Frame for displaying a print preview
<i>wxPreviewCanvas</i> (p. 504)	Canvas for displaying a print preview
<i>wxPreviewControlBar</i> (p. 505)	Standard control bar for a print preview
<i>wxPrintData</i> (p. 508)	Represents information about the document being printed
<i>wxPrintDialog</i> (p. 512)	Standard print dialog
<i>wxPrinter</i> (p. 513)	Class representing the printer
<i>wxPrinterDC</i> (p. 515)	Printer device context
<i>wxPrintout</i> (p. 516)	Class representing a particular printout
<i>wxPrintPreview</i> (p. 519)	Class representing a print preview

Database classes

Database classes overview (p. 922)

wxWindows provides two alternative sets of classes for accessing Microsoft's ODBC (Open Database Connectivity) product. The new version by Remstar is documented in a separate manual. The older classes are as follows:

<i>wxDatabase</i> (p. 132)	Database class
<i>wxQueryCol</i> (p. 530)	Class representing a column
<i>wxQueryField</i> (p. 533)	Class representing a field
<i>wxRecordSet</i> (p. 550)	Class representing one or more record

Drag and drop and clipboard classes

Drag and drop and clipboard overview (p. 975)

<i>wxDataObject</i> (p. 138)	Data object class
<i>wxTextDataObject</i> (p. 723)	Text data object class
<i>wxFileDataObject</i> (p. 723)	File data object class
<i>wxBitmapDataObject</i> (p. 59)	Bitmap data object class
<i>wxPrivateDataObject</i> (p. 523)	Private data object class
<i>wxClipboard</i> (p. 82)	Clipboard class
<i>wxDropTarget</i> (p. 218)	Drop target class
<i>wxFileDropTarget</i> (p. 252)	File drop target class
<i>wxTextDropTarget</i> (p. 726)	Text drop target class
<i>wxDropSource</i> (p. 216)	Drop source class

File related classes

wxWindows has several small classes to work with disk files, see *file classes overview* (p. 977) for more details.

<i>wxFile</i> (p. 241)	Low-level file input/output
<i>wxTempFile</i> (p. 710)	Class to safely replace an existing file
<i>wxTextFile</i> (p. 730)	Class for working with text files as with arrays of lines

Stream classes

wxWindows has its own set of stream classes, as an alternative to often buggy standard stream libraries, and to provide enhanced functionality.

<i>wxStreamBase</i> (p. 646)	Stream base class
<i>wxStreamBuffer</i> (p. 648)	Stream buffer class
<i>wxInputStream</i> (p. 346)	Input stream class
<i>wxOutputStream</i> (p. 476)	Output stream class
<i>wxFilterInputStream</i> (p. 262)	Filtered input stream class
<i>wxFilterOutputStream</i> (p. 262)	Filtered output stream class

wxDatInputStream (p. 140) Platform-independent data input stream class
wxDatOutputStream (p. 141) Platform-independent data output stream class
wxFileInputStream (p. 256) File input stream class
wxFileOutputStream (p. 257) File output stream class
wxZlibInputStream (p. 844) Zlib (compression) input stream class
wxZlibOutputStream (p. 844) Zlib (compression) output stream class
wxSocketInputStream (p. 622) Socket input stream class
wxSocketOutputStream (p. 622) Socket output stream class

Miscellaneous

wxAcceleratorTable (p. 2) Accelerator table
wxApp (p. 6) Application class
wxAutomationObject (p. 30) OLE automation class
wxConfig (p. 111) Classes for configuration reading/writing
wxHelpController (p. 313) Family of classes for controlling help windows
wxLayoutAlgorithm (p. 362) An alternative window layout facility
wxProcess (p. 525) Process class
wxTimer (p. 745) Timer class
wxSystemSettings (p. 678) System settings class

7 Topic overviews

This chapter contains a selection of topic overviews.

7.1 wxApp overview

Classes: *wxApp* (p. 6)

A wxWindows application does not have a *main* procedure; the equivalent is the *OnInit* (p. 12) member defined for a class derived from *wxApp*. *OnInit* will usually create a top window as a bare minimum.

Unlike in earlier versions of wxWindows, *OnInit* does not return a frame. Instead it returns a boolean value which indicates whether processing should continue (TRUE) or not (FALSE). You call *wxApp::SetTopWindow* (p. 15) to let wxWindows know about the top window.

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

An application closes by destroying all windows. Because all frames must be destroyed for the application to exit, it is advisable to use parent frames wherever possible when creating new frames, so that deleting the top level frame will automatically delete child frames. The alternative is to explicitly delete child frames in the top-level frame's *wxWindow::OnCloseWindow* (p. 819) handler.

In emergencies the *wxExit* (p. 866) function can be called to kill the application.

An example of defining an application follows:

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

IMPLEMENT_APP(DerivedApp)

bool DerivedApp::OnInit()
{
    wxFrame *the_frame = new wxFrame(NULL, argv[0]);
    ...
    SetTopWindow(the_frame);

    return TRUE;
}
```

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

You can also use `DECLARE_APP(appClass)` in a header file to declare the `wxGetApp` function which returns a reference to the application object.

7.2 wxString overview

Classes: *wxString* (p. 654), *wxArrayString* (p. 25), *wxStringTokenizer* (p. 676)

7.2.1 Introduction

wxString is a class which represents a character string of arbitrary length (limited by `MAX_INT` which is usually 2147483647 on 32 bit machines) and containing arbitrary characters. The ASCII NUL character is allowed, although care should be taken when passing strings containing it to other functions.

wxString only works with ASCII (8 bit characters) strings as of this release, but support for UNICODE (16 bit characters) is planned for the next one.

This class has all the standard operations you can expect to find in a string class: dynamic memory management (string extends to accommodate new characters), construction from other strings, C strings and characters, assignment operators, access to individual characters, string concatenation and comparison, substring extraction, case conversion, trimming and padding (with spaces), searching and replacing and both C-like *Printf()* (p. 669) and stream-like insertion functions as well as much more - see *wxString* (p. 654) for a list of all functions.

7.2.2 Comparison of wxString to other string classes

The advantages of using a special string class instead of working directly with C strings are so obvious that there is a huge number of such classes available. The most important advantage is the need to always remember to allocate/free memory for C strings; working with fixed size buffers almost inevitably leads to buffer overflows. At last, C++ has a standard string class (`std::string`). So why the need for *wxString*?

There are several advantages:

1. **Efficiency** This class was made to be as efficient as possible: both in terms of size (each *wxString* object takes exactly the same space as a *char ** pointer, see *reference counting* (p. 901)) and speed. It also provides performance *statistics gathering code* (p. 902) which may be enabled to fine tune the memory allocation strategy for your particular application - and the gain might be quite big.
2. **Compatibility** This class tries to combine almost full compatibility with the old *wxWindows 1.xx wxString* class, some reminiscence to MFC *CString* class and 90% of the functionality of `std::string` class.
3. **Rich set of functions** Some of the functions present in *wxString* are very useful but don't exist in most of other string classes: for example, *AfterFirst* (p. 662), *BeforeLast* (p. 663), *operator<<* (p. 673) or *Printf* (p. 669). Of course, all the standard string operations are supported as well.

4. **UNICODE** In this release, `wxString` only supports *construction* from a UNICODE string, but in the next one it will be capable of also storing its internal data in either ASCII or UNICODE format.
5. **Used by wxWindows** And, of course, this class is used everywhere inside `wxWindows` so there is no performance loss which would result from conversions of objects of any other string class (including `std::string`) to `wxString` internally by `wxWindows`.

However, there are several problems as well. The most important one is probably that there are often several functions to do exactly the same thing: for example, to get the length of the string either one of *length()* (p. 668), *Len()* (p. 668) or *Length()* (p. 668) may be used. The first function, as almost all the other functions in lowercase, is `std::string` compatible. The second one is "native" `wxString` version and the last one is `wxWindows` 1.xx way. So the question is: which one is better to use? And the answer is that:

The usage of `std::string` compatible functions is strongly advised! It will both make your code more familiar to other C++ programmers (who are supposed to have knowledge of `std::string` but not of `wxString`), let you reuse the same code in both `wxWindows` and other programs (by just typedefing `wxString` as `std::string` when used outside `wxWindows`) and by staying compatible with future versions of `wxWindows` which will probably start using `std::string` sooner or later too.

In the situations where there is no corresponding `std::string` function, please try to use the new `wxString` methods and not the old `wxWindows` 1.xx variants which are deprecated and may disappear in future versions.

7.2.3 Some advice about using `wxString`

Probably the main trap with using this class is the implicit conversion operator to *const char **. It is advised that you use *c_str()* (p. 663) instead to clearly indicate when the conversion is done. Specifically, the danger of this implicit conversion may be seen in the following code fragment:

```
// this function converts the input string to uppercase, output it to
// the screen
// and returns the result
const char *SayHELLO(const wxString& input)
{
    wxString output = input.Upper();

    printf("Hello, %s!\n", output);

    return output;
}
```

There are two nasty bugs in these three lines. First of them is in the call to the *printf()* function. Although the implicit conversion to C strings is applied automatically by the compiler in the case of

```
puts(output);
```

because the argument of *puts()* is known to be of the type *const char **, this is **not** done

for *printf()* which is a function with variable number of arguments (and whose arguments are of unknown types). So this call may do anything at all (including displaying the correct string on screen), although the most likely result is a program crash. The solution is to use *c_str()* (p. 663): just replace this line with

```
printf("Hello, %s!\n", output.c_str());
```

The second bug is that returning *output* doesn't work. The implicit cast is used again, so the code compiles, but as it returns a pointer to a buffer belonging to a local variable which is deleted as soon as the function exits, its contents is totally arbitrary. The solution to this problem is also easy: just make the function return *wxString* instead of a C string.

This leads us to the following general advice: all functions taking string arguments should take *const wxString&* (this makes assignment to the strings inside the function faster because of *reference counting* (p. 901)) and all functions returning strings should return *wxString* - this makes it safe to return local variables.

7.2.4 Other string related functions and classes

As most programs use character strings, the standard C library provides quite a few functions to work with them. Unfortunately, some of them have rather counter-intuitive behaviour (like *strncpy()* which doesn't always terminate the resulting string with a NULL) and are in general not very safe (passing NULL to them will probably lead to program crash). Moreover, some very useful functions are not standard at all. This is why in addition to all *wxString* functions, there are also a few global string functions which try to correct these problems: *IsEmpty()* (p. 851) verifies whether the string is empty (returning TRUE for NULL pointers), *Strlen()* (p. 851) also handles NULLs correctly and returns 0 for them and *Stricmp()* (p. 851) is just a platform-independent version of case-insensitive string comparison function known either as *stricmp()* or *strcasecmp()* on different platforms.

There is another class which might be useful when working with *wxString*: *wxStringTokenizer* (p. 676). It is helpful when a string must be broken into tokens and replaces the standard C library *strtok()* function.

And the very last string-related class is *wxArrayString* (p. 25): it is just a version of the "template" dynamic array class which is specialized to work with strings. Please note that this class is specially optimized (using its knowledge of the internal structure of *wxString*) for storing strings and so it is vastly better from a performance point of view than a *wxObjectArray* of *wxStrings*.

7.2.5 Reference counting and why you shouldn't care about it

wxString objects use a technique known as *copy on write* (COW). This means that when a string is assigned to another, no copying really takes place: only the reference count on the shared string data is incremented and both strings share the same data.

But as soon as one of the two (or more) strings is modified, the data has to be copied

because the changes to one of the strings shouldn't be seen in the others. As data copying only happens when the string is written to, this is known as COW.

What is important to understand is that all this happens absolutely transparently to the class users and that whether a string is shared or not is not seen from the outside of the class - in any case, the result of any operation on it is the same.

Probably the unique case when you might want to think about reference counting is when a string character is taken from a string which is not a constant (or a constant reference). In this case, due to C++ rules, the "read-only" *operator[]* (which is the same as *GetChar()* (p. 665)) cannot be chosen and the "read/write" *operator[]* (the same as *GetWritableChar()* (p. 665)) is used instead. As the call to this operator may modify the string, its data is unshared (COW is done) and so if the string was really shared there is some performance loss (both in terms of speed and memory consumption). In the rare cases when this may be important, you might prefer using *GetChar()* (p. 665) instead of the array subscript operator for this reasons. Please note that *at()* (p. 658) method has the same problem as the subscript operator in this situation and so using it is not really better. Also note that if all string arguments to your functions are passed as *const wxString&* (see the section *Some advice* (p. 900)) this situation will almost never arise because for constant references the correct operator is called automatically.

7.2.6 Tuning wxString for your application

Note: this section is strictly about performance issues and is absolutely not necessary to read for using wxString class. Please skip it unless you feel familiar with profilers and relative tools. If you do read it, please also read the preceding section about *reference counting* (p. 901).

For the performance reasons wxString doesn't allocate exactly the amount of memory needed for each string. Instead, it adds a small amount of space to each allocated block which allows it to not reallocate memory (a relatively expensive operation) too often as when, for example, a string is constructed by subsequently adding one character at a time to it, as for example in:

```
// delete all vowels from the string
wxString DeleteAllVowels(const wxString& original)
{
    wxString result;

    size_t len = original.length();
    for ( size_t n = 0; n < len; n++ )
    {
        if ( strchr("aeuio", tolower(original[n])) == NULL )
            result += original[n];
    }

    return result;
}
```

This is quite a common situation and not allocating extra memory at all would lead to very bad performance in this case because there would be as many memory (re)allocations as there are consonants in the original string. Allocating too much extra

memory would help to improve the speed in this situation, but due to a great number of `wxString` objects typically used in a program would also increase the memory consumption too much.

The very best solution in precisely this case would be to use `Alloc()` (p. 662) function to preallocate, for example, `len` bytes from the beginning - this will lead to exactly one memory allocation being performed (because the result is at most as long as the original string).

However, using `Alloc()` is tedious and so `wxString` tries to do its best. The default algorithm assumes that memory allocation is done in granularity of at least 16 bytes (which is the case on almost all of wide-spread platforms) and so nothing is lost if the amount of memory to allocate is rounded up to the next multiple of 16. Like this, no memory is lost and 15 iterations from 16 in the example above won't allocate memory but use the already allocated pool.

The default approach is quite conservative. Allocating more memory may bring important performance benefits for programs using (relatively) few very long strings. The amount of memory allocated is configured by the setting of `EXTRA_ALLOC` in the file `string.cpp` during compilation (be sure to understand why its default value is what it is before modifying it!). You may try setting it to greater amount (say twice `nLen`) or to 0 (to see performance degradation which will follow) and analyse the impact of it on your program. If you do it, you will probably find it helpful to also define `WXSTRING_STATISTICS` symbol which tells the `wxString` class to collect performance statistics and to show them on `stderr` on program termination. This will show you the average length of strings your program manipulates, their average initial length and also the percent of times when memory wasn't reallocated when string concatenation was done but the already preallocated memory was used (this value should be about 98% for the default allocation policy, if it is less than 90% you should really consider fine tuning `wxString` for your application).

It goes without saying that a profiler should be used to measure the precise difference the change to `EXTRA_ALLOC` makes to your program.

7.3 Container classes overview

Classes: `wxList` (p. 367), `wxArray` (p. 15)

`wxWindows` uses itself several container classes including doubly-linked lists and dynamic arrays (i.e. arrays which expand automatically when they become full). For both historical and portability reasons `wxWindows` does not use STL which provides the standard implementation of many container classes in C++. First of all, `wxWindows` has existed since well before STL was written, and secondly we don't believe that today compilers can deal really well with all of STL classes (this is especially true for some less common platforms). Of course, the compilers are evolving quite rapidly and hopefully their progress will allow to base future versions of `wxWindows` on STL - but this is not yet the case.

`wxWindows` container classes don't pretend to be as powerful or full as STL ones, but they are quite useful and may be compiled with absolutely any C++ compiler. They're used internally by `wxWindows`, but may, of course, be used in your programs as well if

you wish.

The list classes in wxWindows are doubly-linked lists which may either own the objects they contain (meaning that the list deletes the object when it is removed from the list or the list itself is destroyed) or just store the pointers depending on whether you called or not `wxList::DeleteContents` (p. 370) method.

Dynamic arrays resemble C arrays but with two important differences: they provide run-time range checking in debug builds and they expand automatically the allocated memory when there is no more space for new items. They come in two sorts: the "plain" arrays which store either built-in types such as "char", "int" or "bool" or the pointers to arbitrary objects, or "object arrays" which own the object pointers to which they store.

For the same portability reasons, the container classes implementation in wxWindows does not use templates, but is rather based on C preprocessor i.e. is done with the macros: `WX_DECLARE_LIST` and `WX_DEFINE_LIST` for the linked lists and `WX_DECLARE_ARRAY`, `WX_DECLARE_OBJARRAY` and `WX_DEFINE_OBJARRAY` for the dynamic arrays. The "DECLARE" macro declares a new container class containing the elements of given type and is needed for all three types of container classes: lists, arrays and objarrays. The "DEFINE" classes must be inserted in your program in a place where the **full declaration of container element class is in scope** (i.e. not just forward declaration), otherwise destructors of the container elements will not be called! As array classes never delete the items they contain anyhow, there is no `WX_DEFINE_ARRAY` macro for them.

Examples of usage of these macros may be found in `wxList` (p. 367) and `wxArray` (p. 15) documentation.

Finally, wxWindows predefines several commonly used container classes. `wxList` is defined for compatibility with previous versions as a list containing `wxObjects` and `wxStringList` as a list of C-style strings (`char *`), both of these classes are deprecated and should not be used in new programs. The following array classes are defined: `wxArrayInt`, `wxArrayLong`, `wxArrayPtrVoid` and `wxArrayString`. The first three store elements of corresponding types, but `wxArrayString` is somewhat special: it is an optimized version of `wxArray` which uses its knowledge about `wxString` (p. 654) reference counting schema.

7.4 Log classes overview

Classes: `wxLog` (p. 399), `wxLogStderr`, `wxLogOstream`, `wxLogTextCtrl`, `wxLogWindow`, `wxLogGui`, `wxLogNull`

This is a general overview of logging classes provided by wxWindows. The word logging here has a broad sense, including all of the program output, not only non interactive messages. The logging facilities included in wxWindows provide the base `wxLog` class which defines the standard interface for a *log target* as well as several standard implementations of it and a family of functions to use with them.

First of all, no knowledge of `wxLog` classes is needed to use them. For this, you should only know about `wxLogXXX()` functions. All of them have the same syntax as `printf()`, i.e. they take the format string as the first argument and a variable number of arguments.

Here are all of them:

- **wxLogFatalError** which is like *wxLogError*, but also terminates the program with the exit code 3 (using *abort()* standard function also terminates the program with this exit code).
- **wxLogError** is the function to use for error messages, i.e. the messages that must be shown to the user. The default processing is to pop up a message box to inform the user about it.
- **wxLogWarning** for warnings - they are also normally shown to the user, but don't interrupt the program work.
- **wxLogMessage** is for all normal, informational messages. They also appear in a message box by default (but it can be changed, see below). Notice that the standard behaviour is to not show informational messages if there are any errors later - the logic being that the later error messages make the informational messages preceding them meaningless.
- **wxLogVerbose** is for verbose output. Normally, it's suppressed, but might be activated if the user wishes to know more details about the program progress (another, but possibly confusing name for the same function is **wxLogInfo**).
- **wxLogStatus** is for status messages - they will go into the status bar of the active or specified (as the first argument) *wxFrame* (p. 275) if it has one.
- **wxLogSysError** is mostly used by *wxWindows* itself, but might be handy for logging errors after system call (API function) failure. It logs the specified message text as well as the last system error code (*errno* or *::GetLastError()* depending on the platform) and the corresponding error message. The second form of this function takes the error code explicitly as the first argument.
- **wxLogDebug** is the right function for debug output. It only does anything at all in the debug mode (when the preprocessor symbol `__WXDEBUG__` is defined) and expands to nothing in release mode (otherwise).
- **wxLogTrace** as **wxLogDebug** only does something in debug build. The reason for making it a separate function from it is that usually there are a lot of trace messages, so it might make sense to separate them from other debug messages which would be flooded in them. Moreover, the second version of this function takes a trace mask as the first argument which allows to further restrict the amount of messages generated.

The usage of these functions should be fairly straightforward, however it may be asked why not use the other logging facilities, such as C standard `stdio` functions or C++ streams. The short answer is that they're all very good generic mechanisms, but are not really adapted for *wxWindows*, while the log classes are. Some of advantages in using *wxWindows* log functions are:

- **Portability** It's a common practice to use *printf()* statements or `cout/cerr` C++ streams for writing out some (debug or otherwise) information. Although it works just fine under Unix, these messages go strictly nowhere under Windows where the `stdout` of GUI programs is not assigned to anything. Thus, you might view *wxLogMessage()* as a simple substitute for *printf()*.
- **Flexibility** The output of *wxLog* functions can be redirected or suppressed entirely based on their importance, which is either impossible or difficult to do with traditional methods. For example, only error messages, or only error messages and warnings might be logged, filtering out all informational

messages.

- **Completeness** Usually, an error message should be presented to the user when some operation fails. Let's take a quite simple but common case of a file error: suppose that you're writing your data file on disk and there is not enough space. The actual error might have been detected inside `wxWindows` code (say, in `wxFile::Write`), so the calling function doesn't really know the exact reason of the failure, it only knows that the data file couldn't be written to the disk. However, as `wxWindows` uses `wxLogError()` in this situation, the exact error code (and the corresponding error message) will be given to the user together with "high level" message about data file writing error.

After having enumerated all the functions which are normally used to log the messages, and why would you want to use them we now describe how all this works.

`wxWindows` has the notion of a *log target*: it's just a class deriving from `wxLog` (p. 399). As such, it implements the virtual functions of the base class which are called when a message is logged. Only one log target is *active* at any moment, this is the one used by `wxLogXXX()` functions. The normal usage of a log object (i.e. object of a class derived from `wxLog`) is to install it as the active target with a call to `SetActiveTarget()` and it will be used automatically by all subsequent calls to `wxLogXXX()` functions.

To create a new log target class you only need to derive it from `wxLog` and implement one (or both) of `DoLog()` and `DoLogString()` in it. The second one is enough if you're happy with the standard `wxLog` message formatting (prepending "Error:" or "Warning:", timestamping &c) but just want to send the messages somewhere else. The first one may be overridden to do whatever you want but you have to distinguish between the different message types yourself.

There are some predefined classes deriving from `wxLog` and which might be helpful to see how you can create a new log target class and, of course, may also be used without any change. There are:

- **wxLogStderr** This class logs messages to a *FILE **, using `stderr` by default as its name suggests.
- **wxLogStream** This class has the same functionality as `wxLogStderr`, but uses *ostream* and *cerr* instead of *FILE ** and `stderr`.
- **wxLogGui** This is the standard log target for `wxWindows` applications (it's used by default if you don't do anything) and provides the most reasonable handling of all types of messages for given platform.
- **wxLogWindow** This log target provides a "log console" which collects all messages generated by the application and also passes them to the previous active log target. The log window frame has a menu allowing user to clear the log, close it completely or save all messages to file.
- **wxLogNull** The last log class is quite particular: it doesn't do anything. The objects of this class may be instantiated to (temporarily) suppress output of `wxLogXXX()` functions. As an example, trying to open a non-existing file will usually provoke an error message, but if for some reasons it's unwanted, just use this construction:

```
wxFile file;
```

```
// wxFile.Open() normally complains if file can't be opened, we
don't want it
{
    wxLogNull logNo;
    if ( !file.Open("bar") )
        ... process error ourselves ...
} // ~wxLogNull called, old log sink restored

wxLogMessage("..."); // ok
```

7.5 Config classes overview

Classes: *wxConfig* (p. 111)

This overview briefly describes what the config classes are and what they are for. All the details about how to use them may be found in the description of the *wxConfigBase* (p. 111) class and the documentation of the file, registry and INI file based implementations mentions all the features/limitations specific to each one of these versions.

The config classes provide a way to store some application configuration information. They were especially designed for this usage and, although may probably be used for many other things as well, should be limited to it. It means that this information should be:

1. Typed, i.e. strings or numbers for the moment. You can not store binary data, for example.
2. Small. For instance, it is not recommended to use the Windows registry for amounts of data more than a couple of kilobytes.
3. Not performance critical, neither from speed nor from a memory consumption point of view.

On the other hand, the features provided make them very useful for storing all kinds of small to medium volumes of hierarchically-organized, heterogenous data. In short, this is a place where you can conveniently stuff all your data (numbers and strings) organizing it in a tree where you use the filesystem-like paths to specify the location of a piece of data. In particular, these classes were designed to be as easy to use as possible.

From another point of view, they provide an interface which hides the differences between the Windows registry and the standard Unix text format configuration files. Other (future) implementations of *wxConfigBase* might also understand GTK resource files or their analogues on the KDE side.

In any case, each implementation of *wxConfigBase* does its best to make the data look the same way everywhere. Due to the limitations of the underlying physical storage as in the case of *wxIniConfig*, it may not implement 100% of the base class functionality.

There are groups of entries and the entries themselves. Each entry contains either a

string or a number (or a boolean value; support for other types of data such as dates or timestamps is planned) and is identified by the full path to it: something like `/MyApp/UserPreferences/Colors/Foreground`. The previous elements in the path are the group names, and each name may contain an arbitrary number of entries and subgroups. The path components are **always** separated with a slash, even though some implementations use the backslash internally. Further details (including how to read/write these entries) may be found in the documentation for *wxConfigBase* (p. 111).

7.6 Bitmaps and icons overview

Classes: *wxBitmap* (p. 39), *wxBitmapHandler* (p. 50), *wxIcon* (p. 318), *wxCursor* (p. 128).

The *wxBitmap* class encapsulates the concept of a platform-dependent bitmap, either monochrome or colour. Platform-specific methods for creating a *wxBitmap* object from an existing file are catered for, and this is an occasion where conditional compilation will sometimes be required.

A bitmap created dynamically or loaded from a file can be selected into a memory device context (instance of *wxMemoryDC* (p. 415)). This enables the bitmap to be copied to a window or memory device context using *wxDC::Blit* (p. 152), or to be used as a drawing surface. The **wxToolBarSimple** class is implemented using bitmaps, and the toolbar demo shows one of the toolbar bitmaps being used for drawing a miniature version of the graphic which appears on the main window.

See *wxMemoryDC* (p. 415) for an example of drawing onto a bitmap.

The following shows the conditional compilation required to load a bitmap under Unix and in Windows. The alternative is to use the string version of the bitmap constructor, which loads a file under Unix and a resource or file under Windows, but has the disadvantage of requiring the XPM icon file to be available at run-time.

```
#if defined(__WXGTK__) || defined(__WXMOTIF__)
#include "mondrian.xpm"
#endif
```

A macro, *wxICON*, is available which creates an icon using an XPM on the appropriate platform, or an icon resource on Windows.

```
wxIcon icon(wxICON(mondrian));

// Equivalent to:

#if defined(__WXGTK__) || defined(__WXMOTIF__)
wxIcon icon(mondrian_xpm);
#endif

#if defined(__WMSW__)
wxIcon icon("mondrian");
#endif
```

7.6.1 Supported bitmap file formats

The following lists the formats handled on different platforms. Note that missing or partially-implemented formats can be supplemented by using *wxImage* (p. 325) to load the data, and then converting it to *wxBitmap* form.

wxBitmap

Under Windows, *wxBitmap* may load the following formats:

- Windows bitmap resource (`wxBITMAP_TYPE_BMP_RESOURCE`)
- Windows bitmap file (`wxBITMAP_TYPE_BMP`)
- PNG file (`wxBITMAP_TYPE_PNG`). Currently 4-bit (16-colour) PNG files do not load properly.
- XPM data and file (`wxBITMAP_TYPE_XPM`)

Under *wxGTK*, *wxBitmap* may load the following formats:

- Windows bitmap file (`wxBITMAP_TYPE_BMP`)
- PNG (`wxBITMAP_TYPE_PNG`).
- XPM data and file (`wxBITMAP_TYPE_XPM`)

Under *wxMotif*, *wxBitmap* may load the following formats:

- XBM data and file (`wxBITMAP_TYPE_XBM`)
- XPM data and file (`wxBITMAP_TYPE_XPM`)

wxIcon

Under Windows, *wxIcon* may load the following formats:

- Windows icon resource (`wxBITMAP_TYPE_ICO_RESOURCE`)
- Windows icon file (`wxBITMAP_TYPE_ICO`)
- XPM data and file (`wxBITMAP_TYPE_XPM`)

Under *wxGTK*, *wxIcon* may load the following formats:

- PNG (`wxBITMAP_TYPE_PNG`).
- XPM data and file (`wxBITMAP_TYPE_XPM`)

Under *wxMotif*, *wxIcon* may load the following formats:

- XBM data and file (`wxBITMAP_TYPE_XBM`)
- XPM data and file (`wxBITMAP_TYPE_XPM`)

wxCursor

Under Windows, *wxCursor* may load the following formats:

- Windows cursor resource (`wxBITMAP_TYPE_CUR_RESOURCE`)
- Windows cursor file (`wxBITMAP_TYPE_CUR`)

- Windows icon file (wxBITMAP_TYPE_ICO)
- Windows bitmap file (wxBITMAP_TYPE_BMP)

Under wxGTK, wxCursor may load the following formats (in addition to stock cursors):

- None (stock cursors only).

Under wxMotif, wxCursor may load the following formats:

- XBM data and file (wxBITMAP_TYPE_XBM)

7.6.2 Bitmap format handlers

To provide extensibility, the functionality for loading and saving bitmap formats is not implemented in the wxBitmap class, but in a number of handler classes, derived from wxBitmapHandler. There is a static list of handlers which wxBitmap examines when a file load/save operation is requested. Some handlers are provided as standard, but if you have special requirements, you may wish to initialise the wxBitmap class with some extra handlers which you write yourself or receive from a third party.

To add a handler object to wxBitmap, your application needs to include the header which implements it, and then call the static function *wxBitmap::AddHandler* (p. 42). For example:

```
#include <wx/pnghand.h>
#include <wx/xpmhand.h>
...
// Initialisation
wxBitmap::AddHandler(new wxPNGFileHandler);
wxBitmap::AddHandler(new wxXPMFileHandler);
wxBitmap::AddHandler(new wxXPMDDataHandler);
...
```

Assuming the handlers have been written correctly, you should now be able to load and save PNG files and XPM files using the usual wxBitmap API.

Note: bitmap handlers are not implemented on all platforms. Currently, the above is only necessary on Windows, to save the extra overhead of formats that may not be necessary (if you don't use them, they are not linked into the executable). Unix platforms have PNG and XPM capability built-in (where supported).

7.7 wxDialog overview

Classes: *wxDialog* (p. 178)

A dialog box is similar to a panel, in that it is a window which can be used for placing controls, with the following exceptions:

1. A surrounding frame is implicitly created.
2. Extra functionality is automatically given to the dialog box, such as tabbing between items (currently Windows only).
3. If the dialog box is *modal*, the calling program is blocked until the dialog box is dismissed.

Under Windows 3, modal dialogs have to be emulated using modeless dialogs and a message loop. This is because Windows 3 expects the contents of a modal dialog to be loaded from a resource file or created on receipt of a dialog initialization message. This is too restrictive for *wxWindows*, where any window may be created and displayed before its contents are created.

For a set of dialog convenience functions, including file selection, see *Dialog functions* (p. 852).

See also *wxPanel* (p. 488) and *wxWindow* (p. 798) for inherited member functions. Validation of data in controls is covered in *Validator overview* (p. 969).

7.8 Font overview

Class: *wxFont* (p. 264)

A font is an object which determines the appearance of text, primarily when drawing text to a window or device context. A font is determined by up to six parameters:

Point size	This is the standard way of referring to text size.
Family	Supported families are: wxDEFAULT , wxDECORATIVE , wxROMAN , wxSCRIPT , wxSWISS , wxMODERN . wxMODERN is a fixed pitch font; the others are either fixed or variable pitch.
Style	The value can be wxNORMAL , wxSLANT or wxITALIC .
Weight	The value can be wxNORMAL , wxLIGHT or wxBOLD .
Underlining	The value can be TRUE or FALSE.
Face name	An optional string specifying the actual typeface to be used. If NULL, a default typeface will chosen based on the family.

Specifying a family, rather than a specific typeface name, ensures a degree of portability across platforms because a suitable font will be chosen for the given font family.

Under Windows, the face name can be one of the installed fonts on the user's system. Since the choice of fonts differs from system to system, either choose standard Windows fonts, or if allowing the user to specify a face name, store the family id with any file that might be transported to a different Windows machine or other platform.

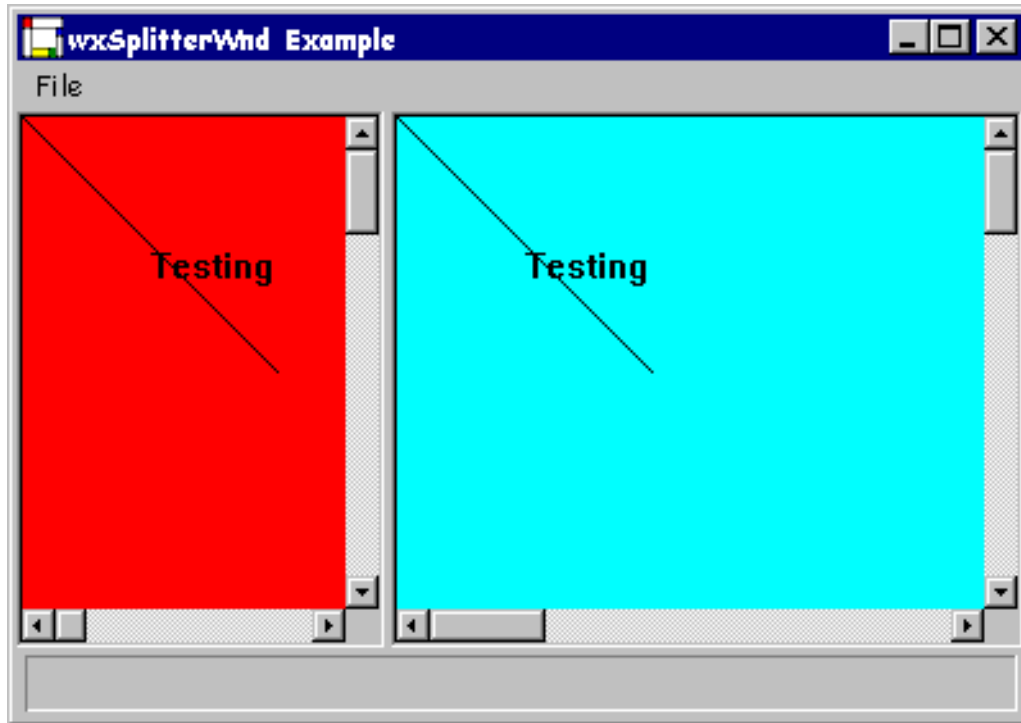
Note: There is currently a difference between the appearance of fonts on the two platforms, if the mapping mode is anything other than `wxMM_TEXT`. Under X, font size is always specified in points. Under MS Windows, the unit for text is points but the text is

scaled according to the current mapping mode. However, user scaling on a device context will also scale fonts under both environments.

7.9 wxSplitterWindow overview

Classes: *wxSplitterWindow* (p. 626)

The following screenshot shows the appearance of a splitter window with a vertical split.



The style `wxSP_3D` has been used to show a 3D border and 3D sash.

7.9.1 Example

The following fragment shows how to create a splitter window, creating two subwindows and hiding one of them.

```
splitter = new wxSplitterWindow(this, -1, wxPoint(0, 0), wxSize(400,
400), wxSP_3D);

leftWindow = new MyWindow(splitter);
leftWindow->SetScrollbars(20, 20, 50, 50);

rightWindow = new MyWindow(splitter);
rightWindow->SetScrollbars(20, 20, 50, 50);
rightWindow->Show(FALSE);

splitter->Initialize(leftWindow);
```



```
// Set this to prevent unsplitting
// splitter->SetMinimumPaneSize(20);
```

The next fragment shows how the splitter window can be manipulated after creation.

```
void MyFrame::OnSplitVertical(wxCommandEvent& event)
{
    if ( splitter->IsSplit() )
        splitter->Unsplit();
    leftWindow->Show(TRUE);
    rightWindow->Show(TRUE);
    splitter->SplitVertically( leftWindow, rightWindow );
}

void MyFrame::OnSplitHorizontal(wxCommandEvent& event)
{
    if ( splitter->IsSplit() )
        splitter->Unsplit();
    leftWindow->Show(TRUE);
    rightWindow->Show(TRUE);
    splitter->SplitHorizontally( leftWindow, rightWindow );
}

void MyFrame::OnUnsplit(wxCommandEvent& event)
{
    if ( splitter->IsSplit() )
        splitter->Unsplit();
}
```

7.10 wxTreeCtrl overview

Classes: *wxTreeCtrl* (p. 761), *wxImageList* (p. 339)

The tree control displays its items in a tree like structure. Each item has its own (optional) icon and a label. An item may be either collapsed (meaning that its children are not visible) or expanded (meaning that its children are shown). Each item in the tree is identified by its *itemId* which is of opaque data type *wxTreeItemId*.

The items text and image may be retrieved and changed with *GetItemText* (p. 767)/*SetItemText* (p. 772) and *GetItemImage* (p. 767)/*SetItemImage* (p. 772). In fact, an item may even have two images associated with it: the normal one and another one for selected state which is set/retrieved with *SetItemSelectedImage* (p. 772)/*GetItemSelectedImage* (p. 769) functions, but this functionality might be unavailable on some platforms.

Tree items have several attributes: an item may be selected or not, visible or not, bold or not. It may also be expanded or collapsed. All these attributes may be retrieved with the corresponding functions: *IsSelected* (p. 770), *IsVisible* (p. 770), *IsBold* (p. 770) and *IsExpanded* (p. 770). Only one item at a time may be selected, selecting another one

(with *SelectItem* (p. 771)) automatically unselects the previously selected one.

In addition to its icon and label, a user-specific data structure may be associated with all tree items. If you wish to do it, you should derive a class from *wxTreeItemData* which is a very simple class having only one function *GetId()* which returns the id of the item this data is associated with. This data will be freed by the control itself when the associated item is deleted (all items are deleted when the control is destroyed), so you shouldn't delete it yourself (if you do it, you should call *SetItemData(NULL)* (p. 772) to prevent the tree from deleting the pointer second time). The associated data may be retrieved with *GetItemData()* (p. 766) function.

Working with trees is relatively straightforward if all the items are added to the tree at the moment of its creation. However, for large trees it may be very inefficient. To improve the performance you may want to delay adding the items to the tree until the branch containing the items is expanded: so, in the beginning, only the root item is created (with *AddRoot* (p. 763)). Other items are added when *EVT_TREE_ITEM_EXPANDING* event is received: then all items lying immediately under the item being expanded should be added, but, of course, only when this event is received for the first time for this item - otherwise, the items would be added twice if the user expands/collapses/reexpands the branch.

The tree control provides functions for enumerating its items. There are 3 groups of enumeration functions: for the children of a given item, for the sibling of the given item and for the visible items (those which are currently shown to the user: an item may be invisible either because its branch is collapsed or because it is scrolled out of view). Child enumeration functions require the caller to give them a *cookie* parameter: it is a number which is opaque to the caller but is used by the tree control itself to allow multiple enumerations to run simultaneously (this is explicitly allowed). The only thing to remember is that the *cookie* passed to *GetFirstChild* (p. 766) and to *GetNextChild* (p. 767) should be the same variable (and that nothing should be done with it by the user code).

Among other features of the tree control are: item sorting with *SortChildren* (p. 772) which uses the user-defined comparison function *OnCompareItems* (p. 770) (by default the comparison is the alphabetic comparison of tree labels), hit testing (determining to which portion of the control the given point belongs, useful for implementing drag-and-drop in the tree) with *HitTest* (p. 769) and editing of the tree item labels in place (see *EditLabel* (p. 764)).

Finally, the tree control has a keyboard interface: the cursor navigation (arrow) keys may be used to change the current selection. <HOME> and <END> are used to go to the first/last sibling of the current item. '+', '-' and '*' expand, collapse and toggle the current branch. Note, however, that and <INS> keys do nothing by default, but it is usual to associate them with deleting item from a tree and inserting a new one into it.

7.11 wxListCtrl overview

Classes: *wxListCtrl* (p. 381), *wxImageList* (p. 339)

Sorry, this topic has yet to be written.

7.12 wxImageList overview

Classes: *wxImageList* (p. 339)

An image list is a list of images that may have transparent areas. The class helps an application organise a collection of images so that they can be referenced by integer index instead of by pointer.

Image lists are used in *wxNotebook* (p. 464), *wxListCtrl* (p. 381), *wxTreeCtrl* (p. 381) and some other control classes.

7.13 Common dialogs overview

Classes: *wxColourDialog* (p. 93), *wxFontDialog* (p. 273), *wxPrintDialog* (p. 512), *wxFileDialog* (p. 248), *wxDirDialog* (p. 185), *wxTextEntryDialog* (p. 725), *wxMessageDialog* (p. 439), *wxSingleChoiceDialog* (p. 593), *wxMultipleChoiceDialog* (p. 459)

Common dialog classes and functions encapsulate commonly-needed dialog box requirements. They are all 'modal', grabbing the flow of control until the user dismisses the dialog, to make them easy to use within an application.

Some dialogs have both platform-dependent and platform-independent implementations, so that if underlying windowing systems that do not provide the required functionality, the generic classes and functions can stand in. For example, under MS Windows, *wxColourDialog* uses the standard colour selector. There is also an equivalent called *wxGenericColourDialog* for other platforms, and a macro defines *wxColourDialog* to be the same as *wxGenericColourDialog* on non-MS Windows platforms. However, under MS Windows, the generic dialog can also be used, for testing or other purposes.

7.13.1 wxColourDialog overview

Classes: *wxColourDialog* (p. 93), *wxColourData* (p. 89)

The *wxColourDialog* presents a colour selector to the user, and returns with colour information.

The MS Windows colour selector

Under Windows, the native colour selector common dialog is used. This presents a dialog box with three main regions: at the top left, a palette of 48 commonly-used colours is shown. Under this, there is a palette of 16 'custom colours' which can be set by the application if desired. Additionally, the user may open up the dialog box to show a right-hand panel containing controls to select a precise colour, and add it to the custom colour palette.

The generic colour selector

Under non-MS Windows platforms, the colour selector is a simulation of most of the

features of the MS Windows selector. Two palettes of 48 standard and 16 custom colours are presented, with the right-hand area containing three sliders for the user to select a colour from red, green and blue components. This colour may be added to the custom colour palette, and will replace either the currently selected custom colour, or the first one in the palette if none is selected. The RGB colour sliders are not optional in the generic colour selector. The generic colour selector is also available under MS Windows; use the name `wxGenericColourDialog`.

Example

In the `samples/dialogs` directory, there is an example of using the `wxColourDialog` class. Here is an excerpt, which sets various parameters of a `wxColourData` object, including a grey scale for the custom colours. If the user did not cancel the dialog, the application retrieves the selected colour and uses it to set the background of a window.

```
wxColourData data;
data.SetChooseFull(TRUE);
for (int i = 0; i < 16; i++)
{
    wxColour colour(i*16, i*16, i*16);
    data.SetCustomColour(i, colour);
}

wxColourDialog dialog(this, &data);
if (dialog.ShowModal() == wxID_OK)
{
    wxColourData retData = dialog.GetColourData();
    wxColour col = retData.GetColour();
    wxBrush brush(col, wxSOLID);
    myWindow->SetBackground(brush);
    myWindow->Clear();
    myWindow->Refresh();
}
```

7.13.2 wxFontDialog overview

Classes: *wxFontDialog* (p. 273), *wxFontData* (p. 270)

The `wxFontDialog` presents a font selector to the user, and returns with font and colour information.

The MS Windows font selector

Under Windows, the native font selector common dialog is used. This presents a dialog box with controls for font name, point size, style, weight, underlining, strikeouts and text foreground colour. A sample of the font is shown on a white area of the dialog box. Note that in the translation from full MS Windows fonts to wxWindows font conventions, strikeouts is ignored and a font family (such as Swiss or Modern) is deduced from the actual font name (such as Arial or Courier). The full range of Windows fonts cannot be used in wxWindows at present.

The generic font selector

Under non-MS Windows platforms, the font selector is simpler. Controls for font family, point size, style, weight, underlining and text foreground colour are provided, and a sample is shown upon a white background. The generic font selector is also available under MS Windows; use the name `wxGenericFontDialog`.

In both cases, the application is responsible for deleting the new font returned from calling `wxFontDialog::Show` (if any). This returned font is guaranteed to be a new object and not one currently in use in the application.

Example

In the `samples/dialogs` directory, there is an example of using the `wxFontDialog` class. The application uses the returned font and colour for drawing text on a canvas. Here is an excerpt:

```
wxFontData data;
data.SetInitialFont(canvasFont);
data.SetColour(canvasTextColour);

wxFontDialog dialog(this, &data);
if (dialog.ShowModal() == wxID_OK)
{
    wxFontData retData = dialog.GetFontData();
    canvasFont = retData.GetChosenFont();
    canvasTextColour = retData.GetColour();
    myWindow->Refresh();
}
```

7.13.3 wxPrintDialog overview

Classes: *wxPrintDialog* (p. 512), *wxPrintData* (p. 508)

This class represents the print and print setup common dialogs. You may obtain a *wxPrinterDC* (p. 515) device context from a successfully dismissed print dialog.

The `samples/printing` example shows how to use it: see *Printing overview* (p. 950) for an excerpt from this example.

7.13.4 wxFileDialog overview

Classes: *wxFileDialog* (p. 248)

Pops up a file selector box. In Windows, this is the common file selector dialog. In X, this is a file selector box with somewhat less functionality. The path and filename are distinct elements of a full file pathname. If path is "", the current directory will be used. If filename is "", no default filename will be supplied. The wildcard determines what files are displayed in the file selector, and file extension supplies a type extension for the required filename. Flags may be a combination of `wxOPEN`, `wxSAVE`, `wxOVERWRITE_PROMPT`, `wxHIDE_READONLY`, or 0. They are only significant at present in Windows.

Both the X and Windows versions implement a wildcard filter. Typing a filename containing wildcards (*, ?) in the filename text item, and clicking on Ok, will result in only those files matching the pattern being displayed. In the X version, supplying no default name will result in the wildcard filter being inserted in the filename text item; the filter is ignored if a default name is supplied.

Under Windows (only), the wildcard may be a specification for multiple types of file with a description for each, such as:

```
"BMP files (*.bmp) | *.bmp | GIF files (*.gif) | *.gif"
```

7.13.5 wxDirDialog overview

Classes: *wxDirDialog* (p. 185)

This dialog shows a directory selector dialog, allowing the user to select a single directory.

7.13.6 wxTextEntryDialog overview

Classes: *wxTextEntryDialog* (p. 725)

This is a dialog with a text entry field. The value that the user entered is obtained using *wxTextEntryDialog::GetValue* (p. 726).

7.13.7 wxMessageDialog overview

Classes: *wxMessageDialog* (p. 439)

This dialog shows a message, plus buttons that can be chosen from OK, Cancel, Yes, and No. Under Windows, an optional icon can be shown, such as an exclamation mark or question mark.

The return value of *wxMessageDialog::ShowModal* (p. 440) indicates which button the user pressed.

7.13.8 wxSingleChoiceDialog overview

Classes: *wxSingleChoiceDialog* (p. 593)

This dialog shows a list of choices, plus OK and (optionally) Cancel. The user can select one of them. The selection can be obtained from the dialog as an index, a string or client data.

7.13.9 wxMultipleChoiceDialog overview

Classes: *wxMultipleChoiceDialog* (p. 459)

This dialog shows a list of choices, plus OK and (optionally) Cancel. The user can select one or more of them.

7.14 Constraints overview

Classes: *wxLayoutConstraints* (p. 365), *wxIndividualLayoutConstraint* (p. 342).

Objects of class *wxLayoutConstraint* can be associated with a window to define the way its subwindows are laid out, with respect to their siblings or parent.

The class consists of the following eight constraints of class *wxIndividualLayoutConstraint*, some or all of which should be accessed directly to set the appropriate constraints.

- **left:** represents the left hand edge of the window
- **right:** represents the right hand edge of the window
- **top:** represents the top edge of the window
- **bottom:** represents the bottom edge of the window
- **width:** represents the width of the window
- **height:** represents the height of the window
- **centreX:** represents the horizontal centre point of the window
- **centreY:** represents the vertical centre point of the window

Most constraints are initially set to have the relationship *wxUnconstrained*, which means that their values should be calculated by looking at known constraints. The exceptions are *width* and *height*, which are set to *wxAsIs* to ensure that if the user does not specify a constraint, the existing width and height will be used, to be compatible with panel items which often have take a default size. If the constraint is *wxAsIs*, the dimension will not be changed.

To call the *wxWindow::Layout* (p. 815) function which evaluates constraints, you can either call *wxWindow::SetAutoLayout* to tell default *OnSize* handlers to call *Layout*, or override *OnSize* and call *Layout* yourself.

7.14.1 Constraint layout: more detail

By default, windows do not have a *wxLayoutConstraints* object. In this case, much layout must be done explicitly, by performing calculations in *OnSize* members, except for the case of frames that have one subwindow, where *wxFrame::OnSize* takes care of resizing the child.

To avoid the need for these rather awkward calculations, the user can create a *wxLayoutConstraints* object and associate it with a window with *wxWindow::SetConstraints*. This object contains a constraint for each of the window edges, two for the centre point, and two for the window size. By setting some or all of these constraints appropriately, the user can achieve quite complex layout by defining

relationships between windows.

In `wxWindows`, each window can be constrained relative to either its *siblings* on the same window, or the *parent*. The layout algorithm therefore operates in a top-down manner, finding the correct layout for the children of a window, then the layout for the grandchildren, and so on. Note that this differs markedly from native Motif layout, where constraints can ripple upwards and can eventually change the frame window or dialog box size. We assume in `wxWindows` that the *user* is always 'boss' and specifies the size of the outer window, to which subwindows must conform. Obviously, this might be a limitation in some circumstances, but it suffices for most situations, and the simplification avoids some of the nightmarish problems associated with programming Motif.

When the user sets constraints, many of the constraints for windows edges and dimensions remain unconstrained. For a given window, the `wxWindow::Layout` algorithm first resets all constraints in all children to have unknown edge or dimension values, and then iterates through the constraints, evaluating them. For unconstrained edges and dimensions, it tries to find the value using known relationships that always hold. For example, an unconstrained *width* may be calculated from the *left* and *right edges*, if both are currently known. For edges and dimensions with user-supplied constraints, these constraints are evaluated if the inputs of the constraint are known.

The algorithm stops when all child edges and dimension are known (success), or there are unknown edges or dimensions but there has been no change in this cycle (failure).

It then sets all the window positions and sizes according to the values it has found.

Because the algorithm is iterative, the order in which constraints are considered is irrelevant.

7.14.2 Window layout examples

7.14.2.1 Example 1: subwindow layout

This example specifies a panel and a window side by side, with a text subwindow below it.

```
frame->panel = new wxPanel(frame, -1, wxPoint(0, 0), wxSize(1000,
500), 0);
frame->scrollWindow = new MyScrolledWindow(frame, -1, wxPoint(0, 0),
wxSize(400, 400), wxRETAINED);
frame->text_window = new MyTextWindow(frame, -1, wxPoint(0, 250),
wxSize(400, 250));

// Set constraints for panel subwindow
wxLayoutConstraints *c1 = new wxLayoutConstraints;

c1->left.SameAs      (frame, wxLeft);
c1->top.SameAs       (frame, wxTop);
c1->right.PercentOf  (frame, wxWidth, 50);
c1->height.PercentOf (frame, wxHeight, 50);
```



```

frame->panel->SetConstraints(c1);

// Set constraints for scrollWindow subwindow
wxLayoutConstraints *c2 = new wxLayoutConstraints;

c2->left.SameAs      (frame->panel, wxRight);
c2->top.SameAs       (frame, wxTop);
c2->right.SameAs     (frame, wxRight);
c2->height.PercentOf (frame, wxHeight, 50);

frame->scrollWindow->SetConstraints(c2);

// Set constraints for text subwindow
wxLayoutConstraints *c3 = new wxLayoutConstraints;
c3->left.SameAs      (frame, wxLeft);
c3->top.Below        (frame->panel);
c3->right.SameAs     (frame, wxRight);
c3->bottom.SameAs    (frame, wxBottom);

frame->text_window->SetConstraints(c3);

```

7.14.2.2 Example 2: panel item layout

This example sizes a button width to 80 percent of the panel width, and centres it horizontally. A listbox and multitext item are placed below it. The listbox takes up 40 percent of the panel width, and the multitext item takes up the remainder of the width. Margins of 5 pixels are used.

```

// Create some panel items
wxButton *btn1 = new wxButton(frame->panel, -1, "A button") ;

wxLayoutConstraints *b1 = new wxLayoutConstraints;
b1->centreX.SameAs      (frame->panel, wxCentreX);
b1->top.SameAs          (frame->panel, wxTop, 5);
b1->width.PercentOf     (frame->panel, wxWidth, 80);
b1->height.PercentOf    (frame->panel, wxHeight, 10);
btn1->SetConstraints(b1);

wxListBox *list = new wxListBox(frame->panel, -1, "A list",
                                wxPoint(-1, -1), wxSize(200, 100));

wxLayoutConstraints *b2 = new wxLayoutConstraints;
b2->top.Below           (btn1, 5);
b2->left.SameAs         (frame->panel, wxLeft, 5);
b2->width.PercentOf     (frame->panel, wxWidth, 40);
b2->bottom.SameAs       (frame->panel, wxBottom, 5);
list->SetConstraints(b2);

wxTextCtrl *mtext = new wxTextCtrl(frame->panel, -1, "Multiline text",
" Some text",
                                wxPoint(-1, -1), wxSize(150, 100),
wxTE_MULTILINE);

wxLayoutConstraints *b3 = new wxLayoutConstraints;
b3->top.Below           (btn1, 5);
b3->left.RightOf        (list, 5);
b3->right.SameAs        (frame->panel, wxRight, 5);

```

```
b3->bottom.SameAs      (frame->panel, wxBottom, 5);
mtext->SetConstraints(b3);
```

7.15

Database classes overview

Classes: *wxDatabase* (p. 132), *wxRecordSet* (p. 550), *wxQueryCol* (p. 530), *wxQueryField* (p. 533)

Note that more sophisticated ODBC classes are provided by the Remstar database classes: please see the separate HTML and Word documentation.

wxWindows provides a set of classes for accessing a subset of Microsoft's ODBC (Open Database Connectivity) product. Currently, this wrapper is available under MS Windows only, although ODBC may appear on other platforms, and a generic or product-specific SQL emulator for the ODBC classes may be provided in *wxWindows* at a later date.

ODBC presents a unified API (Application Programmer's Interface) to a wide variety of databases, by interfacing indirectly to each database or file via an ODBC driver. The language for most of the database operations is SQL, so you need to learn a small amount of SQL as well as the *wxWindows* ODBC wrapper API. Even though the databases may not be SQL-based, the ODBC drivers translate SQL into appropriate operations for the database or file: even text files have rudimentary ODBC support, along with dBASE, Access, Excel and other file formats.

The run-time files for ODBC are bundled with many existing database packages, including MS Office. The required header files, *sql.h* and *sqlext.h*, are bundled with several compilers including MS VC++ and Watcom C++. The only other way to obtain these header files is from the ODBC SDK, which is only available with the MS Developer Network CD-ROMs -- at great expense. If you have *odbc.dll*, you can make the required import library *odbc.lib* using the tool 'implib'. You need to have *odbc.lib* in your compiler library path.

The minimum you need to distribute with your application is *odbc.dll*, which must go in the Windows system directory. For the application to function correctly, ODBC drivers must be installed on the user's machine. If you do not use the database classes, *odbc.dll* will be loaded but not called (so ODBC does not need to be setup fully if no ODBC calls will be made).

A sample is distributed with *wxWindows* in *samples/odbc*. You will need to install the sample dbf file as a data source using the ODBC setup utility, available from the control panel if ODBC has been fully installed.

7.15.1 Procedures for writing an ODBC application

You first need to create a *wxDatabase* object. If you want to get information from the ODBC manager instead of from a particular database (for example using *wxRecordSet::GetDataSources* (p. 554)), then you do not need to call *wxDatabase::Open* (p. 136). If you do wish to connect to a datasource, then call *wxDatabase::Open*. You can reuse your *wxDatabase* object, calling *wxDatabase::Close*

and `wxDatabase::Open` multiple times.

Then, create a `wxRecordSet` object for retrieving or sending information. For ODBC manager information retrieval, you can create it as a dynaset (retrieve the information as needed) or a snapshot (get all the data at once). If you are going to call `wxRecordSet::ExecuteSQL` (p. 553), you need to create it as a snapshot. Dynaset mode is not yet implemented for user data.

Having called a function such as `wxRecordSet::ExecuteSQL` or `wxRecordSet::GetDataSources`, you may have a number of records associated with the recordset, if appropriate to the operation. You can now retrieve information such as the number of records retrieved and the actual data itself. Use `wxRecordSet::GetFieldData` (p. 555) or `wxRecordSet::GetFieldDataPtr` (p. 556) to get the data or a pointer to it, passing a column index or name. The data returned will be for the current record. To move around the records, use `wxRecordSet::MoveNext` (p. 561), `wxRecordSet::MovePrev` (p. 561) and associated functions.

You can use the same recordset for multiple operations, or delete the recordset and create a new one.

Note that when you delete a `wxDatabase`, any associated recordsets also get deleted, so beware of holding onto invalid pointers.

7.15.2 wxDatabase overview

Database classes overview (p. 922)

Class: `wxDatabase` (p. 132)

Every database object represents an ODBC connection. To do anything useful with a database object you need to bind a `wxRecordSet` object to it. All you can do with `wxDatabase` is opening/closing connections and getting some info about it (users, passwords, and so on).

7.15.3 wxQueryCol overview

Database classes overview (p. 922)

Class: `wxQueryCol` (p. 530)

Every data column is represented by an instance of this class. It contains the name and type of a column and a list of `wxQueryFields` where the real data is stored. The links to user-defined variables are stored here, as well.

7.15.4 wxQueryField overview

Database classes overview (p. 922)

Class: *wxQueryField* (p. 533)

As every data column is represented by an instance of the class *wxQueryCol*, every data item of a specific column is represented by an instance of *wxQueryField*. Each column contains a list of *wxQueryFields*. If *wxRecordSet* is of the type *wxOPEN_TYPE_DYNASET*, there will be only one field for each column, which will be updated every time you call functions like *wxRecordSet::Move* or *wxRecordSet::GoTo*. If *wxRecordSet* is of the type *wxOPEN_TYPE_SNAPSHOT*, all data returned by an ODBC function will be loaded at once and the number of *wxQueryField* instances for each column will depend on the number of records.

7.15.5 wxRecordSet overview

Database classes overview (p. 922)

Class: *wxRecordSet* (p. 550)

Each *wxRecordSet* represents a database query. You can make multiple queries at a time by using multiple *wxRecordSets* with a *wxDatabase* or you can make your queries in sequential order using the same *wxRecordSet*.

7.15.6 ODBC SQL data types

Database classes overview (p. 922)

These are the data types supported in ODBC SQL. Note that there are other, extended level conformance types, not currently supported in *wxWindows*.

CHAR(<i>n</i>)	A character string of fixed length <i>n</i> .
VARCHAR(<i>n</i>)	A varying length character string of maximum length <i>n</i> .
LONG VARCHAR(<i>n</i>)	A varying length character string: equivalent to VARCHAR for the purposes of ODBC.
DECIMAL(<i>p</i> , <i>s</i>)	An exact numeric of precision <i>p</i> and scale <i>s</i> .
NUMERIC(<i>p</i> , <i>s</i>)	Same as DECIMAL.
SMALLINT	A 2 byte integer.
INTEGER	A 4 byte integer.
REAL	A 4 byte floating point number.
FLOAT	An 8 byte floating point number.
DOUBLE PRECISION	Same as FLOAT.

These data types correspond to the following ODBC identifiers:

SQL_CHAR	A character string of fixed length.
SQL_VARCHAR	A varying length character string.
SQL_DECIMAL	An exact numeric.
SQL_NUMERIC	Same as SQL_DECIMAL.
SQL_SMALLINT	A 2 byte integer.

SQL_INTEGER	A 4 byte integer.
SQL_REAL	A 4 byte floating point number.
SQL_FLOAT	An 8 byte floating point number.
SQL_DOUBLE	Same as SQL_FLOAT.

7.15.7 A selection of SQL commands

Database classes overview (p. 922)

The following is a very brief description of some common SQL commands, with examples.

7.15.7.1 Create

Creates a table.

Example:

```
CREATE TABLE Book
  (BookNumber      INTEGER      PRIMARY KEY
  , CategoryCode   CHAR(2)      DEFAULT 'RO' NOT NULL
  , Title          VARCHAR(100) UNIQUE
  , NumberOfPages  SMALLINT
  , RetailPriceAmount NUMERIC(5,2)
  )
```

7.15.7.2 Insert

Inserts records into a table.

Example:

```
INSERT INTO Book
  (BookNumber, CategoryCode, Title)
VALUES(5, 'HR', 'The Lark Ascending')
```

7.15.7.3 Select

The Select operation retrieves rows and columns from a table. The criteria for selection and the columns returned may be specified.

Examples:

```
SELECT * FROM Book
```

Selects all rows and columns from table Book.

```
SELECT Title, RetailPriceAmount FROM Book WHERE RetailPriceAmount > 20.0
```

Selects columns Title and RetailPriceAmount from table Book, returning only the rows that match the WHERE clause.

```
SELECT * FROM Book WHERE CatCode = 'LL' OR CatCode = 'RR'
```

Selects all columns from table Book, returning only the rows that match the WHERE clause.

```
SELECT * FROM Book WHERE CatCode IS NULL
```

Selects all columns from table Book, returning only rows where the CatCode column is NULL.

```
SELECT * FROM Book ORDER BY Title
```

Selects all columns from table Book, ordering by Title, in ascending order. To specify descending order, add DESC after the ORDER BY Title clause.

```
SELECT Title FROM Book WHERE RetailPriceAmount >= 20.0 AND  
RetailPriceAmount <= 35.0
```

Selects records where RetailPriceAmount conforms to the WHERE expression.

7.15.7.4 Update

Updates records in a table.

Example:

```
UPDATE Incident SET X = 123 WHERE ASSET = 'BD34'
```

This example sets a field in column 'X' to the number 123, for the record where the column ASSET has the value 'BD34'.

7.16 Device context overview

Classes: *wxDC* (p. 151), *wxPostScriptDC* (p. 504), *wxMetafileDC* (p. 442), *wxMemoryDC* (p. 415), *wxPrinterDC* (p. 515), *wxScreenDC* (p. 578), *wxClientDC* (p. 81), *wxPaintDC* (p. 483), *wxWindowDC* (p. 843).

A *wxDC* is a *device context* onto which graphics and text can be drawn. The device context is intended to represent a number of output devices in a generic way, with the same API being used throughout.

Some device contexts are created temporarily in order to draw on a window. This is true of *wxScreenDC* (p. 578), *wxClientDC* (p. 81), *wxPaintDC* (p. 483), and *wxWindowDC* (p. 843). The following describes the differences between these device contexts and when you should use them.

- **wxScreenDC.** Use this to paint on the screen, as opposed to an individual window.
- **wxClientDC.** Use this to paint on the client area of window (the part without borders and other decorations), but do not use it from within an *wxWindow::OnPaint* (p. 825) event.
- **wxPaintDC.** Use this to paint on the client area of a window, but *only* from within an *wxWindow::OnPaint* (p. 825) event.
- **wxWindowDC.** Use this to paint on the whole area of a window, including decorations. This may not be available on non-Windows platforms.

To use a client, paint or window device context, create an object on the stack with the window as argument, for example:

```
void MyWindow::OnMyCmd(wxCommandEvent& event)
{
    wxClientDC dc(window);
    DrawMyPicture(dc);
}
```

Try to write code so it is parameterised by wxDC - if you do this, the same piece of code may write to a number of different devices, by passing a different device context. This doesn't work for everything (for example not all device contexts support bitmap drawing) but will work most of the time.

7.17 Debugging overview

Classes, functions and macros: *wxDebugContext* (p. 173), *wxObject* (p. 471), *wxLog* (p. 399), *Log functions* (p. 884), *Debug macros* (p. 886)

Various classes, functions and macros are provided in wxWindows to help you debug your application. Most of these are only available if you compile both wxWindows, your application and *all* libraries that use wxWindows with the `__WXDEBUG__` symbol defined. You can also test the `__WXDEBUG__` symbol in your own applications to execute code that should be active only in debug mode.

wxDebugContext

wxDebugContext (p. 173) is a class that never gets instantiated, but ties together various static functions and variables. It allows you to dump all objects to that stream, write statistics about object allocation, and check memory for errors.

It is good practice to define a *wxObject::Dump* (p. 472) member function for each class you derive from a wxWindows class, so that *wxDebugContext::Dump* (p. 173) can call it and give valuable information about the state of the application.

If you have difficulty tracking down a memory leak, recompile in debugging mode and call *wxDebugContext::Dump* (p. 173) and *wxDebugContext::PrintStatistics* (p. 175) at appropriate places. They will tell you what objects have not yet been deleted, and what kinds of object they are. In fact, in debug mode wxWindows will automatically detect memory leaks when your application is about to exit, and if there are any leaks, will give you information about the problem. (How much information depends on the operating

system and compiler -- some systems don't allow all memory logging to be enabled). See the memcheck sample for example of usage.

For `wxDebugContext` to do its work, the *new* and *delete* operators for `wxObject` have been redefined to store extra information about dynamically allocated objects (but not statically declared objects). This slows down a debugging version of an application, but can find difficult-to-detect memory leaks (objects are not deallocated), overwrites (writing past the end of your object) and underwrites (writing to memory in front of the object).

If debugging mode is on and the symbol `wxUSE_GLOBAL_MEMORY_OPERATORS` is set to 1 in `setup.h`, 'new' is defined to be:

```
#define new new(__FILE__, __LINE__)
```

All occurrences of 'new' in `wxWindows` and your own application will use the overridden form of the operator with two extra arguments. This means that the debugging output (and error messages reporting memory problems) will tell you what file and on what line you allocated the object. Unfortunately not all compilers allow this definition to work properly, but most do.

Debug macros

You should also use *debug macros* (p. 886) as part of a 'defensive programming' strategy, scattering `wxASSERT`s liberally to test for problems in your code as early as possible. Forward thinking will save a surprising amount of time in the long run.

`wxASSERT` (p. 887) is used to pop up an error message box when a condition is not true. You can also use `wxASSERT_MSG` (p. 887) to supply your own helpful error message. For example:

```
void MyClass::MyFunction(wxObject* object)
{
    wxASSERT_MSG( (object != NULL), "object should not be NULL in
MyFunction!" );

    ...
};
```

The message box allows you to continue execution or abort the program. If you are running the application inside a debugger, you will be able to see exactly where the problem was.

Logging functions

You can use the `wxLogDebug` (p. 886) and `wxLogTrace` (p. 886) functions to output debugging information in debug mode; it will do nothing for non-debugging code.

7.17.1 wxDebugContext overview

Debugging overview (p. 927)

Class: *wxDebugContext* (p. 173)

wxDebugContext is a class for performing various debugging and memory tracing operations.

This class has only static data and function members, and there should be no instances. Probably the most useful members are *SetFile* (for directing output to a file, instead of the default standard error or debugger output); *Dump* (for dumping the dynamically allocated objects) and *PrintStatistics* (for dumping information about allocation of objects). You can also call *Check* to check memory blocks for integrity.

Here's an example of use. The *SetCheckpoint* ensures that only the allocations done after the checkpoint will be dumped.

```
wxDebugContext::SetCheckpoint();

wxDebugContext::SetFile("c:\\temp\\debug.log");

wxString *thing = new wxString;

char *ordinaryNonObject = new char[1000];

wxDebugContext::Dump();
wxDebugContext::PrintStatistics();
```

You can use *wxDebugContext* if `__WXDEBUG__` is defined, or you can use it at any other time (if `wxUSE_DEBUG_CONTEXT` is set to 1 in `setup.h`). It is not disabled in non-debug mode because you may not wish to recompile *wxWindows* and your entire application just to make use of the error logging facility.

Note: *wxDebugContext::SetFile* has a problem at present, so use the default stream instead. Eventually the logging will be done through the *wxLog* facilities instead.

7.18 Window deletion overview

Classes: *wxCloseEvent* (p. 84), *wxWindow* (p. 798)

Window deletion can be a confusing subject, so this overview is provided to help make it clear when and how you delete windows, or respond to user requests to close windows.

What is the sequence of events in a window deletion?

When the user clicks on the system close button or system close command, in a frame or a dialog, *wxWindows* calls *wxWindow::Close* (p. 802). This in turn generates an `EVT_CLOSE` event: see *wxWindow::OnCloseWindow* (p. 819).

It is the duty of the application to define a suitable event handler, and decide whether or not to destroy the window. If the application is for some reason forcing the application to close (*wxCloseEvent::CanVeto* (p. 85) returns `FALSE`), the window should always be

destroyed, otherwise there is the option to ignore the request, or maybe wait until the user has answered a question before deciding whether it's safe to close. The handler for `EVT_CLOSE` should signal to the calling code if it does not destroy the window, by calling `wxCloseEvent::Veto` (p. 86). Calling this provides useful information to the calling code.

The `wxCloseEvent` handler should only call `wxWindow::Destroy` (p. 805) to delete the window, and not use the **delete** operator. This is because for some window classes, `wxWindows` delays actual deletion of the window until all events have been processed, since otherwise there is the danger that events will be sent to a non-existent window.

As reinforced in the next section, calling `Close` does not guarantee that the window will be destroyed. Call `wxWindow::Destroy` (p. 805) if you want to be certain that the window is destroyed.

How can the application close a window itself?

Your application can either use `wxWindow::Close` (p. 802) event just as the framework does, or it can call `wxWindow::Destroy` (p. 805) directly. If using `Close()`, you can pass a `TRUE` argument to this function to tell the event handler that we definitely want to delete the frame and it cannot be vetoed.

The advantage of using `Close` instead of `Destroy` is that it will call any clean-up code defined by the `EVT_CLOSE` handler; for example it may close a document contained in a window after first asking the user whether the work should be saved. `Close` can be vetoed by this process (return `FALSE`), whereas `Destroy` definitely destroys the window.

What is the default behaviour?

The default close event handler for `wxDialog` simulates a Cancel command, generating a `wxID_CANCEL` event. Since the handler for this cancel event might itself call **Close**, there is a check for infinite looping. The default handler for `wxID_CANCEL` hides the dialog (if modeless) or calls `EndModal(wxID_CANCEL)` (if modal). In other words, by default, the dialog *is not destroyed* (it might have been created on the stack, so the assumption of dynamic creation cannot be made).

The default close event handler for `wxFrame` destroys the frame using `Destroy()`.

Under Windows, `wxDialog` defines a handler for `wxWindow::OnCharHook` (p. 818) that generates a Cancel event if the Escape key has been pressed.

What should I do when the user calls up Exit from a menu?

You can simply call `wxWindow::Close` (p. 802) on the frame. This will invoke your own close event handler which may destroy the frame.

You can do checking to see if your application can be safely exited at this point, either from within your close event handler, or from within your exit menu command handler. For example, you may wish to check that all files have been saved. Give the user a chance to save and quit, to not save but quit anyway, or to cancel the exit command altogether.

What should I do to upgrade my 1.xx OnClose to 2.0?

In wxWindows 1.xx, the **OnClose** function did not actually delete 'this', but signalled to the calling function (either **Close**, or the wxWindows framework) to delete or not delete the window.

To update your code, you should provide an event table entry in your frame or dialog, using the EVT_CLOSE macro. The event handler function might look like this:

```
void MyFrame::OnCloseWindow(wxCloseEvent& event)
{
    if (MyDataHasBeenModified())
    {
        wxMessageDialog* dialog = new wxMessageDialog(this,
            "Save changed data?", "My app", wxYES_NO|wxCANCEL);

        int ans = dialog->ShowModal();
        dialog->Destroy();

        switch (ans)
        {
            case wxID_YES:        // Save, then destroy, quitting app
                SaveMyData();
                this->Destroy();
                break;
            case wxID_NO:         // Don't save; just destroy, quitting app
                this->Destroy();
                break;
            case wxID_CANCEL:     // Do nothing - so don't quit app.
            default:
                if (!event.CanVeto()) // Test if we can veto this deletion
                    this->Destroy(); // If not, destroy the window anyway.
                else
                    event.Veto();    // Notify the calling code that we didn't
delete the frame.
                break;
        }
    }
}
```

How do I exit the application gracefully?

A wxWindows application automatically exits when the designated top window, or the last frame or dialog, is destroyed. Put any application-wide cleanup code in *wxApp::OnExit* (p. 10) (this is a virtual function, not an event handler).

Do child windows get deleted automatically?

Yes, child windows are deleted from within the parent destructor. This includes any children that are themselves frames or dialogs, so you may wish to close these child frame or dialog windows explicitly from within the parent close handler.

What about other kinds of window?

So far we've been talking about 'managed' windows, i.e. frames and dialogs. Windows with parents, such as controls, don't have delayed destruction and don't usually have close event handlers, though you can implement them if you wish. For consistency, continue to use the `wxWindow::Destroy` (p. 805) function instead of the **delete** operator when deleting these kinds of windows explicitly.

7.19 Scrolling overview

Classes: `wxWindow` (p. 798), `wxScrolledWindow` (p. 586), `wxIcon` (p. 318), `wxScrollBar` (p. 579).

Scrollbars come in various guises in `wxWindows`. All windows have the potential to show a vertical scrollbar and/or a horizontal scrollbar: it's a basic capability of a window. However, in practice, not all windows do make use of scrollbars, such as a single-line `wxTextCtrl`.

Because any class derived from `wxWindow` (p. 798) may have scrollbars, there are functions to manipulate the scrollbars and event handlers to intercept scroll events. But just because a window generates a scroll event, doesn't mean that the window necessarily handles it and physically scrolls the window. The base class `wxWindow` in fact doesn't have any default functionality to handle scroll events. If you created a `wxWindow` object with scrollbars, and then clicked on the scrollbars, nothing at all would happen. This is deliberate, because the *interpretation* of scroll events varies from one window class to another.

`wxScrolledWindow` (p. 586) (formerly `wxCanvas`) is an example of a window that adds functionality to make scrolling really work. It assumes that scrolling happens in consistent units, not different-sized jumps, and that page size is represented by the visible portion of the window. It's suited to drawing applications, but perhaps not so suitable for a sophisticated editor in which the amount scrolled may vary according to the size of text on a given line. For this, you would derive from `wxWindow` and implement scrolling yourself. `wxGrid` (p. 297) is an example of a class that implements its own scrolling, largely because columns and rows can vary in size.

The scrollbar model

The function `wxWindow::SetScrollbar` (p. 837) gives a clue about the way a scrollbar is modelled. This function takes the following arguments:

orientation	Which scrollbar: <code>wxVERTICAL</code> or <code>wxHORIZONTAL</code> .
position	The position of the scrollbar in scroll units.
visible	The size of the visible portion of the scrollbar, in scroll units.
range	The maximum position of the scrollbar.
refresh	Whether the scrollbar should be repainted.

orientation determines whether we're talking about the built-in horizontal or vertical scrollbar.

position is simply the position of the 'thumb' (the bit you drag to scroll around). It's given in scroll units, and so is relative to the total range of the scrollbar.

visible gives the number of scroll units that represents the portion of the window currently visible. Normally, a scrollbar is capable of indicating this visually by showing a different length of thumb.

range is the maximum value of the scrollbar, where zero is the start position. You choose the units that suit you, so if you wanted to display text that has 100 lines, you would set this to 100. Note that this doesn't have to correspond to the number of pixels scrolled - it's up to you how you actually show the contents of the window.

refresh just indicates whether the scrollbar should be repainted immediately or not.

An example

Let's say you wish to display 50 lines of text, using the same font. The window is sized so that you can only see 16 lines at a time.

You would use:

```
SetScrollbar(wxVERTICAL, 0, 16, 50);
```

Note that with the window at this size, the thumb position can never go above 50 minus 16, or 34.

You can determine how many lines are currently visible by dividing the current view size by the character height in pixels.

When defining your own scrollbar behaviour, you will always need to recalculate the scrollbar settings when the window size changes. You could therefore put your scrollbar calculations and `SetScrollbar` call into a function named `AdjustScrollbars`, which can be called initially and also from your `wxWindow::OnSize` (p. 828) event handler function.

7.20 Document/view overview

Classes: `wxDocument` (p. 207), `wxView` (p. 793), `wxDocTemplate` (p. 202), `wxDocManager` (p. 189), `wxDocParentFrame` (p. 200), `wxDocChildFrame` (p. 187), `wxDocMDIParentFrame` (p. 199), `wxDocMDIChildFrame` (p. 197), `wxCommand` (p. 101), `wxCommandProcessor` (p. 108)

The document/view framework is found in most application frameworks, because it can dramatically simplify the code required to build many kinds of application.

The idea is that you can model your application primarily in terms of *documents* to store

data and provide interface-independent operations upon it, and *views* to visualise and manipulate the data. Documents know how to do input and output given stream objects, and views are responsible for taking input from physical windows and performing the manipulation on the document data. If a document's data changes, all views should be updated to reflect the change.

The framework can provide many user-interface elements based on this model. Once you have defined your own classes and the relationships between them, the framework takes care of popping up file selectors, opening and closing files, asking the user to save modifications, routing menu commands to appropriate (possibly default) code, even some default print/preview functionality and support for command undo/redo. The framework is highly modular, allowing overriding and replacement of functionality and objects to achieve more than the default behaviour.

These are the overall steps involved in creating an application based on the document/view framework:

1. Define your own document and view classes, overriding a minimal set of member functions e.g. for input/output, drawing and initialization.
2. Define any subwindows (such as a scrolled window) that are needed for the view(s). You may need to route some events to views or documents, for example `OnPaint` needs to be routed to `wxView::OnDraw`.
3. Decide what style of interface you will use: Microsoft's MDI (multiple document child frames surrounded by an overall frame), SDI (a separate, unconstrained frame for each document), or single-window (one document open at a time, as in Windows Write).
4. Use the appropriate `wxDocParentFrame` and `wxDocChildFrame` classes. Construct an instance of `wxDocParentFrame` in your `wxApp::OnInit`, and a `wxDocChildFrame` (if not single-window) when you initialize a view. Create menus using standard menu ids (such as `wxID_OPEN`, `wxID_PRINT`), routing non-application-specific identifiers to the base frame's `OnMenuCommand`.
5. Construct a single `wxDocManager` instance at the beginning of your `wxApp::OnInit`, and then as many `wxDocTemplate` instances as necessary to define relationships between documents and views. For a simple application, there will be just one `wxDocTemplate`.

If you wish to implement Undo/Redo, you need to derive your own class(es) from `wxCommand` and use `wxCommandProcessor::Submit` instead of directly executing code. The framework will take care of calling `Undo` and `Do` functions as appropriate, so long as the `wxID_UNDO` and `wxID_REDO` menu items are defined in the view menu.

Here are a few examples of the tailoring you can do to go beyond the default framework behaviour:

- Override `wxDocument::OnCreateCommandProcessor` to define a different Do/Undo strategy, or a command history editor.
- Override `wxView::OnCreatePrintout` to create an instance of a derived `wxPrintout` (p. 516) class, to provide multi-page document facilities.
- Override `wxDocManager::SelectDocumentPath` to provide a different file selector.

- Limit the maximum number of open documents and the maximum number of undo commands.

Note that to activate framework functionality, you need to use some or all of the *wxWindows predefined command identifiers* (p. 939) in your menus.

7.20.1 wxDocument overview

Document/view framework overview (p. 933)

Class: *wxDocument* (p. 207)

The *wxDocument* class can be used to model an application's file-based data. It is part of the document/view framework supported by *wxWindows*, and cooperates with the *wxView* (p. 793), *wxDocTemplate* (p. 202) and *wxDocManager* (p. 189) classes.

Using this framework can save a lot of routine user-interface programming, since a range of menu commands -- such as open, save, save as -- are supported automatically. The programmer just needs to define a minimal set of classes and member functions for the framework to call when necessary. Data, and the means to view and edit the data, are explicitly separated out in this model, and the concept of multiple *views* onto the same data is supported.

Note that the document/view model will suit many but not all styles of application. For example, it would be overkill for a simple file conversion utility, where there may be no call for *views* on *documents* or the ability to open, edit and save files. But probably the majority of applications are document-based.

See the example application in `samples/docview`.

To use the abstract *wxDocument* class, you need to derive a new class and override at least the member functions `SaveObject` and `LoadObject`. `SaveObject` and `LoadObject` will be called by the framework when the document needs to be saved or loaded.

Use the macros `DECLARE_DYNAMIC_CLASS` and `IMPLEMENT_DYNAMIC_CLASS` in order to allow the framework to create document objects on demand. When you create a *wxDocTemplate* (p. 202) object on application initialization, you should pass `CLASSINFO(YourDocumentClass)` to the *wxDocTemplate* constructor so that it knows how to create an instance of this class.

If you do not wish to use the *wxWindows* method of creating document objects dynamically, you must override `wxDocTemplate::CreateDocument` to return an instance of the appropriate class.

7.20.2 wxView overview

Document/view framework overview (p. 933)

Class: *wxView* (p. 793)

The *wxView* class can be used to model the viewing and editing component of an application's file-based data. It is part of the document/view framework supported by *wxWindows*, and cooperates with the *wxDocument* (p. 207), *wxDocTemplate* (p. 202) and *wxDocManager* (p. 189) classes.

See the example application in `samples/docview`.

To use the abstract *wxView* class, you need to derive a new class and override at least the member functions `OnCreate`, `OnDraw`, `OnUpdate` and `OnClose`. You'll probably want to override `OnMenuCommand` to respond to menu commands from the frame containing the view.

Use the macros `DECLARE_DYNAMIC_CLASS` and `IMPLEMENT_DYNAMIC_CLASS` in order to allow the framework to create view objects on demand. When you create a *wxDocTemplate* (p. 202) object on application initialization, you should pass `CLASSINFO(YourViewClass)` to the *wxDocTemplate* constructor so that it knows how to create an instance of this class.

If you do not wish to use the *wxWindows* method of creating view objects dynamically, you must override `wxDocTemplate::CreateView` to return an instance of the appropriate class.

7.20.3 *wxDocTemplate* overview

Document/view framework overview (p. 933)

Class: *wxDocTemplate* (p. 202)

The *wxDocTemplate* class is used to model the relationship between a document class and a view class. The application creates a document template object for each document/view pair. The list of document templates managed by the *wxDocManager* instance is used to create documents and views. Each document template knows what file filters and default extension are appropriate for a document/view combination, and how to create a document or view.

For example, you might write a small doodling application that can load and save lists of line segments. If you had two views of the data -- graphical, and a list of the segments -- then you would create one document class *DoodleDocument*, and two view classes (*DoodleGraphicView* and *DoodleListView*). You would also need two document templates, one for the graphical view and another for the list view. You would pass the same document class and default file extension to both document templates, but each would be passed a different view class. When the user clicks on the Open menu item, the file selector is displayed with a list of possible file filters -- one for each *wxDocTemplate*. Selecting the filter selects the *wxDocTemplate*, and when a file is selected, that template will be used for creating a document and view. Under non-Windows platforms, the user will be prompted for a list of templates before the file selector is shown, since most file selectors do not allow a choice of file filters.

For the case where an application has one document type and one view type, a single document template is constructed, and dialogs will be appropriately simplified.

`wxDocTemplate` is part of the document/view framework supported by `wxWindows`, and cooperates with the `wxView` (p. 793), `wxDocument` (p. 207) and `wxDocManager` (p. 189) classes.

See the example application in `samples/docview`.

To use the `wxDocTemplate` class, you do not need to derive a new class. Just pass relevant information to the constructor including `CLASSINFO(YourDocumentClass)` and `CLASSINFO(YourViewClass)` to allow dynamic instance creation. If you do not wish to use the `wxWindows` method of creating document objects dynamically, you must override `wxDocTemplate::CreateDocument` and `wxDocTemplate::CreateView` to return instances of the appropriate class.

NOTE: the document template has nothing to do with the C++ template construct. C++ templates are not used anywhere in `wxWindows`.

7.20.4 wxDocManager overview

Document/view framework overview (p. 933)

Class: `wxDocManager` (p. 189)

The `wxDocManager` class is part of the document/view framework supported by `wxWindows`, and cooperates with the `wxView` (p. 793), `wxDocument` (p. 207) and `wxDocTemplate` (p. 202) classes.

A `wxDocManager` instance coordinates documents, views and document templates. It keeps a list of document and template instances, and much functionality is routed through this object, such as providing selection and file dialogs. The application can use this class 'as is' or derive a class and override some members to extend or change the functionality. Create an instance of this class near the beginning of your application initialization, before any documents, views or templates are manipulated.

There may be multiple `wxDocManager` instances in an application.

See the example application in `samples/docview`.

7.20.5 wxCommand overview

Document/view framework overview (p. 933)

Classes: `wxCommand` (p. 101), `wxCommandProcessor` (p. 108)

`wxCommand` is a base class for modelling an application command, which is an action usually performed by selecting a menu item, pressing a toolbar button or any other

means provided by the application to change the data or view.

Instead of the application functionality being scattered around switch statements and functions in a way that may be hard to read and maintain, the functionality for a command is explicitly represented as an object which can be manipulated by a framework or application. When a user interface event occurs, the application *submits* a command to a *wxCommandProcessor* (p. 938) object to execute and store.

The wxWindows document/view framework handles Undo and Redo by use of *wxCommand* and *wxCommandProcessor* objects. You might find further uses for *wxCommand*, such as implementing a macro facility that stores, loads and replays commands.

An application can derive a new class for every command, or, more likely, use one class parameterized with an integer or string command identifier.

7.20.6 wxCommandProcessor overview

Document/view framework overview (p. 933)

Classes: *wxCommandProcessor* (p. 108), *wxCommand* (p. 101)

wxCommandProcessor is a class that maintains a history of *wxCommand* instances, with undo/redo functionality built-in. Derive a new class from this if you want different behaviour.

7.20.7 wxFileHistory overview

Document/view framework overview (p. 933)

Classes: *wxFileHistory* (p. 253), *wxDocManager* (p. 189)

wxFileHistory encapsulates functionality to record the last few files visited, and to allow the user to quickly load these files using the list appended to the File menu.

Although *wxFileHistory* is used by *wxDocManager*, it can be used independently. You may wish to derive from it to allow different behaviour, such as popping up a scrolling list of files.

By calling *wxFileHistory::FileHistoryUseMenu* you can associate a file menu with the file history, that will be used for appending the filenames. They are appended using menu identifiers in the range *wxID_FILE1* to *wxID_FILE9*.

In order to respond to a file load command from one of these identifiers, you need to handle them using an event handler, for example:

```
BEGIN_EVENT_TABLE(wxDocParentFrame, wxFrame)
    EVT_MENU(wxID_EXIT, wxDocParentFrame::OnExit)
```

```
EVT_MENU_RANGE(wxID_FILE1, wxID_FILE9, wxDocParentFrame::OnMRUFile)
END_EVENT_TABLE()

void wxDocParentFrame::OnExit(wxCommandEvent& WXUNUSED(event))
{
    Close();
}

void wxDocParentFrame::OnMRUFile(wxCommandEvent& event)
{
    wxString f(m_docManager->GetHistoryFile(event.GetSelection() -
wxID_FILE1));
    if (f != "")
        (void)m_docManager->CreateDocument(f, wxDOC_SILENT);
}
```

7.20.8 wxWindows predefined command identifiers

To allow communication between the application's menus and the document/view framework, several command identifiers are predefined for you to use in menus. The framework recognizes them and processes them if you forward commands from `wxFrame::OnMenuCommand` (or perhaps from toolbars and other user interface constructs).

- `wxID_OPEN` (5000)
- `wxID_CLOSE` (5001)
- `wxID_NEW` (5002)
- `wxID_SAVE` (5003)
- `wxID_SAVEAS` (5004)
- `wxID_REVERT` (5005)
- `wxID_EXIT` (5006)
- `wxID_UNDO` (5007)
- `wxID_REDO` (5008)
- `wxID_HELP` (5009)
- `wxID_PRINT` (5010)
- `wxID_PRINT_SETUP` (5011)
- `wxID_PREVIEW` (5012)

7.21 Event handling overview

Classes: *wxEvtHandler* (p. 224), *wxWindow* (p. 798), *wxEvent* (p. 221)

7.21.1 Introduction

Before version 2.0 of wxWindows, events were handled by the application either by supplying callback functions, or by overriding virtual member functions such as **OnSize**.

From wxWindows 2.0, *event tables* are used instead, with a few exceptions.

An event table is placed in an implementation file to tell `wxWindows` how to map events to member functions. These member functions are not virtual functions, but they all similar in form: they take a single `wxEvent`-derived argument, and have a void return type.

Here's an example of an event table.

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU      (wxID_EXIT, MyFrame::OnExit)
    EVT_MENU      (DO_TEST,   MyFrame::DoTest)
    EVT_SIZE      (           MyFrame::OnSize)
    EVT_BUTTON    (BUTTON1,   MyFrame::OnButton1)
END_EVENT_TABLE()
```

The first two entries map menu commands to two different member functions. The `EVT_SIZE` macro doesn't need a window identifier, since normally you are only interested in the current window's size events. (In fact you could intercept a particular window's size event by using `EVT_CUSTOM(wxEVT_SIZE, id, func)`.)

The `EVT_BUTTON` macro demonstrates that the originating event does not have to come from the window class implementing the event table - if the event source is a button within a panel within a frame, this will still work, because event tables are searched up through the hierarchy of windows. In this case, the button's event table will be searched, then the parent panel's, then the frame's.

As mentioned before, the member functions that handle events do not have to be virtual. Indeed, the member functions should not be virtual as the event handler ignores that the functions are virtual, i.e. overriding a virtual member function in a derived class will not have any effect. These member functions take an event argument, and the class of event differs according to the type of event and the class of the originating window. For size events, *wxSizeEvent* (p. 596) is used. For menu commands and most control commands (such as button presses), *wxCommandEvent* (p. 103) is used. When controls get more complicated, then specific event classes are used, such as *wxTreeEvent* (p. 774) for events from *wxTreeCtrl* (p. 761) windows.

As well as the event table in the implementation file, there must be a `DECLARE_EVENT_TABLE` macro in the class definition. For example:

```
class MyFrame: public wxFrame {
    DECLARE_DYNAMIC_CLASS(MyFrame)

public:
    ...
    void OnExit(wxCommandEvent& event);
    void OnSize(wxSizeEvent& event);
protected:
    int      m_count;
    ...
    DECLARE_EVENT_TABLE()
};
```

7.21.2 How events are processed

When an event is received from the windowing system, `wxWindows` calls `wxEvtHandler::ProcessEvent` (p. 227) on the first event handler object belonging to the window generating the event.

It may be noted that `wxWindows`' event processing system implements something very close to virtual methods in normal C++, i.e. it is possible to alter the behaviour of a class by overriding its event handling functions. In many cases this works even for changing the behaviour of native controls. For example it is possible to filter out a number of key events sent by the system to a native text control by overriding `wxTextCtrl` and defining a handler for key events using `EVT_KEY_DOWN`. This would indeed prevent any key events from being sent to the native control - which might not be what is desired. In this case the event handler function has to call `Skip()` so as to indicate that it did NOT handle the event at all.

In practice, this would look like this if the derived text control only accepts 'a' to 'z' and 'A' to 'Z':

```
void MyTextCtrl::OnChar(wxKeyEvent& event)
{
    if ( isalpha( event.KeyCode() ) )
    {
        // key code is within legal range. we call event.Skip() so the
        // event can be processed either in the base wxWindows class
        // or the native control.

        event.Skip();
    }
    else
    {
        // illegal key hit. we don't call event.Skip() so the
        // event is not processed anywhere else.

        wxBell();
    }
}
```

The normal order of event table searching by `ProcessEvent` is as follows:

1. If the object is disabled (via a call to `wxEvtHandler::SetEvtHandlerEnabled` (p. 229)) the function skips to step (6).
2. If the object is a `wxWindow`, **ProcessEvent** is recursively called on the window's `wxValidator` (p. 781). If this returns TRUE, the function exits.
3. **SearchEventTable** is called for this event handler. If this fails, the base class table is tried, and so on until no more tables exist or an appropriate function was found, in which case the function exits.
4. The search is applied down the entire chain of event handlers (usually the chain has a length of one). If this succeeds, the function exits.
5. If the object is a `wxWindow` and the event is a `wxCommandEvent`, **ProcessEvent** is recursively applied to the parent window's event handler. If this returns TRUE, the function exits.

6. Finally, **ProcessEvent** is called on the `wxApp` object.

Note that your application may wish to override `ProcessEvent` to redirect processing of events. This is done in the document/view framework, for example, to allow event handlers to be defined in the document or view.

As mentioned above, only command events are recursively applied to the parents event handler. As this quite often causes confusion for users, here is a list of system events which will NOT get sent to the parent's event handler:

<code>wxEvent</code> (p. 221)	The event base class
<code>wxActivateEvent</code> (p. 5)	A window or application activation event
<code>wxCloseEvent</code> (p. 84)	A close window or end session event
<code>wxEraseEvent</code> (p. 220)	An erase background event
<code>wxFocusEvent</code> (p. 263)	A window focus event
<code>wxKeyEvent</code> (p. 359)	A keypress event
<code>wxIdleEvent</code> (p. 317)	An idle event
<code>wxInitDialogEvent</code> (p. 345)	A dialog initialisation event
<code>wxJoystickEvent</code> (p. 356)	A joystick event
<code>wxMenuEvent</code> (p. 438)	A menu event
<code>wxMouseEvent</code> (p. 450)	A mouse event
<code>wxMoveEvent</code> (p. 458)	A move event
<code>wxPaintEvent</code> (p. 484)	A paint event
<code>wxQueryLayoutInfoEvent</code> (p. 535)	Used to query layout information
<code>wxSizeEvent</code> (p. 596)	A size event
<code>wxSysColourChangedEvent</code> (p. 678)	A system colour change event
<code>wxUpdateUIEvent</code> (p. 775)	A user interface update event

In some cases, it might be desired by the programmer to get a certain number of system events in a parent window, for example all key events sent to, but not used by, the native controls in a dialog. In this case, a special event handler will have to be written that will override `ProcessEvent()` in order to pass all events (or any selection of them) to the parent window. See next section.

7.21.3 Pluggable event handlers

In fact, you don't have to derive a new class from a window class if you don't want to. You can derive a new class from `wxEvtHandler` instead, defining the appropriate event table, and then call `wxWindow::SetEventHandler` (p. 834) (or, preferably, `wxWindow::PushEventHandler` (p. 829)) to make this event handler the object that responds to events. This way, you can avoid a lot of class derivation, and use the same event handler object to handle events from instances of different classes. If you ever have to call a window's event handler manually, use the `GetEventHandler` function to retrieve the window's event handler and use that to call the member function. By default, `GetEventHandler` returns a pointer to the window itself unless an application has redirected event handling using `SetEventHandler` or `PushEventHandler`.

One use of `PushEventHandler` is to temporarily or permanently change the behaviour of the GUI. For example, you might want to invoke a dialog editor in your application that

changes aspects of dialog boxes. You can grab all the input for an existing dialog box, and edit it 'in situ', before restoring its behaviour to normal. So even if the application has derived new classes to customize behaviour, your utility can indulge in a spot of body-snatching. It could be a useful technique for on-line tutorials, too, where you take a user through a series of steps and don't want them to diverge from the lesson. Here, you can examine the events coming from buttons and windows, and if acceptable, pass them through to the original event handler. Use `PushEventHandler/PopEventHandler` to form a chain of event handlers, where each handler processes a different range of events independently from the other handlers.

7.21.4 Window identifiers

Window identifiers are integers, and are used to uniquely determine window identity in the event system (though you can use it for other purposes). In fact, identifiers do not need to be unique across your entire application just so long as they are unique within a particular context you're interested in, such as a frame and its children. You may use the `wxID_OK` identifier, for example, on any number of dialogs so long as you don't have several within the same dialog.

If you pass -1 to a window constructor, an identifier will be generated for you, but beware: if things don't respond in the way they should, it could be because of an id conflict. It's safer to supply window ids at all times. Automatic generation of identifiers starts at 1 so may well conflict with your own identifiers.

The following standard identifiers are supplied. You can use `wxID_HIGHEST` to determine the number above which it is safe to define your own identifiers. Or, you can use identifiers below `wxID_LOWEST`.

```
#define wxID_LOWEST          4999

#define wxID_OPEN            5000
#define wxID_CLOSE          5001
#define wxID_NEW             5002
#define wxID_SAVE            5003
#define wxID_SAVEAS          5004
#define wxID_REVERT          5005
#define wxID_EXIT            5006
#define wxID_UNDO            5007
#define wxID_REDO            5008
#define wxID_HELP            5009
#define wxID_PRINT           5010
#define wxID_PRINT_SETUP     5011
#define wxID_PREVIEW         5012
#define wxID_ABOUT           5013
#define wxID_HELP_CONTENTS   5014
#define wxID_HELP_COMMANDS   5015
#define wxID_HELP_PROCEDURES 5016
#define wxID_HELP_CONTEXT    5017

#define wxID_CUT              5030
#define wxID_COPY             5031
#define wxID_PASTE           5032
#define wxID_CLEAR           5033
#define wxID_FIND            5034
```

```
#define wxID_DUPLICATE          5035
#define wxID_SELECTALL          5036

#define wxID_FILE1              5050
#define wxID_FILE2              5051
#define wxID_FILE3              5052
#define wxID_FILE4              5053
#define wxID_FILE5              5054
#define wxID_FILE6              5055
#define wxID_FILE7              5056
#define wxID_FILE8              5057
#define wxID_FILE9              5058

#define wxID_OK                  5100
#define wxID_CANCEL              5101
#define wxID_APPLY               5102
#define wxID_YES                 5103
#define wxID_NO                  5104
#define wxID_STATIC              5105

#define wxID_HIGHEST             5999
```

7.21.5 Event macros summary

Generic event table macros

EVT_CUSTOM(event, id, func)	Allows you to add a custom event table entry by specifying the event identifier (such as <code>wxEVT_SIZE</code>), the window identifier, and a member function to call.
EVT_CUSTOM_RANGE(event, id1, id2, func)	The same as <code>EVT_CUSTOM</code> , but responds to a range of window identifiers.
EVT_COMMAND(id, event, func)	The same as <code>EVT_CUSTOM</code> , but expects a member function with a <code>wxCommandEvent</code> argument.
EVT_COMMAND_RANGE(id1, id2, event, func)	The same as <code>EVT_CUSTOM_RANGE</code> , but expects a member function with a <code>wxCommandEvent</code> argument.

Macros listed by event class

The documentation for specific event macros is organised by event class. Please refer to these sections for details.

<i>wxActivateEvent</i> (p. 5)	The <code>EVT_ACTIVATE</code> and <code>EVT_ACTIVATE_APP</code> macros intercept activation and deactivation events.
<i>wxCommandEvent</i> (p. 103)	A range of commonly-used control events.
<i>wxCloseEvent</i> (p. 84)	The <code>EVT_CLOSE</code> macro handles window

<i>wxDropFilesEvent</i> (p. 214)	closure called via <i>wxWindow::Close</i> (p. 802).
<i>wxEraseEvent</i> (p. 220)	The <code>EVT_DROP_FILES</code> macros handles file drop events.
<i>wxFocusEvent</i> (p. 263)	The <code>EVT_ERASE_BACKGROUND</code> macro is used to handle window erase requests.
<i>wxKeyEvent</i> (p. 359)	The <code>EVT_SET_FOCUS</code> and <code>EVT_KILL_FOCUS</code> macros are used to handle keyboard focus events.
<i>wxIdleEvent</i> (p. 317)	<code>EVT_CHAR</code> and <code>EVT_CHAR_HOOK</code> macros handle keyboard input for any window.
<i>wxInitDialogEvent</i> (p. 345)	The <code>EVT_IDLE</code> macro handle application idle events (to process background tasks, for example).
<i>wxListEvent</i> (p. 394)	The <code>EVT_INIT_DIALOG</code> macro is used to handle dialog initialisation.
<i>wxMenuEvent</i> (p. 438)	These macros handle <i>wxListCtrl</i> (p. 381) events.
<i>wxMouseEvent</i> (p. 450)	These macros handle special menu events (not menu commands).
<i>wxMoveEvent</i> (p. 458)	Mouse event macros can handle either individual mouse events or all mouse events.
<i>wxUpdateUIEvent</i> (p. 775)	The <code>EVT_MOVE</code> macro is used to handle a window move.
<i>wxPaintEvent</i> (p. 484)	The <code>EVT_UPDATE_UI</code> macro is used to handle user interface update pseudo-events, which are generated to give the application the chance to update the visual state of menus, toolbars and controls.
<i>wxScrollEvent</i> (p. 584)	The <code>EVT_PAINT</code> macro is used to handle window paint requests.
<i>wxSizeEvent</i> (p. 596)	These macros are used to handle scroll events from windows, <i>wxScrollBar</i> (p. 579), and <i>wxSpinButton</i> (p. 622).
<i>wxSysColourChangedEvent</i> (p. 678)	The <code>EVT_SIZE</code> macro is used to handle a window resize.
<i>wxTreeEvent</i> (p. 774)	The <code>EVT_SYS_COLOUR_CHANGED</code> macro is used to handle events informing the application that the user has changed the system colours (Windows only).
	These macros handle <i>wxTreeCtrl</i> (p. 761) events.

7.22 Writing a wxWindows application: a rough guide

To set a wxWindows application going, you'll need to derive a *wxApp* (p. 6) class and

override `wxApp::OnInit` (p. 12).

An application must have a top-level `wxFrame` (p. 275) or `wxDialog` (p. 178) window. Each frame may contain one or more instances of classes such as `wxPanel` (p. 488), `wxSplitterWindow` (p. 626) or other windows and controls.

A frame can have a `wxMenuBar` (p. 426), a `wxToolBar` (p. 746), a status line, and a `wxIcon` (p. 318) for when the frame is iconized.

A `wxPanel` (p. 488) is used to place controls (classes derived from `wxControl` (p. 125)) which are used for user interaction. Examples of controls are `wxButton` (p. 36), `wxCheckBox` (p. 70), `wxChoice` (p. 75), `wxListBox` (p. 373), `wxRadioBox` (p. 537), `wxSlider` (p. 597).

Instances of `wxDialog` (p. 178) can also be used for controls and they have the advantage of not requiring a separate frame.

Instead of creating a dialog box and populating it with items, it is possible to choose one of the convenient common dialog classes, such as `wxMessageDialog` (p. 439) and `wxFileDialog` (p. 248).

You never draw directly onto a window - you use a *device context* (DC). `wxDC` (p. 151) is the base for `wxClientDC` (p. 81), `wxPaintDC` (p. 483), `wxMemoryDC` (p. 415), `wxPostScriptDC` (p. 504), `wxMemoryDC` (p. 415), `wxMetafileDC` (p. 442) and `wxPrinterDC` (p. 515). If your drawing functions have **wxDC** as a parameter, you can pass any of these DCs to the function, and thus use the same code to draw to several different devices. You can draw using the member functions of **wxDC**, such as `wxDC::DrawLine` (p. 155) and `wxDC::DrawText` (p. 157). Control colour on a window (`wxColour` (p. 86)) with brushes (`wxBrush` (p. 61)) and pens (`wxPen` (p. 494)).

To intercept events, you add a `DECLARE_EVENT_TABLE` macro to the window class declaration, and put a `BEGIN_EVENT_TABLE ... END_EVENT_TABLE` block in the implementation file. Between these macros, you add event macros which map the event (such as a mouse click) to a member function. These might override predefined event handlers such as `wxWindow::OnChar` (p. 817) and `wxWindow::OnMouseEvent` (p. 824).

Most modern applications will have an on-line, hypertext help system; for this, you need `wxHelp` and the `wxHelpController` (p. 313) class to control `wxHelp`.

GUI applications aren't all graphical wizardry. List and hash table needs are catered for by `wxList` (p. 367), `wxStringList` (p. 674) and `wxHashTable` (p. 310). You will undoubtedly need some platform-independent *file functions* (p. 845), and you may find it handy to maintain and search a list of paths using `wxPathList` (p. 492). There's a *miscellany* (p. 861) of operating system and other functions.

See also *Classes by Category* (p. 890) for a list of classes.

7.23 Interprocess communication overview

Classes: `wxDDEServer` (p. 172), `wxDDEConnection` (p. 167), `wxDDEClient` (p. 166), `wxTCPServer` (p. 709), `wxTCPConnection` (p. 705), `wxTCPClient` (p. 703)

`wxWindows` has a number of different classes to help with interprocess communication and network programming. This section only discusses one family of classes - the DDE-like protocol - but here's a list of other useful classes:

- `wxSocketEvent` (p. 617), `wxSocketBase` (p. 607), `wxSocketClient` (p. 615), `wxSocketServer` (p. 620): classes for the low-level TCP/IP API.
- `wxProtocol` (p. 528), `wxURL` (p. 779), `wxFTP` (p. 287), `wxHTTP`: classes for programming popular Internet protocols.

Further information on these classes will be available in due course.

`wxWindows` has a high-level protocol based on Windows DDE. There are two implementations of this DDE-like protocol: one using real DDE running on Windows only, and another using TCP/IP (sockets) that runs on most platforms. Since the API is the same apart from the names of the classes, you should find it easy to switch between the two implementations.

The following description refers to 'DDE' but remember that the equivalent `wxTCP...` classes can be used in much the same way.

Three classes are central to the DDE API:

1. `wxDDEClient`. This represents the client application, and is used only within a client program.
2. `wxDDEServer`. This represents the server application, and is used only within a server program.
3. `wxDDEConnection`. This represents the connection from the current client or server to the other application (server or client), and can be used in both server and client programs. Most DDE transactions operate on this object.

Messages between applications are usually identified by three variables: connection object, topic name and item name. A data string is a fourth element of some messages. To create a connection (a conversation in Windows parlance), the client application sends the message `MakeConnection` to the client object, with a string service name to identify the server and a topic name to identify the topic for the duration of the connection. Under Unix, the service name must contain an integer port identifier.

The server then responds and either vetos the connection or allows it. If allowed, a connection object is created which persists until the connection is closed. The connection object is then used for subsequent messages between client and server.

To create a working server, the programmer must:

1. Derive a class from `wxDDEServer`.
2. Override the handler `OnAcceptConnection` for accepting or rejecting a connection, on the basis of the topic argument. This member must create and return a connection object if the connection is accepted.
3. Create an instance of your server object, and call `Create` to activate it, giving it a service name.
4. Derive a class from `wxDDEConnection`.

5. Provide handlers for various messages that are sent to the server side of a `wxDDEConnection`.

To create a working client, the programmer must:

1. Derive a class from `wxDDEClient`.
2. Override the handler `OnMakeConnection` to create and return an appropriate connection object.
3. Create an instance of your client object.
4. Derive a class from `wxDDEConnection`.
5. Provide handlers for various messages that are sent to the client side of a `wxDDEConnection`.
6. When appropriate, create a new connection by sending a `MakeConnection` message to the client object, with arguments host name (processed in Unix only), service name, and topic name for this connection. The client object will call `OnMakeConnection` to create a connection object of the desired type.
7. Use the `wxDDEConnection` member functions to send messages to the server.

7.23.1 Data transfer

These are the ways that data can be transferred from one application to another.

- **Execute:** the client calls the server with a data string representing a command to be executed. This succeeds or fails, depending on the server's willingness to answer. If the client wants to find the result of the `Execute` command other than success or failure, it has to explicitly call `Request`.
- **Request:** the client asks the server for a particular data string associated with a given item string. If the server is unwilling to reply, the return value is `NULL`. Otherwise, the return value is a string (actually a pointer to the connection buffer, so it should not be deallocated by the application).
- **Poke:** The client sends a data string associated with an item string directly to the server. This succeeds or fails.
- **Advise:** The client asks to be advised of any change in data associated with a particular item. If the server agrees, the server will send an `OnAdvise` message to the client along with the item and data.

The default data type is `wxCF_TEXT` (ASCII text), and the default data size is the length of the null-terminated string. Windows-specific data types could also be used on the PC.

7.23.2 Examples

See the sample programs *server* and *client* in the IPC samples directory. Run the server, then the client. This demonstrates using the `Execute`, `Request`, and `Poke` commands from the client, together with an `Advise` loop: selecting an item in the server list box causes that item to be highlighted in the client list box.

7.23.3 More DDE details

A `wxDDEClient` object represents the client part of a client-server DDE (Dynamic Data Exchange) conversation (available in both Windows and Unix).

To create a client which can communicate with a suitable server, you need to derive a class from `wxDDEConnection` and another from `wxDDEClient`. The custom `wxDDEConnection` class will intercept communications in a 'conversation' with a server, and the custom `wxDDEServer` is required so that a user-overridden `wxDDEClient::OnMakeConnection` (p. 167) member can return a `wxDDEConnection` of the required class, when a connection is made.

For example:

```
class MyConnection: public wxDDEConnection
{
public:
    MyConnection(void)::wxDDEConnection(ipc_buffer, 3999) {}
    ~MyConnection(void) {}
    bool OnAdvise(const wxString& topic, const wxString& item, char *data,
int size, wxIPCFFormat format)
    { wxMessageBox(topic, data); }
};

class MyClient: public wxDDEClient
{
public:
    MyClient(void) {}
    wxConnectionBase *OnMakeConnection(void) { return new MyConnection; }
};
```

Here, **MyConnection** will respond to *OnAdvise* (p. 170) messages sent by the server.

When the client application starts, it must create an instance of the derived `wxDDEClient`. In the following, command line arguments are used to pass the host name (the name of the machine the server is running on) and the server name (identifying the server process). Calling `wxDDEClient::MakeConnection` (p. 167) implicitly creates an instance of **MyConnection** if the request for a connection is accepted, and the client then requests an *Advise* loop from the server, where the server calls the client when data has changed.

```
wxString server = "4242";
wxString hostName;
wxGetHostName(hostName);

// Create a new client
MyClient *client = new MyClient;
connection = (MyConnection *)client->MakeConnection(hostName, server,
"IPC TEST");

if (!connection)
{
    wxMessageBox("Failed to make connection to server", "Client Demo
Error");
    return NULL;
}
```

```
connection->StartAdvise("Item");
```

Note that it is no longer necessary to call `wxDDEInitialize` or `wxDDECleanUp`, since `wxWindows` will do this itself if necessary.

7.24 Printing overview

Classes: *wxPrintout* (p. 516), *wxPrinter* (p. 513), *wxPrintPreview* (p. 519), *wxPrinterDC* (p. 515), *wxPrintDialog* (p. 512).

The printing framework relies on the application to provide classes whose member functions can respond to particular requests, such as 'print this page' or 'does this page exist in the document?'. This method allows `wxWindows` to take over the housekeeping duties of turning preview pages, calling the print dialog box, creating the printer device context, and so on: the application can concentrate on the rendering of the information onto a device context. The printing framework is mainly a Windows feature; PostScript support under non-Windows platforms is emerging but has not been rigorously tested.

The *document/view framework* (p. 933) creates a default `wxPrintout` object for every view, calling `wxView::OnDraw` to achieve a prepackaged print/preview facility.

A document's printing ability is represented in an application by a derived `wxPrintout` class. This class prints a page on request, and can be passed to the `Print` function of a `wxPrinter` object to actually print the document, or can be passed to a `wxPrintPreview` object to initiate previewing. The following code (from the printing sample) shows how easy it is to initiate printing, previewing and the print setup dialog, once the `wxPrintout` functionality has been defined. Notice the use of `MyPrintout` for both printing and previewing. All the preview user interface functionality is taken care of by `wxWindows`. For details on how `MyPrintout` is defined, please look at the printout sample code.

```
case WXPRINT_PRINT:
{
    wxPrinter printer;
    MyPrintout printout("My printout");
    printer.Print(this, &printout, TRUE);
    break;
}
case WXPRINT_PREVIEW:
{
    // Pass two printout objects: for preview, and possible printing.
    wxPrintPreview *preview = new wxPrintPreview(new MyPrintout, new
MyPrintout);
    wxPreviewFrame *frame = new wxPreviewFrame(preview, this, "Demo
Print Preview", 100, 100, 600, 650);
    frame->Centre(wxBOTH);
    frame->Initialize();
    frame->Show(TRUE);
    break;
}
case WXPRINT_PRINT_SETUP:
{
    wxPrintDialog printerDialog(this);
    printerDialog.GetPrintData().SetSetupDialog(TRUE);
    printerDialog.Show(TRUE);
    break;
}
```

```
}
7.25
```

The wxWindows resource system

From version 1.61, wxWindows has an optional *resource file* facility, which allows separation of dialog, menu, bitmap and icon specifications from the application code.

It is similar in principle to the Windows resource file (whose ASCII form is suffixed .RC and whose binary form is suffixed .RES). The wxWindows resource file is currently ASCII-only, suffixed .WXR. Note that under Windows, the .WXR file does not *replace* the native Windows resource file, it merely supplements it. There is no existing native resource format in X (except for the defaults file, which has limited expressive power).

Using wxWindows resources for panels and dialogs has an effect on how you deal with panel item callbacks: you can't specify a callback function in a resource file, so how do you achieve the same effect as with programmatic panel construction? The solution is similar to that adopted by Windows, which is to use the *parent* panel or dialog to intercept user events.

From 1.61, wxWindows routes panel item events that do not have a callback to the *OnCommand* (p. 818) member of the panel (or dialog). So, to use panel or dialog resources, you need to derive a new class and override the default (empty) *OnCommand* member. The first argument is a reference to a wxWindow, and the second is a reference to a wxCommandEvent. Check the name of the panel item that's generating an event by using the *wxWindow::GetName* (p. 810) function and a string comparison function such as *wxStringEq* (p. 851). You may need to cast the reference to an appropriate specific type to perform some operations.

To obtain a pointer to a panel item when you only have the name (for example, when you need to set a value of a text item from outside of the **OnCommand** function), use the function *wxFindWindowByName* (p. 867).

For details of functions for manipulating resource files and loading user interface elements, see *wxWindows resource functions* (p. 881).

7.25.1 The format of a .WXR file

A wxWindows resource file may look a little odd at first. It's C++ compatible, comprising mostly of static string variable declarations with PrologIO syntax within the string.

Here's a sample .WXR file:

```
/*
 * wxWindows Resource File
 * Written by wxBuilder
 *
 */

#include "noname.ids"

static char *aiai_resource = "bitmap(name = 'aiai_resource', \
```

```

    bitmap = ['ai'ai', wxBITMAP_TYPE_BMP_RESOURCE, 'WINDOWS'],\
    bitmap = ['ai'ai.xpm', wxBITMAP_TYPE_XPM, 'X'])).";

static char *menuBar11 = "menu(name = 'menuBar11',\
    menu = \
    [\
        ['&File', 1, '', \
            ['&Open File', 2, 'Open a file'],\
            ['&Save File', 3, 'Save a file'],\
            [],\
            ['E&xit', 4, 'Exit program']\
        ],\
        ['&Help', 5, '', \
            ['&About', 6, 'About this program']\
        ]\
    ])).";

static char *project_resource = "icon(name = 'project_resource',\
    icon = ['project', wxBITMAP_TYPE_ICO_RESOURCE, 'WINDOWS'],\
    icon = ['project_data', wxBITMAP_TYPE_XBM, 'X'])).";

static char *panel3 = "dialog(name = 'panel3',\
    style = '',\
    title = 'untitled',\
    button_font = [14, 'wxSWISS', 'wxNORMAL', 'wxBOLD', 0],\
    label_font = [10, 'wxSWISS', 'wxNORMAL', 'wxNORMAL', 0],\
    x = 0, y = 37, width = 292, height = 164,\
    control = [wxButton, 'OK', '', 'button5', 23, 34, -1, -1,\
        'ai'ai_resource'],\
    control = [wxMessage, 'A Label', '', 'message7', 166, 61, -1, -1,\
        'ai'ai_resource'],\
    control = [wxText, 'Text', 'wxVERTICAL_LABEL', 'text8', 24, 110, -1, -\
1])).";

```

As you can see, C++-style comments are allowed, and apparently include files are supported too: but this is a special case, where the included file is a file of defines shared by the C++ application code and resource file to relate identifiers (such as `FILE_OPEN`) to integers.

Each *resource object* is of standard PrologIO syntax, that is, an object name such as **dialog** or **icon**, then an open parenthesis, a list of comma-delimited attribute/value pairs, a closing parenthesis, and a full stop. Backslashes are required to escape newlines, for the benefit of C++ syntax. If double quotation marks are used to delimit strings, they need to be escaped with backslash within a C++ string (so it's easier to use single quotation marks instead).

A note on PrologIO string syntax: A string that begins with an alphabetic character, and contains only alphanumeric characters, hyphens and underscores, need not be quoted at all. Single quotes and double quotes may be used to delimit more complex strings. In fact, single-quoted and no-quoted strings are actually called *words*, but are treated as strings for the purpose of the resource system.

A resource file like this is typically included in the application main file, as if it were a normal C++ file. This eliminates the need for a separate resource file to be distributed alongside the executable. However, the resource file can be dynamically loaded if desired (for example by a non-C++ language such as CLIPS, Prolog or Python).

Once included, the resources need to be 'parsed' (interpreted), because so far the data is just a number of static string variables. The function `::wxResourceParseData` is called early on in initialization of the application (usually in `wxApp::OnInit`) with a variable as argument. This may need to be called a number of times, one for each variable. However, more than one resource 'object' can be stored in one string variable at a time, so you can get all your resources into one variable if you want to.

`::wxResourceParseData` parses the contents of the resource, ready for use by functions such as `::wxResourceCreateBitmap` and `wxPanel::LoadFromResource`.

If a `wxWindows` resource object (such as a bitmap resource) refers to a C++ data structure, such as static XBM or XPM data, a further call (`::wxResourceRegisterBitmapData`) needs to be made on initialization to tell `wxWindows` about this data. The `wxWindows` resource object will refer to a string identifier, such as 'project_data' in the example file above. This identifier will be looked up in a table to get the C++ static data to use for the bitmap or icon.

In the C++ fragment below, the WXR resource file is included, and appropriate resource initialization is carried out in `OnInit`. Note that at this stage, no actual `wxWindows` dialogs, menus, bitmaps or icons are created; their 'templates' are merely being set up for later use.

```
/*
 * File:      noname.cc
 * Purpose:   main application module, generated by wxBuilder.
 */

#include "wx.h"
#include "wx_help.h"
#include "noname.h"

// Includes the dialog, menu etc. resources
#include "noname.wxr"

// Includes XBM data
#include "project.xbm"

// Declare an instance of the application: allows the program to start
AppClass theApp;

// Called to initialize the program
wxFrame *AppClass::OnInit(void)
{
#ifdef wx_x
    wxResourceRegisterBitmapData("project_data", project_bits,
project_width, project_height);
#endif
    wxResourceParseData(menuBar11);
    wxResourceParseData(aiai_resource);
    wxResourceParseData(project_resource);
    wxResourceParseData(panel3);
    ...
}
```

7.25.2 Dialog resource format

A dialog resource object may be used for either panels or dialog boxes, and consists of the following attributes. In the following, a *font specification* is a list consisting of point size, family, style, weight, underlined, optional facename.

Attribute	Value
name	The name of the resource.
style	Optional dialog box or panel window style.
title	The title of the dialog box (unused if a panel).
.modal	Whether modal: 1 if modal, 0 if modeless, absent if a panel resource.
button_font	The font used for control buttons: a list comprising point size (integer), family (string), font style (string), font weight (string) and underlining (0 or 1).
label_font	The font used for control labels: a list comprising point size (integer), family (string), font style (string), font weight (string) and underlining (0 or 1).
x	The x position of the dialog or panel.
y	The y position of the dialog or panel.
width	The width of the dialog or panel.
height	The height of the dialog or panel.
background_colour	The background colour of the dialog or panel. Only valid if the style includes wxUSER_COLOURS.
label_colour	The default label colour for the children of the dialog or panel. Only valid if the style includes wxUSER_COLOURS.
button_colour	The default button text colour for the children of the dialog or panel. Only valid if the style includes wxUSER_COLOURS.
label_font	Font spec
button_font	Font spec

Then comes zero or more attributes named 'control' for each control (panel item) on the dialog or panel. The value is a list of further elements. In the table below, the names in the first column correspond to the first element of the value list, and the second column details the remaining elements of the list.

Control	Values
wxButton	title (string), window style (string), name (string), x, y, width, height, button bitmap resource (optional string), button font spec

wxCheckBox	title (string), window style (string), name (string), x, y, width, height, default value (optional integer, 1 or 0), label font spec
wxChoice	title (string), window style (string), name (string), x, y, width, height, values (optional list of strings), label font spec, button font spec
wxComboBox	title (string), window style (string), name (string), x, y, width, height, default text value, values (optional list of strings), label font spec, button font spec
wxGauge	title (string), window style (string), name (string), x, y, width, height, value (optional integer), range (optional integer), label font spec, button font spec
wxGroupBox	title (string), window style (string), name (string), x, y, width, height, label font spec
wxListBox	title (string), window style (string), name (string), x, y, width, height, values (optional list of strings), multiple (optional string, wxSINGLE or wxMULTIPLE), label font spec, button font spec
wxMessage	title (string), window style (string), name (string), x, y, width, height, message bitmap resource (optional string), label font spec
wxMultiText	title (string), window style (string), name (string), x, y, width, height, default value (optional string), label font spec, button font spec
wxRadioBox	title (string), window style (string), name (string), x, y, width, height, values (optional list of strings), number of rows or cols, label font spec, button font spec
wxRadioButton	title (string), window style (string), name (string), x, y, width, height, default value (optional integer, 1 or 0), label font spec
wxScrollBar	title (string), window style (string), name (string), x, y, width, height, value (optional integer), page length (optional integer), object length (optional integer), view length (optional integer)
wxSlider	title (string), window style (string), name (string), x, y, width, height, value (optional integer), minimum (optional integer), maximum (optional integer), label font spec, button font spec
wxText	title (string), window style (string), name (string), x, y, width, height, default value (optional string), label font spec, button font spec

7.25.3 Menubar resource format

A menubar resource object consists of the following attributes.

Attribute	Value
name	The name of the menubar resource.
menu	A list containing all the menus, as detailed below.

The value of the **menu** attribute is a list of menu item specifications, where each menu item specification is itself a list comprising:

- title (a string)
- menu item identifier (a string or non-zero integer, see below)
- help string (optional)
- 0 or 1 for the 'checkable' parameter (optional)
- optionally, further menu item specifications if this item is a pulldown menu.

If the menu item specification is the empty list ([]), this is interpreted as a menu separator.

If further (optional) information is associated with each menu item in a future release of wxWindows, it will be placed after the help string and before the optional pulldown menu specifications.

Note that the menu item identifier must be an integer if the resource is being included as C++ code and then parsed on initialisation. Unfortunately, #define substitution is not performed inside strings, and therefore the program cannot know the mapping. However, if the .WXR file is being loaded dynamically, wxWindows will attempt to replace string identifiers with #defined integers, because it is able to parse the included #defines.

7.25.4 Bitmap resource format

A bitmap resource object consists of a name attribute, and one or more **bitmap** attributes. There can be more than one of these to allow specification of bitmaps that are optimum for the platform and display.

- Bitmap name or filename.
- Type of bitmap; for example, wxBITMAP_TYPE_BMP_RESOURCE. See class reference under **wxBitmap** for a full list).
- Platform this bitmap is valid for; one of WINDOWS, X, MAC and ANY.
- Number of colours (optional).
- X resolution (optional).
- Y resolution (optional).

7.25.5 Icon resource format

An icon resource object consists of a name attribute, and one or more **icon** attributes. There can be more than one of these to allow specification of icons that are optimum for the platform and display.

- Icon name or filename.
- Type of icon; for example, `wxBITMAP_TYPE_ICO_RESOURCE`. See class reference under **wxBitmap** for a full list).
- Platform this bitmap is valid for; one of `WINDOWS`, `X`, `MAC` and `ANY`.
- Number of colours (optional).
- X resolution (optional).
- Y resolution (optional).

7.25.6 Resource format design issues

The `.WXR` file format is a recent addition and subject to change. The use of an ASCII resource file format may seem rather inefficient, but this choice has a number of advantages:

- Since it is C++ compatible, it can be included into an application's source code, eliminating the problems associated with distributing a separate resource file with the executable. However, it can also be loaded dynamically from a file, which will be required for non-C++ programs that use `wxWindows`.
- No extra binary file format and separate converter need be maintained for the `wxWindows` project (although others are welcome to add the equivalent of the Windows 'rc' resource parser and a binary format).
- It would be difficult to append a binary resource component onto an executable in a portable way.
- The file format is essentially the PrologIO object format, for which a parser already exists, so parsing is easy. For those programs that use PrologIO anyway, the size overhead of the parser is minimal.

The disadvantages of the approach include:

- Parsing adds a small execution overhead to program initialization.
- Under 16-bit Windows especially, global data is at a premium. Using a `.RC` resource table for some `wxWindows` resource data may be a partial solution, although `.RC` strings are limited to 255 characters.
- Without a resource preprocessor, it is not possible to substitute integers for identifiers (so menu identifiers have to be written as integers in the resource object, in addition to providing `#defines` for application code convenience).

7.25.7 Compiling the resource system

To enable the resource system, set **wxUSE_WX_RESOURCES** to 1 in `setup.h`. If your

wxWindows makefile supports it, set the same name in the makefile to 1.

7.26 Run time class information overview

Classes: *wxObject* (p. 471), *wxClassInfo* (p. 79).

One of the failings of C++ is that no run-time information is provided about a class and its position in the inheritance hierarchy. Another is that instances of a class cannot be created just by knowing the name of a class, which makes facilities such as persistent storage hard to implement.

Most C++ GUI frameworks overcome these limitations by means of a set of macros and functions and wxWindows is no exception. Each class that you wish to be known the type system should have a macro such as `DECLARE_DYNAMIC_CLASS` just inside the class declaration. The macro `IMPLEMENT_DYNAMIC_CLASS` should be in the implementation file. Note that these are entirely optional; use them if you wish to check object types, or create instances of classes using the class name. However, it is good to get into the habit of adding these macros for all classes.

Variations on these *macros* (p. 876) are used for multiple inheritance, and abstract classes that cannot be instantiated dynamically or otherwise.

`DECLARE_DYNAMIC_CLASS` inserts a static *wxClassInfo* declaration into the class, initialized by `IMPLEMENT_DYNAMIC_CLASS`. When initialized, the *wxClassInfo* object inserts itself into a linked list (accessed through *wxClassInfo::first* and *wxClassInfo::next* pointers). The linked list is fully created by the time all global initialisation is done.

`IMPLEMENT_DYNAMIC_CLASS` is a macro that not only initialises the static *wxClassInfo* member, but defines a global function capable of creating a dynamic object of the class in question. A pointer to this function is stored in *wxClassInfo*, and is used when an object should be created dynamically.

wxObject::IsKindOf uses the linked list of *wxClassInfo*. It takes a *wxClassInfo* argument, so use `CLASSINFO(className)` to return an appropriate *wxClassInfo* pointer to use in this function.

The function *wxCreateDynamicObject* (p. 862) can be used to construct a new object of a given type, by supplying a string name. If you have a pointer to the *wxClassInfo* object instead, then you can simply call *wxClassInfo::CreateObject*.

7.26.1 wxClassInfo

Run time class information overview (p. 958)

Class: *wxClassInfo* (p. 79)

This class stores meta-information about classes. An application may use macros such as `DECLARE_DYNAMIC_CLASS` and `IMPLEMENT_DYNAMIC_CLASS` to record run-time information about a class, including:

- its position in the inheritance hierarchy;
- the base class name(s) (up to two base classes are permitted);
- a string representation of the class name;
- a function that can be called to construct an instance of this class.

The `DECLARE_...` macros declare a static `wxClassInfo` variable in a class, which is initialized by macros of the form `IMPLEMENT_...` in the implementation C++ file. Classes whose instances may be constructed dynamically are given a global constructor function which returns a new object.

You can get the `wxClassInfo` for a class by using the `CLASSINFO` macro, e.g. `CLASSINFO(wxFrame)`. You can get the `wxClassInfo` for an object using `wxObject::GetClassInfo`.

See also *wxObject* (p. 471) and *wxCreateDynamicObject* (p. 862).

7.26.2 Example

In a header file `wx_frame.h`:

```
class wxFrame: public wxWindow
{
    DECLARE_DYNAMIC_CLASS(wxFrame)

private:
    char *frameTitle;
public:
    ...
};
```

In a C++ file `wx_frame.cc`:

```
IMPLEMENT_DYNAMIC_CLASS(wxFrame, wxWindow)

wxFrame::wxFrame(void)
{
    ...
}
```

7.27

Window styles

Window styles are used to specify alternative behaviour and appearances for windows, when they are created. The symbols are defined in such a way that they can be combined in a 'bit-list' using the C++ *bitwise-or* operator. For example:

```
wxCAPTION | wxMINIMIZE_BOX | wxMAXIMIZE_BOX | wxTHICK_FRAME
```

For the window styles specific to each window class, please see the documentation for the window. Most windows can use the generic styles listed for *wxWindow* (p. 798) in addition to their own styles.

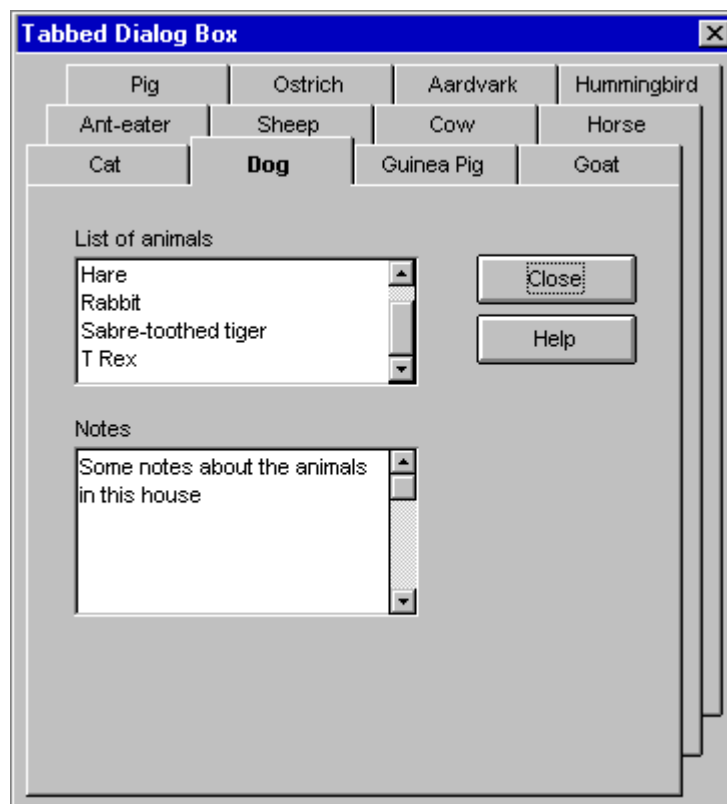
7.28 Tab classes overview

Classes: *wxTabView* (p. 687), *wxPanelTabView* (p. 491), *wxTabbedPanel* (p. 683), *wxTabbedDialog* (p. 682), *wxTabControl* (p. 684)

The tab classes provides a way to display rows of tabs (like file divider tabs), which can be used to switch between panels or other information. Tabs are most commonly used in dialog boxes where the number of options is too great to fit on one dialog.

The appearance and behaviour of a *wxTabbedDialog*

The following screenshot shows the appearance of the sample tabbed dialog application.



By clicking on the tabs, the user can display a different set of controls. In the example, the Close and Help buttons remain constant. These two buttons are children of the main dialog box, whereas the other controls are children of panels which are shown and hidden according to which tab is active.

A tabbed dialog may have several layers (rows) of tabs, each being offset vertically and horizontally from the previous. Tabs work in columns, in that when a tab is pressed, it swaps place with the tab on the first row of the same column, in order to give the effect of displaying that tab. All tabs must be of the same width. This is a constraint of the implementation, but it also means that the user will find it easier to find tabs since there are distinct tab columns. On some tabbed dialog implementations, tabs jump around seemingly randomly because tabs have different widths. In this implementation, a tab

can always be found on the same column.

Tabs are always drawn along the top of the view area; the implementation does not allow for vertical tabs or any other configuration.

Using tabs

The tab classes provide facilities for switching between contexts by means of 'tabs', which look like file divider tabs.

You must create both a *view* to handle the tabs, and a *window* to display the tabs and related information. The `wxTabbedDialog` and `wxTabbedPanel` classes are provided for convenience, but you could equally well construct your own window class and derived tab view.

If you wish to display a tabbed dialog - the most common use - you should follow these steps.

1. Create a new `wxTabbedDialog` class, and any buttons you wish always to be displayed (regardless of which tab is active).
2. Create a new `wxPanelTabView`, passing the dialog as the first argument.
3. Set the view rectangle with `wxTabView::SetViewRect` (p. 695), to specify the area in which child panels will be shown. The tabs will sit on top of this view rectangle.
4. Call `wxTabView::CalculateTabWidth` (p. 688) to calculate the width of the tabs based on the view area. This is optional if, for example, you have one row of tabs which does not extend the full width of the view area.
5. Call `wxTabView::AddTab` (p. 688) for each of the tabs you wish to create, passing a unique identifier and a tab label.
6. Construct a number of windows, one for each tab, and call `wxPanelTabView::AddTabWindow` (p. 492) for each of these, passing a tab identifier and the window.
7. Set the tab selection.
8. Show the dialog.

Under Motif, you may also need to size the dialog just before setting the tab selection, for unknown reasons.

Some constraints you need to be aware of:

- All tabs must be of the same width.
- Omit the `wxTAB_STYLE_COLOUR_INTERIOR` flag to ensure that the dialog background and tab backgrounds match.

7.28.1 Example

The following fragment is taken from the file `test.cpp`.

```
void MyDialog::Init(void)
```

```

{
    int dialogWidth = 365;
    int dialogHeight = 390;

    wxButton *okButton = new wxButton(this, wxID_OK, "Close", wxPoint(100,
330), wxSize(80, 25));
    wxButton *cancelButton = new wxButton(this, wxID_CANCEL, "Cancel",
wxPoint(185, 330), wxSize(80, 25));
    wxButton *HelpButton = new wxButton(this, wxID_HELP, "Help",
wxPoint(270, 330), wxSize(80, 25));
    okButton->SetDefault();

    // Note, omit the wxTAB_STYLE_COLOUR_INTERIOR, so we will guarantee a
match
    // with the panel background, and save a bit of time.
    wxPanelTabView *view = new wxPanelTabView(this, wxTAB_STYLE_DRAW_BOX);

    wxRectangle rect;
    rect.x = 5;
    rect.y = 70;
    // Could calculate the view width from the tab width and spacing,
    // as below, but let's assume we have a fixed view width.
    // rect.width = view->GetTabWidth()*4 + 3*view-
>GetHorizontalTabSpacing();
    rect.width = 326;
    rect.height = 250;

    view->SetViewRect(rect);

    // Calculate the tab width for 4 tabs, based on a view width of 326
and
    // the current horizontal spacing. Adjust the view width to exactly
fit
    // the tabs.
    view->CalculateTabWidth(4, TRUE);

    if (!view->AddTab(TEST_TAB_CAT, wxString("Cat")))
        return;

    if (!view->AddTab(TEST_TAB_DOG, wxString("Dog")))
        return;
    if (!view->AddTab(TEST_TAB_GUINEAPIG, wxString("Guinea Pig")))
        return;
    if (!view->AddTab(TEST_TAB_GOAT, wxString("Goat")))
        return;
    if (!view->AddTab(TEST_TAB_ANTEATER, wxString("Ant-eater")))
        return;
    if (!view->AddTab(TEST_TAB_SHEEP, wxString("Sheep")))
        return;
    if (!view->AddTab(TEST_TAB_COW, wxString("Cow")))
        return;
    if (!view->AddTab(TEST_TAB_HORSE, wxString("Horse")))
        return;
    if (!view->AddTab(TEST_TAB_PIG, wxString("Pig")))
        return;
    if (!view->AddTab(TEST_TAB_OSTRICH, wxString("Ostrich")))
        return;
    if (!view->AddTab(TEST_TAB_AARDVARK, wxString("Aardvark")))
        return;
    if (!view->AddTab(TEST_TAB_HUMMINGBIRD, wxString("Hummingbird")))
        return;
}

```

```
// Add some panels
wxPanel *panell1 = new wxPanel(this, -1, wxPoint(rect.x + 20, rect.y +
10), wxSize(290, 220), wxTAB_TRAVERSAL);
(void)new wxButton(panell1, -1, "Press me", wxPoint(10, 10));
(void)new wxTextCtrl(panell1, -1, "1234", wxPoint(10, 40), wxSize(120,
150));

view->AddTabWindow(TEST_TAB_CAT, panell1);

wxPanel *panel2 = new wxPanel(this, -1, wxPoint(rect.x + 20, rect.y +
10), wxSize(290, 220));

wxString animals[] = { "Fox", "Hare", "Rabbit", "Sabre-toothed tiger",
"T Rex" };
(void)new wxListBox(panel2, -1, wxPoint(5, 5), wxSize(170, 80), 5,
animals);

(void)new wxTextCtrl(panel2, -1, "Some notes about the animals in this
house", wxPoint(5, 100), wxSize(170, 100)),
    wxTE_MULTILINE;

view->AddTabWindow(TEST_TAB_DOG, panel2);

// Don't know why this is necessary under Motif...
#ifdef wx_motif
    this->SetSize(dialogWidth, dialogHeight-20);
#endif

view->SetTabSelection(TEST_TAB_CAT);

this->Centre(wxBOTH);
}
```

7.29 wxTabView overview

Classes: *wxTabView* (p. 687), *wxPanelTabView* (p. 491)

A *wxTabView* manages and draws a number of tabs. Because it is separate from the tabbed window implementation, it can be reused in a number of contexts. This library provides tabbed dialog and panel classes to use with the *wxPanelTabView* class, but an application could derive other kinds of view from *wxTabView*.

For example, a help application might draw a representation of a book on a window, with a row of tabs along the top. The new tab view class might be called *wxCanvasTabView*, for example, with the *wxBookCanvas* posting the *OnEvent* function to the *wxCanvasTabView* before processing further, application-specific event processing.

A window class designed to work with a view class must call the view's *OnEvent* and *Draw* functions at appropriate times.

7.30 Toolbar overview

Classes: *wxToolBar* (p. 746)

The toolbar family of classes allows an application to use toolbars in a variety of configurations and styles.

The toolbar is a popular user interface component and contains a set of bitmap buttons or toggles. A toolbar gives faster access to an application's facilities than menus, which have to be popped up and selected rather laboriously.

Instead of supplying one toolbar class with a number of different implementations depending on platform, wxWindows separates out the classes. This is because there are a number of different toolbar styles that you may wish to use simultaneously, and also, future toolbar implementations will emerge (for example, using the new-style Windows 'coolbar' as seen in Microsoft applications) which cannot all be shoe-horned into the one class.

For each platform, the symbol **wxToolBar** is defined to be one of the specific toolbar classes.

The following is a summary of the toolbar classes and their differences.

- **wxToolBarBase**. This is a base class with pure virtual functions, and should not be used directly.
- **wxToolBarSimple**. A simple toolbar class written entirely with generic wxWindows functionality. A simply 3D effect for buttons is possible, but it is not consistent with the Windows look and feel. This toolbar can scroll, and you can have arbitrary numbers of rows and columns.
- **wxToolBarMSW**. This class implements an old-style Windows toolbar, only on Windows. There are small, three-dimensional buttons, which do not (currently) reflect the current Windows colour settings: the buttons are grey. This is the default wxToolBar on 16-bit windows.
- **wxToolBar95**. Uses the native Windows 95 toolbar class. It dynamically adjusts its background and button colours according to user colour settings. `CreateTools` must be called after the tools have been added. No absolute positioning is supported but you can specify the number of rows, and add tool separators with **AddSeparator**. Tooltips are supported. **OnRightClick** is not supported. This is the default wxToolBar on Windows 95, Windows NT 4 and above.

A toolbar might appear as a single row of images under the menubar, or it might be in a separate frame layout in several rows and columns. The class handles the layout of the images, unless explicit positioning is requested.

A tool is a bitmap which can either be a button (there is no 'state', it just generates an event when clicked) or it can be a toggle. If a toggle, a second bitmap can be provided to depict the 'on' state; if the second bitmap is omitted, either the inverse of the first bitmap will be used (for monochrome displays) or a thick border is drawn around the bitmap (for colour displays where inverting will not have the desired result).

The Windows-specific toolbar classes expect 16-colour bitmaps that are 16 pixels wide and 15 pixels high. If you want to use a different size, call **SetToolBitmapSize** as the

demo shows, before adding tools to the button bar. Don't supply more than one bitmap for each tool, because the toolbar generates all three images (normal, depressed and checked) from the single bitmap you give it.

To intercept

7.30.1 Using the toolbar library

Include "wx/toolbar.h", or if using a class directly, one of:

- "wx/msw/tbarmsw.h" for wxToolBarMSW
- "wx/msw/tbar95.h" for wxToolBar95
- "wx/tbarsmpl.h" for wxToolBarSimple

Example of toolbar use are given in the sample program "toolbar". The source is given below.

```

////////////////////////////////////
////
// Name:          test.cpp
// Purpose:       wxToolBar sample
// Author:        Julian Smart
// Modified by:
// Created:       04/01/98
// RCS-ID:       $Id: ttoolbar.tex,v 1.5 1999/02/09 21:22:33 JS Exp $
// Copyright:    (c) Julian Smart
// Licence:      wxWindows licence
////////////////////////////////////
////

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

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP
#include "wx/wx.h"
#endif

#include "wx/toolbar.h"
#include <wx/log.h>

#include "test.h"

#if defined(__WXGTK__) || defined(__WXMOTIF__)
#include "mondrian.xpm"
#include "bitmaps/new.xpm"
#include "bitmaps/open.xpm"
#include "bitmaps/save.xpm"
#include "bitmaps/copy.xpm"
#include "bitmaps/cut.xpm"
#include "bitmaps/print.xpm"
#include "bitmaps/preview.xpm"

```

```
#include "bitmaps/help.xpm"
#endif

IMPLEMENT_APP(MyApp)

// The `main program' equivalent, creating the windows and returning the
// main frame
bool MyApp::OnInit(void)
{
    // Create the main frame window
    MyFrame* frame = new MyFrame((wxFrame *) NULL, -1, (const wxString)
    "wxToolBar Sample",
        wxPoint(100, 100), wxSize(450, 300));

    // Give it a status line
    frame->CreateStatusBar();

    // Give it an icon
    frame->SetIcon(wxICON(mondrian));

    // Make a menubar
    wxMenu *fileMenu = new wxMenu;
    fileMenu->Append(wxID_EXIT, "E&xit", "Quit toolbar sample" );

    wxMenu *helpMenu = new wxMenu;
    helpMenu->Append(wxID_HELP, "&About", "About toolbar sample");

    wxMenuBar* menuBar = new wxMenuBar;

    menuBar->Append(fileMenu, "&File");
    menuBar->Append(helpMenu, "&Help");

    // Associate the menu bar with the frame
    frame->SetMenuBar(menuBar);

    // Create the toolbar
    frame->CreateToolBar(wxNO_BORDER|wxHORIZONTAL|wxTB_FLAT, ID_TOOLBAR);

    frame->GetToolBar()->SetMargins( 2, 2 );

    InitToolBar(frame->GetToolBar());

    // Force a resize. This should probably be replaced by a call to a
    wxFrame
    // function that lays out default decorations and the remaining
    content window.
    wxSizeEvent event(wxSize(-1, -1), frame->GetId());
    frame->OnSize(event);
    frame->Show(TRUE);

    frame->SetStatusText("Hello, wxWindows");

    SetTopWindow(frame);

    return TRUE;
}

bool MyApp::InitToolBar(wxToolBar* toolBar)
{
    // Set up toolbar
    wxBitmap* toolBarBitmaps[8];
```

```

#ifdef __WXMSW__
    toolBarBitmaps[0] = new wxBitmap("icon1");
    toolBarBitmaps[1] = new wxBitmap("icon2");
    toolBarBitmaps[2] = new wxBitmap("icon3");
    toolBarBitmaps[3] = new wxBitmap("icon4");
    toolBarBitmaps[4] = new wxBitmap("icon5");
    toolBarBitmaps[5] = new wxBitmap("icon6");
    toolBarBitmaps[6] = new wxBitmap("icon7");
    toolBarBitmaps[7] = new wxBitmap("icon8");
#else
    toolBarBitmaps[0] = new wxBitmap( new_xpm );
    toolBarBitmaps[1] = new wxBitmap( open_xpm );
    toolBarBitmaps[2] = new wxBitmap( save_xpm );
    toolBarBitmaps[3] = new wxBitmap( copy_xpm );
    toolBarBitmaps[4] = new wxBitmap( cut_xpm );
    toolBarBitmaps[5] = new wxBitmap( preview_xpm );
    toolBarBitmaps[6] = new wxBitmap( print_xpm );
    toolBarBitmaps[7] = new wxBitmap( help_xpm );
#endif

#ifdef __WXMSW__
    int width = 24;
#else
    int width = 16;
#endif
    int currentX = 5;

    toolBar->AddTool(wxID_NEW, *(toolBarBitmaps[0]), wxNullBitmap, FALSE,
currentX, -1, (wxObject *) NULL, "New file");
    currentX += width + 5;
    toolBar->AddTool(wxID_OPEN, *(toolBarBitmaps[1]), wxNullBitmap, FALSE,
currentX, -1, (wxObject *) NULL, "Open file");
    currentX += width + 5;
    toolBar->AddTool(wxID_SAVE, *(toolBarBitmaps[2]), wxNullBitmap, FALSE,
currentX, -1, (wxObject *) NULL, "Save file");
    currentX += width + 5;
    toolBar->AddSeparator();
    toolBar->AddTool(wxID_COPY, *(toolBarBitmaps[3]), wxNullBitmap, FALSE,
currentX, -1, (wxObject *) NULL, "Copy");
    currentX += width + 5;
    toolBar->AddTool(wxID_CUT, *(toolBarBitmaps[4]), wxNullBitmap, FALSE,
currentX, -1, (wxObject *) NULL, "Cut");
    currentX += width + 5;
    toolBar->AddTool(wxID_PASTE, *(toolBarBitmaps[5]), wxNullBitmap,
FALSE, currentX, -1, (wxObject *) NULL, "Paste");
    currentX += width + 5;
    toolBar->AddSeparator();
    toolBar->AddTool(wxID_PRINT, *(toolBarBitmaps[6]), wxNullBitmap,
FALSE, currentX, -1, (wxObject *) NULL, "Print");
    currentX += width + 5;
    toolBar->AddSeparator();
    toolBar->AddTool(wxID_HELP, *(toolBarBitmaps[7]), wxNullBitmap, FALSE,
currentX, -1, (wxObject *) NULL, "Help");

    toolBar->Realize();

    // Can delete the bitmaps since they're reference counted
    int i;
    for (i = 0; i < 8; i++)
        delete toolBarBitmaps[i];

```

```
    return TRUE;
}

// wxID_HELP will be processed for the 'About' menu and the toolbar help
// button.

BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(wxID_EXIT, MyFrame::OnQuit)
    EVT_MENU(wxID_HELP, MyFrame::OnAbout)
    EVT_CLOSE(MyFrame::OnCloseWindow)
    EVT_TOOL_RANGE(wxID_OPEN, wxID_PASTE, MyFrame::OnToolLeftClick)
    EVT_TOOL_ENTER(wxID_OPEN, MyFrame::OnToolEnter)
END_EVENT_TABLE()

// Define my frame constructor
MyFrame::MyFrame(wxFrame* parent, wxWindowID id, const wxString& title,
const wxPoint& pos,
    const wxSize& size, long style):
    wxFrame(parent, id, title, pos, size, style)
{
    m_textWindow = new wxTextCtrl(this, -1, "", wxPoint(0, 0), wxSize(-1,
-1), wxTE_MULTILINE);
}

void MyFrame::OnQuit(wxCommandEvent& WXUNUSED(event))
{
    Close(TRUE);
}

void MyFrame::OnAbout(wxCommandEvent& WXUNUSED(event))
{
    (void)wxMessageBox("wxWindows toolbar sample", "About wxToolBar");
}

// Define the behaviour for the frame closing
// - must delete all frames except for the main one.
void MyFrame::OnCloseWindow(wxCloseEvent& WXUNUSED(event))
{
    Destroy();
}

void MyFrame::OnToolLeftClick(wxCommandEvent& event)
{
    wxString str;
    str.Printf("Clicked on tool %d", event.GetId());
    SetStatusText(str);
}

void MyFrame::OnToolEnter(wxCommandEvent& event)
{
    if (event.GetSelection() > -1)
    {
        wxString str;
        str.Printf("This is tool number %d", event.GetSelection());
        SetStatusText(str);
    }
    else
        SetStatusText("");
}
```


7.31 Validator overview

Classes: *wxValidator* (p. 781), *wxTextValidator* (p. 728), *wxGenericValidator* (p. 295)

The aim of the validator concept is to make dialogs very much easier to write. A validator is an object that can be plugged into a control (such as a *wxTextCtrl*), and mediates between C++ data and the control, transferring the data in either direction and validating it. It also is able to intercept events generated by the control, providing filtering behaviour without the need to derive a new control class.

You can use a stock validator, such as *wxTextValidator* (p. 728) (which does text control data transfer, validation and filtering) and *wxGenericValidator* (p. 295) (which does data transfer for a range of controls); or you can write your own.

Example

Here is an example of *wxTextValidator* usage.

```
wxTextCtrl *txt1 = new wxTextCtrl(this, VALIDATE_TEXT, "",
    wxPoint(10, 10), wxSize(100, 80), 0,
    wxTextValidator(wxFILTER_ALPHA, &g_data.m_string));
```

In this example, the text validator object provides the following functionality:

1. It transfers the value of *g_data.m_string* (a *wxString* variable) to the *wxTextCtrl* when the dialog is initialised.
2. It transfers the *wxTextCtrl* data back to this variable when the dialog is dismissed.
3. It filters input characters so that only alphabetic characters are allowed.

The validation and filtering of input is accomplished in two ways. When a character is input, *wxTextValidator* checks the character against the allowed filter flag (*wxFILTER_ALPHA* in this case). If the character is inappropriate, it is vetoed (does not appear) and a warning beep sounds. The second type of validation is performed when the dialog is about to be dismissed, so if the default string contained invalid characters already, a dialog box is shown giving the error, and the dialog is not dismissed.

Anatomy of a validator

A programmer creating a new validator class should provide the following functionality.

A validator constructor is responsible for allowing the programmer to specify the kind of validation required, and perhaps a pointer to a C++ variable that is used for storing the data for the control. If such a variable address is not supplied by the user, then the validator should store the data internally.

The *wxValidator::Validate* (p. 783) member function should return *TRUE* if the data in the control (not the C++ variable) is valid. It should also show an appropriate message if data was not valid.

The *wxValidator::TransferToWindow* (p. 783) member function should transfer the data from the validator or associated C++ variable to the control.

The *wxValidator::TransferFromWindow* (p. 783) member function should transfer the data from the control to the validator or associated C++ variable.

There should be a copy constructor, and a *wxValidator::Clone* (p. 782) function which returns a copy of the validator object. This is important because validators are passed by reference to window constructors, and must therefore be cloned internally.

You can optionally define event handlers for the validator, to implement filtering. These handlers will capture events before the control itself does.

For an example implementation, see the *valtext.h* and *valtext.cpp* files in the *wxWindows* library.

How validators interact with dialogs

For validators to work correctly, validator functions must be called at the right times during dialog initialisation and dismissal.

When a *wxDialog::Show* (p. 184) is called (for a modeless dialog) or *wxDialog::ShowModal* (p. 185) is called (for a modal dialog), the function *wxWindow::InitDialog* (p. 814) is automatically called. This in turn sends an initialisation event to the dialog. The default handler for the *wxEVT_INIT_DIALOG* event is defined in the *wxWindow* class to simply call the function *wxWindow::TransferDataToWindow* (p. 842). This function finds all the validators in the window's children and calls the *TransferToWindow* function for each. Thus, data is transferred from C++ variables to the dialog just as the dialog is being shown.

If you are using a window or panel instead of a dialog, you will need to call *wxWindow::InitDialog* (p. 814) explicitly before showing the window.

When the user clicks on a button, for example the OK button, the application should first call *wxWindow::Validate* (p. 842), which returns *FALSE* if any of the child window validators failed to validate the window data. The button handler should return immediately if validation failed. Secondly, the application should call *wxWindow::TransferDataFromWindow* (p. 842) and return if this failed. It is then safe to end the dialog by calling *EndModal* (if modal) or *Show* (if modeless).

In fact, *wxDialog* contains a default command event handler for the *wxID_OK* button. It goes like this:

```
void wxDialog::OnOK(wxCommandEvent& event)
{
    if ( Validate() && TransferDataFromWindow() )
    {
        if ( IsModal() )
            EndModal(wxID_OK);
        else
        {

```

```
        SetReturnCode(wxID_OK);  
        this->Show(FALSE);  
    }  
}
```

So if using validators and a normal OK button, you may not even need to write any code for handling dialog dismissal.

If you load your dialog from a resource file, you'll need to iterate through the controls setting validators, since validators can't be specified in a dialog resource.

7.32 wxExpr overview

wxExpr is a C++ class reading and writing a subset of Prolog-like syntax, supporting objects attribute/value pairs.

wxExpr can be used to develop programs with readable and robust data files. Within wxWindows itself, it is used to parse the `.wxr` dialog resource files.

History of wxExpr

During the development of the tool Hardy within the AIAI, a need arose for a data file format for C++ that was easy for both humans and programs to read, was robust in the face of fast-moving software development, and that provided some compatibility with AI languages such as Prolog and LISP.

The result was the wxExpr library (formerly called PrologIO), which is able to read and write a Prolog-like attribute-value syntax, and is additionally capable of writing LISP syntax for no extra programming effort. The advantages of such a library are as follows:

1. The data files are readable by humans;
2. I/O routines are easier to write and debug compared with using binary files;
3. the files are robust: unrecognised data will just be ignored by the application
4. Inbuilt hashing gives a random access capability, useful for when linking up C++ objects as data is read in;
5. Prolog and LISP programs can load the files using a single command.

The library was extended to use the ability to read and write Prolog-like structures for remote procedure call (RPC) communication. The next two sections outline the two main ways the library can be used.

7.32.1 wxExpr for data file manipulation

The fact that the output is in Prolog syntax is irrelevant for most programmers, who just need a reasonable I/O facility. Typical output looks like this:

```
diagram_definition(type = "Spirit Belief Network").  
  
node_definition(type = "Model",  
    image_type = "Diamond",
```

```
attribute_for_label = "name",
attribute_for_status_line = "label",
colour = "CYAN",
default_width = 120,
default_height = 80,
text_size = 10,
can_resize = 1,
has_hypertext_item = 1,
attributes = ["name", "combining_function", "level_of_belief"])).

arc_definition(type = "Potentially Confirming",
  image_type = "Spline",
  arrow_type = "End",
  line_style = "Solid",
  width = 1,
  segmentable = 0,
  attribute_for_label = "label",
  attribute_for_status_line = "label",
  colour = "BLACK",
  text_size = 10,
  has_hypertext_item = 1,
  can_connect_to = ["Evidence", "Cluster", "Model", "Evidence",
"Evidence", "Cluster"],
  can_connect_from = ["Data", "Evidence", "Cluster", "Evidence", "Data",
"Cluster"])).
```

This is substantially easier to read and debug than a series of numbers and strings.

Note the object-oriented style: a file comprises a series of *clauses*. Each clause is an object with a *functor* or object name, followed by a list of attribute-value pairs enclosed in parentheses, and finished with a full stop. Each attribute value may be a string, a word (no quotes), an integer, a real number, or a list with potentially recursive elements.

The way that the facility is used by an application to read in a file is as follows:

1. The application creates a wxExprDatabase instance.
2. The application tells the database to read in the entire file.
3. The application searches the database for objects it requires, decomposing the objects using the wxExpr API. The database may be hashed, allowing rapid linking-up of application data.
4. The application deletes or clears the wxExprDatabase.

Writing a file is just as easy:

1. The application creates a wxExprDatabase instance.
2. The application adds objects to the database using the API.
3. The application tells the database to write out the entire database, in Prolog or LISP notation.
4. The application deletes or clears the wxExprDatabase.

To use the library, include "wxexpr.h".

7.32.2 wxExpr compilation

For UNIX compilation, ensure that YACC and LEX or FLEX are on your system. Check that the makefile uses the correct programs: a common error is to compile `y_tab.c` with a C++ compiler. Edit the `CCLEX` variable in `make.env` to specify a C compiler. Also, do not attempt to compile `lex_yy.c` since it is included by `y_tab.c`.

For DOS compilation, the simplest thing is to copy `dosyacc.c` to `y_tab.c`, and `doslex.c` to `lex_yy.c`. It is `y_tab.c` that must be compiled (`lex_yy.c` is included by `y_tab.c`) so if adding source files to a project file, ONLY add `y_tab.c` plus the `.cc` files. If you wish to alter the parser, you will need YACC and FLEX on DOS.

The DOS tools are available at the AIAI ftp site, in the tools directory. Note that for FLEX installation, you need to copy `flex.skl` into the directory `c:/lib`.

If you are using Borland C++ and wish to regenerate `lex_yy.c` and `y_tab.c` you need to generate `lex_yy.c` with FLEX and then comment out the 'malloc' and 'free' prototypes in `lex_yy.c`. It will compile with lots of warnings. If you get an undefined `_PROIO_YYWRAP` symbol when you link, you need to remove `USE_DEFINE` from the makefile and recompile. This is because the `parser.y` file has a choice of defining this symbol as a function or as a define, depending on what the version of FLEX expects. See the bottom of `parser.y`, and if necessary edit it to make it compile in the opposite way to the current compilation.

To test out `wxExpr` compile the test program (`samples/wxexpr/wxexpr.exe`), and try loading `test.exp` into the test program. Then save it to another file. If the second is identical to the first, `wxExpr` is in a working state.

7.32.3 Bugs

These are the known bugs:

1. Functors are permissible only in the main clause (object). Therefore nesting of structures must be done using lists, not predicates as in Prolog.
2. There is a limit to the size of strings read in (about 5000 bytes).

7.32.4 Using wxExpr

This section is a brief introduction to using the `wxExpr` package.

First, some terminology. A `wxExprDatabase` is a list of *clauses*, each of which represents an object or record which needs to be saved to a file. A clause has a *functor* (name), and a list of attributes, each of which has a value. Attributes may take the following types of value: string, word, integer, floating point number, and list. A list can itself contain any type, allowing for nested data structures.

Consider the following code.

```
wxExprDatabase db;  
  
wxExpr *my_clause = new wxExpr("object");
```

```
my_clause->AddAttributeValue("id", (long)1);
my_clause->AddAttributeValueString("name", "Julian Smart");
db.Append(my_clause);

ofstream file("my_file");
db.Write(file);
```

This creates a database, constructs a clause, adds it to the database, and writes the whole database to a file. The file it produces looks like this:

```
object(id = 1,
      name = "Julian Smart").
```

To read the database back in, the following will work:

```
wxExprDatabase db;
db.Read("my_file");

db.BeginFind();

wxExpr *my_clause = db.FindClauseByFunctor("object");
int id = 0;
wxString name = "None found";

my_clause->GetAttributeValue("id", id);
my_clause->GetAttributeValue("name", name);

cout << "Id is " << id << ", name is " << name << "\n";
```

Note the setting of defaults before attempting to retrieve attribute values, since they may not be found.

7.33 wxGrid classes overview

wxGrid is a class for displaying and editing tabular information.

To use wxGrid, include the wxgrid.h header file and link with the wxGrid library. Create a wxGrid object, or, if you need to override some default behaviour, create an object of a class derived from wxGrid. You need to call CreateGrid before there are any cells in the grid.

All row and column positions start from zero, and dimensions are in pixels.

If you make changes to row or column dimensions, call UpdateDimensions and then AdjustScrollbars. If you make changes to the grid appearance (such as a change of cell background colour or font), call Refresh for the changes to be shown.

7.33.1 Example

The following fragment is taken from the file samples/grid/test.cpp. Note the call to UpdateDimensions, which is required if the application has changed any dimensions such as column width or row height. You may also need to call AdjustScrollbars. In this case, AdjustScrollbars isn't necessary because it will be called by wxGrid::OnSize which

is invoked when the window is first displayed.

```
// Make a grid
frame->grid = new wxGrid(frame, 0, 0, 400, 400);

frame->grid->CreateGrid(10, 8);
frame->grid->SetColumnWidth(3, 200);
frame->grid->SetRowHeight(4, 45);
frame->grid->SetCellValue("First cell", 0, 0);
frame->grid->SetCellValue("Another cell", 1, 1);
frame->grid->SetCellValue("Yet another cell", 2, 2);
frame->grid->SetCellTextFont(wxTheFontList->FindOrCreateFont(12,
wxROMAN, wxITALIC, wxNORMAL), 0, 0);
frame->grid->SetCellTextColour(*wxRED, 1, 1);
frame->grid->SetCellBackgroundColour(*wxCYAN, 2, 2);
frame->grid->UpdateDimensions();
```

7.34

Drag-and-drop and clipboard overview

Classes: *wxDataObject* (p. 138), *wxTextDataObject* (p. 723), *wxDropSource* (p. 216), *wxDropTarget* (p. 218), *wxTextDropTarget* (p. 726), *wxFileDropTarget* (p. 252)

It has to be noted that the API for drag and drop in *wxWindows* is not yet finished which is mostly due to the fact that DnD support under GTK 1.0 is very rudimentary and entirely different from the XDnD protocol used by GTK 1.2. This also entails that not all of the documentation concerning DnD might be correct and some of the code might get broken in the future. The next release of *wxWindows* will be based on GTK 1.2 and will hopefully include a much improved DnD support. The general design on the *wxDropSource* side will be the same but especially the *wxDropTarget* is almost certain to change.

Note that `wxUSE_DRAG_AND_DROP` must be defined in `setup.h` in order to use Drag'n'Drop in *wxWindows*.

This overview describes *wxWindows* support for drag and drop and clipboard operations. Both of these topics are discussed here because, in fact, they're quite related. Drag and drop and clipboard are just two ways of passing the data around and so the code required to implement both types of the operations is almost the same.

Both operations involve passing some data from one program to another, although the data can be received in the same program as the source. In the case of clipboard transfer, the data is first placed on the clipboard and then pasted into the destination program, while for a drag-and-drop operation the data object is not stored anywhere but is created when the user starts dragging and is destroyed as soon as he ends it, whether the operation was ended successfully or cancelled.

To be a *drag source*, i.e. to provide the data which may be dragged by user elsewhere, you should implement the following steps:

- **Preparation:** First of all, the data object must be created and initialized with the data you wish to drag. For example:

```
wxDataObject *my_data = new wxTextDataObject data("This
```

```
string will be dragged.");
```

- **Drag start:** To start dragging process (typically in response to a mouse click) you must call *DoDragDrop* (p. 217) function of *wxDropSource* object which should be constructed like this:

```
wxDropSource dragSource( this );  
dragSource.SetData( my_data );
```

- **Dragging:** The call to *DoDragDrop()* blocks until the user release the mouse button (unless you override *GiveFeedback* (p. 218) function to do something special). When the mouse moves in a window of a program which understands the same drag-and-drop protocol (any program under Windows or any program supporting GTK 1.0 DnD protocol under X Windows), the corresponding *wxDropTarget* (p. 218) methods are called - see below.
- **Processing the result:** *DoDragDrop()* returns an *effect code* which is one of the values of *wxDragResult* (p. 216) enum. Codes of *wxDragError*, *wxDragNone* and *wxDragCancel* have the obvious meaning and mean that there is nothing to do on the sending end (except of possibly logging the error in the first case). *wxDragCopy* means that the data has been successfully copied and doesn't require any specific actions neither. But *wxDragMove* is special because it means that the data must be deleted from where it was copied. If it doesn't make sense (dragging selected text from a read-only file) you should pass *FALSE* as parameter to *DoDragDrop()* in the previous step.

To be a *drop target*, i.e. to receive the data dropped by user you should follow the instructions below:

- **Initialization:** For a window to be drop target, it needs to have an associated *wxDropTarget* (p. 218) object. Normally, you will call *wxWindow::SetDropTarget* (p. 835) during window creation associating you drop target with it. You must derive a class from *wxDropTarget* and override its pure virtual methods. Alternatively, you may derive from *wxTextDropTarget* (p. 726) or *wxFileDropTarget* (p. 252) and override their *OnDropText()* or *OnDropFiles()* method.
- **Drop:** When the user releases the mouse over a window, *wxWindows* queries the associated *wxDropTarget* object if it accepts the data. For this, *GetFormatCount* (p. 219) and *GetFormat* (p. 219) are used and if the format is supported (i.e. is one of returned by *GetFormat()*), then *OnDrop* (p. 219) is called. Otherwise, *wxDragNone* is returned by *DoDragDrop()* and nothing happens.
- **The end:** After processing the data, *DoDragDrop()* returns either *wxDragCopy* or *wxDragMove* depending on the state of the keys (<Ctrl>, <Shift> and <Alt>) at

the moment of drop. There is currently no way for the drop target to change this return code.

7.35 Multithreading overview

Classes: *wxThread* (p. 736), *wxMutex* (p. 459), *wxCriticalSection* (p. 126), *wxCondition* (p. 110)

wxWindows provides a complete set of classes encapsulating objects necessary in multithreaded (MT) programs: the *thread* (p. 736) class itself and different synchronization objects: *mutexes* (p. 459) and *critical sections* (p. 126) with *conditions* (p. 110).

These classes will hopefully make writing MT programs easier and they also provide some extra error checking (compared to the native (be it Win32 or Posix) thread API), however it is still an untrivial undertaking especially for large projects. Before starting an MT application (or starting to add MT features to an existing one) it is worth asking oneself if there is no easier and safer way to implement the same functionality. Of course, in some situations threads really make sense (classical example is a server application which launches a new thread for each new client), but in others it might be a very poor choice (example: launching a separate thread when doing a long computation to show a progress dialog). Other implementation choices are available: for the progress dialog example it is far better to do the calculations in the *idle handler* (p. 317) or call *wxYield()* (p. 876) periodically to update the screen.

If you do decide to use threads in your application, it is strongly recommended that no more than one thread calls GUI functions. The thread sample shows that it *is* possible for many different threads to call GUI functions at once (all the threads created in the sample access GUI), but it is a very poor design choice for anything except an example. The design which uses one GUI thread and several worker threads which communicate with the main one using events is much more robust and will undoubtedly save you countless problems (example: under Win32 a thread can only access GDI objects such as pens, brushes, &c created by itself and not by the other threads).

Final note: in the current release of *wxWindows*, there are no specific facilities for communicating between the threads. However, the usual *ProcessEvent()* (p. 227) function may be used for thread communication too - but you should provide your own synchronisation mechanism if you use it (e.g. just use a critical section before sending a message) because there is no built-in synchronisation.

7.36 File classes and functions overview

Classes: *wxFile* (p. 241), *wxTempFile* (p. 710), *wxTextFile* (p. 730)

Functions: see *file functions* (p. 845).

wxWindows provides some functions and classes to facilitate working with files. As usual, the accent is put on cross-platform features which explains, for example, the *wxTextFile* (p. 730) class which may be used to convert between different types of text files (DOS/Unix/Mac).

`wxFile` may be used for low-level IO. It contains all usual functions to work with files (opening/closing, reading/writing, seeking...) but, compared to using standard C functions, brings error checking (in case of an error a message is logged using `wxLog` (p. 399) facilities) and closes the file automatically in destructor which may be quite convenient.

`wxTempFile` is a very small file designed to make replacing the files contents safer - see its *documentation* (p. 710) for more details.

`wxTextFile` is a general purpose class for working with small text files on line by line basis. It is especially well suited for working with configuration files and program source files. It can be also used to work with files with "non native" line termination characters and write them as "native" files if needed (in fact, the files may be written in any format).

7.37 Internationalization

Although internationalization (i18n for short) of an application involves far more than just translating its text messages to another message (date, time and currency formats need changing too, some languages are written left to right and others right to left, character encoding may differ and many other things may need changing too), it is a necessary first step. `wxWindows` provides facilities for the messages translation with its `wxLocale` (p. 396) class and is itself fully translated into several languages. Please consult `wxWindows` home page for the most up-to-date translations - and if you translate it into one of the languages not done yet, your translations would be gratefully accepted for inclusion into the future versions of the library!

The `wxWindows` approach to i18n closely follows GNU gettext package. `wxWindows` uses the message catalogs which are binary compatible with gettext catalogs and this allows to use all of the programs in this package to work with them. But note that no additional libraries are needed during the run-time, however, so you have only the message catalogs to distribute and nothing else.

During program development you will need the gettext package for working with message catalogs. **Warning:** gettext versions < 0.10 are known to be buggy, so you should find a later version of it!

There are two kinds of message catalogs: source catalogs which are text files with extension `.po` and binary catalogs which are created from the source ones with `msgfmt` program (part of gettext package) and have the extension `.mo`. Only the binary files are needed during program execution.

The program i18n involves several steps:

1. Translating the strings in the program text using `wxGetTranslation` (p. 852) or equivalently the `_()` macro.
2. Extracting the strings to be translated from the program: this uses the work done in the previous step because `xgettext` program used for string extraction may be told (using its `-k` option) to recognise `_()` and `wxGetTranslation` and extract all strings inside the calls to these functions. Alternatively, you may use `-a` option to extract all the strings, but it will usually result in many strings being found which don't have to be translated at all. This will create a text message catalog - a `.po`

- file.
3. Translating the strings extracted in the previous step to other language(s). It involves editing the .po file.
 4. Compiling the .po file into .mo file to be used by the program.
 5. Setting the appropriate locale in your program to use the strings for the given language: see *wxLocale* (p. 396).

See also the GNU gettext documentation linked from `docs/html/index.htm` in your wxWindows distribution.

7.38 Notes on using the reference

In the descriptions of the wxWindows classes and their member functions, note that descriptions of inherited member functions are not duplicated in derived classes unless their behaviour is different. So in using a class such as `wxScrolledWindow`, be aware that `wxWindow` functions may be relevant.

Note also that arguments with default values may be omitted from a function call, for brevity. Size and position arguments may usually be given a value of -1 (the default), in which case wxWindows will choose a suitable value.

Most strings are returned as `wxString` objects. However, for remaining `char *` return values, the strings are allocated and deallocated by wxWindows. Therefore, return values should always be copied for long-term use, especially since the same buffer is often used by wxWindows.

The member functions are given in alphabetical order except for constructors and destructors which appear first.

8 wxPython Notes

This addendum is written by Robin Dunn, author of the wxPython wrapper

8.1 What is wxPython?

wxPython is a blending of the wxWindows GUI classes and the Python (<http://www.python.org/>) programming language.

Python

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

Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. There are interfaces to many system calls and libraries, and new built-in modules are easily written in C or C++. Python is also usable as an extension language for applications that need a programmable interface.

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

wxPython

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

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

Currently wxPython is available for Win32 platforms and the GTK toolkit (wxGTK) on most Unix/X-windows platforms. The effort to enable wxPython for wxMotif will begin shortly. See *Building Python* (p. 2) for details about getting wxPython working for you.

8.2 Why use wxPython?

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

Another good thing to use wxPython for is quick prototyping of your wxWindows apps. With C++ you have to continuously go through the edit-compile-link-run cycle, which can be quite time consuming. With Python it is only an edit-run cycle. You can easily build

an application in a few hours with Python that would normally take a few days or longer with C++. Converting a wxPython app to a C++/wxWindows app should be a straight forward task.

8.3 Other Python GUIs

There are other GUI solutions out there for Python.

Tkinter

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

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

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

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

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

PythonWin

PythonWin is an add-on package for Python for the Win32 platform. It includes wrappers for MFC as well as much of the win32 API. Because of its foundation, it is very familiar for programmers who have experience with MFC and the Win32 API. It is obviously not compatible with other platforms and toolkits. PythonWin is organized as separate packages and modules so you can use the pieces you need without having to use the GUI portions.

Others

There are quite a few other GUI modules available for Python, some in active use, some that havn't been updated for ages. Most are simple wrappers around some C or C++ toolkit or another, and most are not cross-platform compatible. See this link (<http://www.python.org/download/Contributed.html#Graphics>) for a listing of a few of them.

8.4 Building wxPython

I used SWIG (<http://www.swig.org> (<http://www.swig.org>)) to create the source code

for the extension module. This enabled me to only have to deal with a small amount of code and only have to bother with the exceptional issues. SWIG takes care of the rest and generates all the repetitive code for me. You don't need SWIG to build the extension module as all the generated C++ code is included under the `src` directory. If you try to build `wxPython` and get errors because SWIG is missing, then simply touch the `.cpp` and `.py` files so make won't attempt to build them from the `.i` files.

I added a few minor features to SWIG to control some of the code generation. If you want to play around with this the patches are in `wxPython/SWIG.patches` and they should be applied to the 1.1p5 version of SWIG. These new patches are documented at this site (<http://starship.skyport.net/crew/robind/python/#swig>), and they should also end up in the 1.2 version of SWIG.

`wxPython` is organized as a Python package. This means that the directory containing the results of the build process should be a subdirectory of a directory on the `PYTHONPATH`, (and preferably should be named `wxPython`.) You can control where the build process will dump `wxPython` by setting the `TARGETDIR` makefile variable. The default is `$(WXWIN)/utils/wxPython`. If you leave it here then you should add `$(WXWIN)/utils` to your `PYTHONPATH`. However, you may prefer to use something that is already on your `PYTHONPATH`, such as the `site-packages` directory on Unix systems.

Win32

These instructions assume that you have Microsoft Visual C++ 5.0 or 6.0, that you have installed the command-line tools, and that the appropriate environment variables are set for these tools. You should also have Python 1.5.1 installed, and `wxWindows` installed and built as specified below.

1. Build `wxWindows` with `wxUSE_RESOURCE_LOADING_IN_MSW` set to 1 in `include/wx/msw/setup.h` so icons can be loaded dynamically. While there, make sure `wxUSE_OWNER_DRAWN` is also set to 1.
2. Change into the `$(WXWIN)/utils/wxPython/src` directory.
3. Edit `makefile.vc` and specify where your python installation is at. You may also want to fiddle with the `TARGETDIR` variable as described above.
4. Run `nmake -f makefile.vc`
5. If it builds successfully, congratulations! Move on to the next step. If not then you can try mailing the `wxwin-developers` list for help. Also, I will always have a pre-built win32 version of this extension module at <http://alldunn.com/wxPython> (<http://alldunn.com/wxPython>).
6. Change to the `$(WXWIN)/utils/wxPython/tests` directory.
7. Try executing the test programs. Note that some of these print diagnostic or test info to standard output, so they will require the console version of python. For example:

```
python test1.py
```

To run them without requiring a console, you can use the `pythonw.exe` version of Python either from the command line or from a shortcut.

Unix

These directions assume that you have already successfully built wxWindows for GTK, and installed Python 1.5.1. If you build Python yourself, you will get everything installed that you need simply by doing **make install**. If you get Python from an RPM or other pre-packaged source then there will probably be a separate package with the development libraries, etc. that you will need to install.

1. Change into the `$(WXWIN)/utils/wxPython/src` directory.
2. Edit `Setup.in` and ensure that the flags, directories, and toolkit options are correct, (hopefully this will be done by `configuresoon`.) See the above commentary about `TARGETDIR`. There are a few sample `Setup.in.[platform]` files provided.
3. Run this command to generate a makefile:

```
make -f Makefile.pre.in boot
```

4. Once you have the Makefile, run **make** to build and then **make install** to install the wxPython extension module.
5. Change to the `$(WXWIN)/utils/wxPython/tests` directory.
6. Try executing the test programs. For example:

```
python test1.py
```

8.5 Using wxPython

First things first...

I'm not going to try and teach the Python language here. You can do that at the Python Tutorial (<http://www.python.org/doc/tut/tut.html>). I'm also going to assume that you know a bit about wxWindows already, enough to notice the similarities in the classes used.

Take a look at the following wxPython program. You can find a similar program in the `wxPython/tests` directory, named `test7.py`. If your Python and wxPython are properly installed, you should be able to run it by issuing this command:

```
python test7.py
```

```
001: ## import all of the wxPython GUI package
002: from wxPython.wx import *
003:
004: ## Create a new frame class, derived from the wxPython Frame.
005: class MyFrame(wxFrame):
006:
007:     def __init__(self, parent, id, title):
008:         # First, call the base class' __init__ method to create the
frame
009:         wxFrame.__init__(self, parent, id, title,
```

```

010:                                wxPoint(100, 100), wxSize(160, 100))
011:
012:        # Associate some events with methods of this class
013:        EVT_SIZE(self, self.OnSize)
014:        EVT_MOVE(self, self.OnMove)
015:
016:        # Add a panel and some controls to display the size and
position
017:        panel = wxPanel(self, -1)
018:        wxStaticText(panel, -1, "Size:",
019:                        wxDLG_PNT(panel, wxPoint(4, 4)),
wxDefaultSize)
020:        wxStaticText(panel, -1, "Pos:",
021:                        wxDLG_PNT(panel, wxPoint(4, 14)),
wxDefaultSize)
022:        self.sizeCtrl = wxTextCtrl(panel, -1, "",
023:                                    wxDLG_PNT(panel, wxPoint(24,
024:                                    4))),
025:                                    wxDLG_SZE(panel, wxSize(36, -
1))),
026:                                    wxTE_READONLY)
027:        self.posCtrl = wxTextCtrl(panel, -1, "",
028:                                    wxDLG_PNT(panel, wxPoint(24,
029:                                    14))),
030:                                    wxDLG_SZE(panel, wxSize(36, -1)),
031:                                    wxTE_READONLY)
032:        # This method is called automatically when the CLOSE event is
033:        # sent to this window
034:        def OnCloseWindow(self, event):
035:            # tell the window to kill itself
036:            self.Destroy()
037:
038:        # This method is called by the system when the window is
resized,
039:        # because of the association above.
040:        def OnSize(self, event):
041:            size = event.GetSize()
042:            self.sizeCtrl.SetValue("%s, %s" % (size.width,
size.height))
043:
044:            # tell the event system to continue looking for an event
handler,
045:            # so the default handler will get called.
046:            event.Skip()
047:
048:        # This method is called by the system when the window is moved,
049:        # because of the association above.
050:        def OnMove(self, event):
051:            pos = event.GetPosition()
052:            self.posCtrl.SetValue("%s, %s" % (pos.x, pos.y))
053:
054:
055: # Every wxWindows application must have a class derived from wxApp
056: class MyApp(wxApp):
057:
058:     # wxWindows calls this method to initialize the application
059:     def OnInit(self):
060:
061:         # Create an instance of our customized Frame class

```

```

062:         frame = MyFrame(NULL, -1, "This is a test")
063:         frame.Show(true)
064:
065:         # Tell wxWindows that this is our main window
066:         self.SetTopWindow(frame)
067:
068:         # Return a success flag
069:         return true
070:
071:
072: app = MyApp(0)      # Create an instance of the application class
073: app.MainLoop()     # Tell it to start processing events
074:

```

Things to notice

1. At line 2 the wxPython classes, constants, and etc. are imported into the current module's namespace. If you prefer to reduce namespace pollution you can use "from wxPython import wx" and then access all the wxPython identifiers through the wx module, for example, "wx.wxFrame".
2. At line 13 the frame's sizing and moving events are connected to methods of the class. These helper functions are intended to be like the event table macros that wxWindows employs. But since static event tables are impossible with wxPython, we use helpers that are named the same to dynamically build the table. The only real difference is that the first argument to the event helpers is always the window that the event table entry should be added to.
3. Notice the use of wxDLG_PNT and wxDLG_SZE in lines 19 - 29 to convert from dialog units to pixels. These helpers are unique to wxPython since Python can't do method overloading like C++.
4. There is an OnCloseWindow method at line 34 but no call to EVT_CLOSE to attach the event to the method. Does it really get called? The answer is, yes it does. This is because many of the *standard* events are attached to windows that have the associated *standard* method names. I have tried to follow the lead of the C++ classes in this area to determine what is *standard* but since that changes from time to time I can make no guarantees, nor will it be fully documented. When in doubt, use an EVT_*** function.
5. At lines 17 to 21 notice that there are no saved references to the panel or the static text items that are created. Those of you who know Python might be wondering what happens when Python deletes these objects when they go out of scope. Do they disappear from the GUI? They don't. Remember that in wxPython the Python objects are just shadows of the corresponding C++ objects. Once the C++ windows and controls are attached to their parents, the parents manage them and delete them when necessary. For this reason, most wxPython objects do not need to have a __del__ method that explicitly causes the C++ object to be deleted. If you ever have the need to forcibly delete a window, use the Destroy() method as shown on line 36.
6. Just like wxWindows in C++, wxPython apps need to create a class derived from wxApp (line 56) that implements a method named OnInit, (line 59.) This method should create the application's main window (line 62) and use wxApp.SetTopWindow() (line 66) to inform wxWindows about it.
7. And finally, at line 72 an instance of the application class is created. At this

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

8.6 wxWindows classes implemented in wxPython

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

- `wxAcceleratorEntry` (p. 1)
- `wxAcceleratorTable` (p. 2)
- `wxActivateEvent` (p. 5)
- `wxBitmapButton` (p. 54)
- `wxBitmap` (p. 39)
- `wxBrush` (p. 61)
- `wxButton` (p. 36)
- `wxCalculateLayoutEvent` (p. 68)
- `wxCheckBox` (p. 70)
- `wxCheckListBox` (p. 72)
- `wxChoice` (p. 75)
- `wxClientDC` (p. 81)
- `wxCloseEvent` (p. 84)
- `wxColourData` (p. 89)
- `wxColourDialog` (p. 93)
- `wxColour` (p. 86)
- `wxComboBox` (p. 94)
- `wxCommandEvent` (p. 103)
- `wxConfig` (p. 111)
- `wxControl` (p. 125)
- `wxCursor` (p. 128)
- `wxDC` (p. 151)
- `wxDialog` (p. 178)
- `wxDirDialog` (p. 185)
- `wxDropFilesEvent` (p. 214)
- `wxEraseEvent` (p. 220)
- `wxEvent` (p. 221)
- `wxEvtHandler` (p. 224)
- `wxFileDialog` (p. 248)
- `wxFocusEvent` (p. 263)
- `wxFontData` (p. 270)
- `wxFontDialog` (p. 273)
- `wxFont` (p. 264)
- `wxFrame` (p. 275)
- `wxGauge` (p. 291)
- `wxGridCell`

- `wxGridEvent`
- `wxGrid` (p. 297)
- `wxIconizeEvent`
- `wxIcon` (p. 318)
- `wxIdleEvent` (p. 317)
- `wxImageList` (p. 339)
- `wxIndividualLayoutConstraint` (p. 342)
- `wxInitDialogEvent` (p. 345)
- `wxJoystickEvent` (p. 356)
- `wxKeyEvent` (p. 359)
- `wxLayoutAlgorithm` (p. 362)
- `wxLayoutConstraints` (p. 365)
- `wxListBox` (p. 373)
- `wxListCtrl` (p. 381)
- `wxListEvent` (p. 394)
- `wxListItem` (p. 391)
- `wxMDIChildFrame` (p. 404)
- `wxMDIClientWindow` (p. 407)
- `wxMDIParentFrame` (p. 409)
- `wxMask` (p. 403)
- `wxMaximizeEvent`
- `wxMemoryDC` (p. 415)
- `wxMenuBar` (p. 426)
- `wxMenuEvent` (p. 438)
- `wxMenuItem` (p. 433)
- `wxMenu` (p. 418)
- `wxMessageDialog` (p. 439)
- `wxMetaFileDC` (p. 442)
- `wxMiniFrame` (p. 446)
- `wxMouseEvent` (p. 450)
- `wxMoveEvent` (p. 458)
- `wxNotebookEvent` (p. 470)
- `wxNotebook` (p. 464)
- `wxPageSetupData` (p. 477)
- `wxPageSetupDialog` (p. 482)
- `wxPaintDC` (p. 483)
- `wxPaintEvent` (p. 484)
- `wxPalette` (p. 484)
- `wxPanel` (p. 488)
- `wxPen` (p. 494)
- `wxPoint` (p. 503)
- `wxPostScriptDC` (p. 504)
- `wxPrintData` (p. 508)
- `wxPrintDialog` (p. 512)
- `wxPrinterDC` (p. 515)
- `wxQueryLayoutInfoEvent` (p. 535)
- `wxRadioBox` (p. 537)
- `wxRadioButton` (p. 543)

- *wxRealPoint* (p. 546)
- *wxRect* (p. 546)
- *wxRegionIterator* (p. 566)
- *wxRegion* (p. 562)
- *wxSashEvent* (p. 569)
- *wxSashLayoutWindow* (p. 570)
- *wxSashWindow* (p. 573)
- *wxScreenDC* (p. 578)
- *wxScrollBar* (p. 579)
- *wxScrollEvent* (p. 584)
- *wxScrolledWindow* (p. 586)
- *wxShowEvent*
- *wxSingleChoiceDialog* (p. 593)
- *wxSizeEvent* (p. 596)
- *wxSize* (p. 595)
- *wxSlider* (p. 597)
- *wxSpinButton* (p. 622)
- *wxSpinEvent*
- *wxSplitterWindow* (p. 626)
- *wxStaticBitmap* (p. 634)
- *wxStaticBox* (p. 637)
- *wxStaticText* (p. 638)
- *wxStatusBar* (p. 640)
- *wxSysColourChangedEvent* (p. 678)
- *wxTaskBarIcon* (p. 701)
- *wxTextCtrl* (p. 712)
- *wxTextEntryDialog* (p. 725)
- *wxTimer* (p. 745)
- *wxToolBarTool*
- *wxToolBar* (p. 746)
- *wxToolTip*
- *wxTreeCtrl* (p. 761)
- *wxTreeEvent* (p. 774)
- *wxTreeItemData* (p. 773)
- *wxTreeItemId*
- *wxUpdateUIEvent* (p. 775)
- *wxWindowDC* (p. 843)
- *wxWindow* (p. 798)

8.7 Where to go for help

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

<http://starship.python.net/mailman/listinfo/wxpython-users>
(<http://starship.python.net/mailman/listinfo/wxpython-users>)

Or you can send mail directly to the list using this address:

`wxpython-users@starship.python.net`

9 Index



::copystring, 852
::IsEmpty, 853
::Stricmp, 853
::Strlen, 853
::wxBeginBusyCursor, 864
::wxBell, 864
::wxClipboardOpen, 861
::wxCloseClipboard, 862
::wxColourDisplay, 858
::wxConcatFiles, 849
::wxCopyFile, 849
::wxCreateDynamicObject, 865
::wxDDECleanUp, 865
::wxDDEInitialize, 865
::wxDebugMsg, 865
::wxDirExists, 847
::wxDisplayDepth, 858
::wxDisplaySize, 866
::wxDos2UnixFilename, 847
::wxEmptyClipboard, 862
::wxEndBusyCursor, 867
::wxEntry, 866
::wxEnumClipboardFormats, 862
::wxError, 867
::wxExecute, 867
::wxExit, 868
::wxFatalError, 868
::wxFileExists, 847
::wxFileNameFromPath, 847
::wxFileSelector, 854
::wxFindFirstFile, 847
::wxFindMenuItemId, 869
::wxFindNextFile, 848
::wxFindWindowByLabel, 869
::wxFindWindowByName, 869
::wxGetActiveWindow, 869
::wxGetClipboardData, 862
::wxGetClipboardFormatName, 863
::wxGetCwd, 849
::wxGetDisplayName, 870
::wxGetElapsedTime, 870
::wxGetEmailAddress, 849
::wxGetFreeMemory, 871
::wxGetHomeDir, 870
::wxGetHostName, 849, 870
::wxGetMousePosition, 871
::wxGetMultipleChoice, 855
::wxGetOSDirectory, 848
::wxGetOsVersion, 871
::wxGetPrinterCommand, 859
::wxGetPrinterFile, 859
::wxGetPrinterMode, 859
::wxGetPrinterOptions, 859
::wxGetPrinterOrientation, 859
::wxGetPrinterPreviewCommand, 860

::wxGetPrinterScaling, 860
::wxGetPrinterTranslation, 860
::wxGetResource, 872
::wxGetSingleChoice, 856
::wxGetSingleChoiceData, 856
::wxGetSingleChoiceIndex, 856
::wxGetTempFileName, 850
::wxGetTextFromUser, 855
::wxGetTranslation, 854
::wxGetUserId, 850, 872
::wxGetUserName, 850, 873
::wxGetWorkingDirectory, 850
::wxIsAbsolutePath, 848
::wxIsBusy, 873
::wxIsClipboardFormatAvailable, 863
::wxIsWild, 851
::wxKill, 873
::wxLoadUserResource, 873
::wxLogDebug, 888
::wxLogError, 887
::wxLogFatalError, 887
::wxLogMessage, 887
::wxLogStatus, 888
::wxLogSysError, 888
::wxLogTrace, 888
::wxLogVerbose, 888
::wxLogWarning, 887
::wxMakeMetafilePlaceable, 858
::wxMatchWild, 851
::wxMessageBox, 857
::wxMkdir, 851
::wxNewId, 864
::wxNow, 874
::wxOnAssert, 889
::wxOpenClipboard, 863
::wxPathOnly, 848
::wxPostDelete, 874
::wxRegisterClipboardFormat, 863
::wxRegisterId, 864
::wxRemoveFile, 851
::wxRenameFile, 851
::wxResourceAddIdentifier, 883
::wxResourceClear, 883
::wxResourceCreateBitmap, 884
::wxResourceCreateIcon, 884
::wxResourceCreateMenuBar, 884
::wxResourceGetIdentifier, 885
::wxResourceParseData, 885
::wxResourceParseFile, 886
::wxResourceParseString, 886
::wxResourceRegisterBitmapData, 886
::wxResourceRegisterIconData, 887
::wxRmdir, 851
::wxSetClipboardData, 863
::wxSetCursor, 859
::wxSetDisplayName, 874
::wxSetPrinterCommand, 860
::wxSetPrinterFile, 860

-
- ::wxSetPrinterMode, 860
 - ::wxSetPrinterOptions, 861
 - ::wxSetPrinterOrientation, 861
 - ::wxSetPrinterPreviewCommand, 861
 - ::wxSetPrinterScaling, 861
 - ::wxSetPrinterTranslation, 861
 - ::wxSetWorkingDirectory, 851
 - ::wxShell, 875
 - ::wxSleep, 875
 - ::wxSplitPath, 852
 - ::wxStartTimer, 876
 - ::wxStringEq, 853
 - ::wxStringMatch, 853
 - ::wxStripMenuCodes, 875
 - ::wxToLower, 876
 - ::wxToUpper, 876
 - ::wxTrace, 876
 - ::wxTraceLevel, 877
 - ::wxTransferFileToStream, 852
 - ::wxTransferStreamToFile, 852
 - ::wxUnix2DosFilename, 848
 - ::wxWriteResource, 877
 - ::wxYield, 878
- ~ ~ ~
- ~wxAcceleratorTable, 4
 - ~wxApp, 7
 - ~wxArray, 22
 - ~wxArrayString, 27
 - ~wxAutomationObject, 32
 - ~wxBitmap, 42
 - ~wxBitmapButton, 56
 - ~wxBitmapHandler, 51
 - ~wxBrush, 64
 - ~wxBusyCursor, 36
 - ~wxButton, 37
 - ~wxCheckBox, 72
 - ~wxCheckListBox, 74
 - ~wxChoice, 77
 - ~wxClipboard, 83
 - ~wxColourData, 90
 - ~wxColourDialog, 94
 - ~wxComboBox, 96
 - ~wxCommand, 102
 - ~wxCommandProcessor, 109
 - ~wxCondition, 111
 - ~wxConfigBase, 119
 - ~wxCriticalSection, 127
 - ~wxCriticalSectionLocker, 128
 - ~wxCursor, 131
 - ~wxDatabase, 133
 - ~wxDataInputStream, 141
 - ~wxDataObject, 140
 - ~wxDataOutputStream, 142
 - ~wxDate, 144
 - ~wxDC, 152
 - ~wxDialog, 181
 - ~wxDirDialog, 186
 - ~wxDocChildFrame, 188
 - ~wxDocManager, 191
 - ~wxDocMDIChildFrame, 198
 - ~wxDocMDIParentFrame, 200
 - ~wxDocParentFrame, 201
 - ~wxDocTemplate, 205
 - ~wxDocument, 209
 - ~wxDropSource, 218
 - ~wxDropTarget, 219
 - ~wxEvtHandler, 225
 - ~wxExpr, 232
 - ~wxExprDatabase, 239
 - ~wxFile, 243
 - ~wxFileDialog, 250
 - ~wxFileHistory, 255
 - ~wxFileInputStream, 257, 418
 - ~wxFileOutputStream, 258
 - ~wxFileType, 261
 - ~wxFont, 266
 - ~wxFontData, 271
 - ~wxFontDialog, 274
 - ~wxFrame, 278
 - ~wxGauge, 293
 - ~wxGenericValidator, 297
 - ~wxHashTable, 312
 - ~wxHelpController, 314
 - ~wxIcon, 323
 - ~wxImage, 328
 - ~wxImageHandler, 337
 - ~wxInputStream, 347
 - ~wxJoystick, 351
 - ~wxLayoutAlgorithm, 365
 - ~wxList, 371
 - ~wxListBox, 376
 - ~wxListCtrl, 384
 - ~wxLocale, 398
 - ~wxMask, 405
 - ~wxMDIChildFrame, 407
 - ~wxMDIClientWindow, 410
 - ~wxMDIParentFrame, 412
 - ~wxMemoryOutputStream, 419
 - ~wxMenu, 421
 - ~wxMenuBar, 428
 - ~wxMenuItem, 435
 - ~wxMessageDialog, 441
 - ~wxMetafile, 442
 - ~wxMetafileDC, 444
 - ~wxMimeTypesManager, 446
 - ~wxMiniFrame, 449
 - ~wxModule, 450
 - ~wxMutex, 462
 - ~wxMutexLocker, 463
 - ~wxNotebook, 466
 - ~wxObjArray, 22
 - ~wxOutputStream, 477
 - ~wxPageSetupData, 479
 - ~wxPageSetupDialog, 484
 - ~wxPalette, 487
 - ~wxPanel, 491
 - ~wxPanelTabView, 493
 - ~wxPen, 498
 - ~wxPreviewCanvas, 506
 - ~wxPreviewControlBar, 507
 - ~wxPreviewFrame, 509
 - ~wxPrintData, 510

~wxPrintDialog, 514
 ~wxPrinter, 515, 521
 ~wxPrintout, 518
 ~wxPrivateDataObject, 525
 ~wxProcess, 527
 ~wxQueryCol, 532
 ~wxQueryField, 534
 ~wxRadioBox, 540
 ~wxRadioButton, 546
 ~wxRecordSet, 552
 ~wxRegion, 564
 ~wxSashLayoutWindow, 573
 ~wxSashWindow, 576
 ~wxScrollBar, 583
 ~wxScrolledWindow, 589
 ~wxSingleChoiceDialog, 596
 ~wxSlider, 601
 ~wxSocketAddress, 607
 ~wxSocketBase, 609
 ~wxSocketClient, 617
 ~wxSocketHandler, 619
 ~wxSocketServer, 622
 ~wxSortedArray, 22
 ~wxSpinButton, 626
 ~wxSplitterWindow, 629
 ~wxStaticBox, 639
 ~wxStatusBar, 643
 ~wxStreamBase, 648
 ~wxString, 663
 ~wxStringList, 677
 ~wxStringTokenizer, 678
 ~wxTabbedDialog, 684
 ~wxTabCtrl, 698
 ~wxTaskBarIcon, 703
 ~wxTempFile, 713
 ~wxTextCtrl, 716
 ~wxTextEntryDialog, 727
 ~wxTextFile, 737
 ~wxTextValidator, 730
 ~wxThread, 738
 ~wxTimer, 747
 ~wxToolBar, 751
 ~wxTreeCtrl, 764
 ~wxTreeItemData, 775
 ~wxURL, 781
 ~wxValidator, 784
 ~wxVariant, 787
 ~wxView, 796
 ~wxWave, 799
 ~wxWindow, 802

—A—

A selection of SQL commands, 927
 Abort, 515, 530
 Above, 344
 Absolute, 345
 Accept, 622
 AcceptWith, 623
 Access, 243
 Activate, 408, 796
 ActivateNext, 413

ActivatePrevious, 413
 ActivateView, 191
 Add, 22, 27, 340, 341, 494, 677
 AddAttributeValue, 232
 AddAttributeValueString, 233
 AddAttributeValueStringList, 233
 AddAttributeValueWord, 233
 AddBrush, 68
 AddCatalog, 398
 AddCatalogLookupPathPrefix, 398
 AddChild, 802
 AddData, 83
 AddDocument, 191
 AddEnvList, 494
 AddFile, 248
 AddFilesToMenu, 255
 AddFileToHistory, 192, 255
 AddFont, 276
 AddHandler, 42, 328
 Adding items, 19
 AddLine, 736
 AddMonths, 144
 AddNew, 552
 AddPage, 467
 AddPen, 503
 AddRoot, 764
 AddSeparator, 751
 AddTab, 689
 AddTabPanel, 493
 AddTool, 751
 AddView, 209
 AddWeeks, 145
 AddYears, 145
 AdjustScrollbars, 299
 AdvanceSelection, 467
 Advise, 169, 708
 AfterFirst, 664
 AfterLast, 664
 Alloc, 23, 27, 663
 Allocating and deleting wxWindows objects, 8
 AllocData, 534
 AltDown, 362, 454
 Append, 77, 96, 233, 239, 371, 376, 421, 429, 664, 787
 AppendCols, 299
 AppendField, 533
 AppendItem, 765
 AppendRows, 299
 AppendSeparator, 422
 AppendText, 723
 Arg, 233
 argc, 7
 argv, 7
 Arrange, 384
 ArrangeIcons, 413
 AsIs, 345
 AssociateTemplate, 192
 Attach, 244
 AttributeValue, 234

—B—

BeforeFirst, 664
 BeforeLast, 665
 BeginBatch, 299
 BeginDrawing, 152
 BeginFind, 239, 312
 BeginQuery, 552
 BeginTrans, 133
 Below, 345
 BindVar, 532, 552
 Bitmap format handlers, 912
 Bitmap resource format, 959
 Blit, 152
 Blue, 88
 bottom, 367
 Break, 422
 Broadcast, 111
 Bugs, 976
 Build, 607
 Button, 454
 ButtonDClick, 454
 ButtonDown, 358, 455
 ButtonIsDown, 358
 ButtonUp, 358, 455

—C—

c_str, 665
 CalculateTabWidth, 689
 CallMethod, 32
 CanAppend, 552
 Cancel, 133, 553
 CanRestart, 553
 CanScroll, 553
 CanTransact, 133, 553
 CanUndo, 102, 109
 CanUpdate, 133, 553
 CanVeto, 85
 CaptureMouse, 802
 Cascade, 413
 Case conversion, 658
 CellHitTest, 299
 Center, 803
 Centre, 181, 278, 803
 centreX, 367
 centreY, 367
 char*, 745
 Character access, 657
 ChDir, 288
 Check, 74, 173, 423, 429, 436, 779
 Check Windows debug messages, 13
 Checked, 106
 CLASSINFO, 878
 CleanUpHandlers, 42, 328
 CleanupModules, 450
 Clear, 23, 28, 77, 83, 96, 153, 312, 371, 376,
 564, 607, 665, 677, 716, 803
 ClearCommands, 109
 ClearData, 535
 ClearDatabase, 239
 ClearList, 787

ClearSel, 601
 ClearTabs, 690
 ClearTicks, 601
 ClearWindows, 493
 ClientToScreen, 803
 Clone, 298, 730, 784
 Close, 83, 133, 209, 244, 444, 734, 796, 804
 Cmp, 665
 CmpNoCase, 665
 Collapse, 765
 CollapseAndReset, 765
 Command, 126, 279
 Commit, 713
 CommitTrans, 134
 CompareTo, 666
 Comparison, 657
 Comparison of wxString to other string classes,
 901
 Comparison operators, 675
 Compiling the resource system, 960
 Concatenation, 657
 Connect, 225, 617
 Constraint layout: more detail, 921
 ConstructDefaultSQL, 553
 Constructor and destructor, 114, 445
 Constructors and assignment operators, 656
 Constructors and destructors, 18
 Contains, 565, 666
 ControlDown, 362, 455
 ConvertDialogToPixels, 805
 ConvertPixelsToDialog, 806
 ConvertToBitmap, 329
 Copy, 97, 234, 716, 794
 copystring, 853
 Count, 23, 28, 620
 CountTokens, 678
 Create, 38, 42, 51, 56, 72, 77, 97, 119, 173, 181,
 244, 279, 293, 329, 341, 376, 385, 405, 408,
 414, 449, 467, 487, 491, 541, 546, 583, 589,
 601, 626, 629, 637, 640, 641, 643, 698, 711,
 716, 738, 800, 927
 CreateAbortWindow, 515
 CreateButtons, 507
 CreateCanvas, 509
 CreateClient, 410, 620
 CreateControlBar, 509
 CreateDocument, 192, 205
 CreateGrid, 300
 CreateInstance, 32
 CreateLogTarget, 7
 CreateObject, 80
 CreateServer, 620
 CreateStatusBar, 279
 CreateToolBar, 280
 CreateTools, 752
 CreateView, 192, 205
 CrossHair, 153
 CurrentCellVisible, 300
 Customization, 401
 Cut, 97, 716

—D—

Data, 464
 Data transfer, 950
 DECLARE_ABSTRACT_CLASS, 879
 DECLARE_APP, 879
 DECLARE_CLASS, 879
 DECLARE_DYNAMIC_CLASS, 880
 Default, 226
 Default constructors, 21
 delete, 476
 Delete, 97, 313, 376, 553, 677, 739, 765, 788
 Delete entries/groups, 117
 DeleteAll, 119
 DeleteAllItems, 385, 698, 765
 DeleteAllPages, 468
 DeleteAllViews, 210
 DeleteAttributeValue, 235
 DeleteCols, 300
 DeleteColumn, 385
 DeleteContents, 371
 DeleteEntry, 119
 DeleteGroup, 119
 DeleteItem, 385, 698
 DeleteNode, 371
 DeleteObject, 371
 DeletePage, 468
 DeleteRows, 300
 DeleteSubMenu, 436
 Deselect, 377
 Destroy, 329, 806
 DestroyChildren, 807
 DestroyClippingRegion, 154
 Detach, 23, 244, 528
 DeviceToLogicalX, 154
 DeviceToLogicalXRel, 154
 DeviceToLogicalY, 154
 DeviceToLogicalYRel, 154
 Dialog resource format, 956
 Disassemble, 608
 DisassociateTemplate, 192
 Discard, 614, 713
 DiscardEdits, 717
 Disconnect, 170, 708
 Dispatch, 8
 DisplayBlock, 315
 DisplayContents, 315
 DisplaySection, 315
 Do, 102, 109
 DoDragDrop, 218
 DontCreateOnDemand, 119, 402
 Dos2UnixFilename, 847
 DragAcceptFiles, 807
 Dragging, 455
 Draw, 341, 690
 DrawArc, 154
 DrawBitmap, 155
 DrawBlankPage, 521
 DrawEllipse, 155
 DrawEllipticArc, 155
 DrawField, 645
 DrawFieldText, 645

DrawIcon, 155
 DrawLine, 156
 DrawLines, 156
 DrawPoint, 156
 DrawPolygon, 156
 DrawRectangle, 157
 DrawRoundedRectangle, 157
 DrawSpline, 157
 DrawText, 157
 DrawTool, 753
 Dump, 174, 473

—E—

Edges and relationships, 344
 Edit, 385, 554
 EditLabel, 766
 Empty, 23, 28, 666
 Enable, 423, 429, 436, 541, 779, 807
 EnableEffects, 271
 EnableHelp, 479, 510
 EnableMargins, 479
 EnableOrientation, 479
 EnablePageNumbers, 510
 EnablePaper, 480
 EnablePrinter, 480
 EnablePrintToFile, 510
 EnableScrolling, 589
 EnableSelection, 510
 EnableTool, 753
 EnableTop, 430
 EndBatch, 300
 EndDoc, 158
 EndDrawing, 158
 EndDrawingOnTop, 580
 EndEditLabel, 766
 EndModal, 181
 EndPage, 158
 EndQuery, 554
 EnsureFileAccessible, 494
 EnsureVisible, 385, 766
 Enter, 127
 Entering, 455
 Enumeration, 115
 Eof, 244, 735
 Eq, 794
 Error, 609
 ErrorOccured, 134
 ErrorSnapshot, 134
 Event macros summary, 946
 EVT_COMMAND(id, event, func), 947
 EVT_COMMAND_RANGE(id1, id2, event, func), 947
 EVT_CUSTOM(event, id, func), 946
 EVT_CUSTOM_RANGE(event, id1, id2, func), 947
 Example, 914, 962, 964, 978
 Example 1: subwindow layout, 922
 Example 2: panel item layout, 923
 Examples, 951
 Execute, 170, 708
 ExecuteSQL, 554

Exists, 120, 245, 734
 Exit, 450
 ExitMainLoop, 9
 Expand, 766
 ExpandCommand, 262

—F—

fd, 244
 FileHistoryAddFilesToMenu, 193
 FileHistoryLoad, 193
 FileHistoryRemoveMenu, 193
 FileHistorySave, 193
 FileHistoryUseMenu, 193
 FillBuffer, 655
 FillVar, 532
 FillVars, 554
 Find, 372, 666
 FindAbsoluteValidPath, 495
 FindClass, 80
 FindClause, 240
 FindClauseByFunctor, 240
 FindColour, 92
 FindFocus, 807
 FindHandler, 43, 329, 330
 FindItem, 385, 386, 423
 FindItemById, 431
 FindItemForId, 424
 FindMenuItem, 430
 FindName, 93
 FindOrCreateBrush, 68
 FindOrCreateFont, 276
 FindOrCreatePen, 503
 FindString, 77, 97, 377, 541
 FindTabControlForId, 690
 FindTabControlForPosition, 690
 FindTemplateForPath, 194
 FindToolForPosition, 753
 FindValidPath, 495
 FindWindow, 808
 First, 666
 Fit, 808
 Fixed, 655
 FloodFill, 158
 Flush, 120, 245, 402
 Flushable, 655
 FlushBuffer, 655
 FormatDate, 145
 FormatTime, 744
 Frec, 667
 Functions and macros, 237
 Functor, 235

—G—

Genetic mutation, 13
 Get, 120, 313
 GetActive, 6
 GetActiveChild, 415
 GetActiveTarget, 402
 GetAlignment, 537, 573
 GetAllowSymbols, 271

GetAllPages, 511
 GetAppName, 8, 120
 GetAttributeValue, 234
 GetAttributeValueStringList, 234
 GetAuto3D, 8
 GetBackground, 158
 GetBackgroundBrush, 690
 GetBackgroundColour, 436, 691, 808
 GetBackgroundPen, 691
 GetBaseClassName1, 80
 GetBaseClassName2, 80
 GetBatchCount, 301
 GetBezelFace, 293
 GetBitmap, 61, 436, 637
 GetBitmapFocus, 57
 GetBitmapLabel, 56, 57
 GetBitmapSelected, 57
 GetBlue, 330
 GetBool, 788
 GetBottom, 549
 GetBoundingRect, 767
 GetBox, 565
 GetBrush, 158
 GetBufferEnd, 654
 GetBufferPos, 654
 GetBufferStart, 654
 GetButtonChange, 359
 GetButtonState, 351, 359
 GetC, 347
 GetCanvas, 522
 GetCap, 498
 GetCell, 301
 GetCellAlignment, 301
 GetCellBackgroundColour, 301, 302
 GetCells, 302
 GetCellTextColour, 302
 GetCellTextFont, 302
 GetCellValue, 303
 GetChar, 652, 667, 788
 GetCharHeight, 159, 809
 GetCharWidth, 159, 809
 GetChecked, 779
 GetCheckPrevious, 174
 GetChildren, 809
 GetChildrenCount, 767
 GetChooseFull, 90
 GetChosenFont, 272
 GetClassInfo, 474
 GetClassName, 8, 81
 GetClientData, 98, 107, 226, 235, 377
 GetClientSize, 414, 809
 GetClientWindow, 415
 GetClippingBox, 159
 GetCollate, 511
 GetColName, 554
 GetColour, 64, 90, 272, 498
 GetColourData, 94
 GetColPosition, 686
 GetCols, 303
 GetColType, 554, 555
 GetColumn, 386
 GetColumns, 78, 555

GetColumnWidth, 303, 386
GetCommand, 1
GetCommandProcessor, 210
GetCommands, 109
GetConstraints, 809
GetContentType, 531
GetCount, 23, 28, 767, 787
GetCountPerPage, 386
GetCurFocus, 698
GetCurrentDocument, 194
GetCurrentLine, 735
GetCurrentPage, 522
GetCurrentRecord, 555
GetCurrentRect, 303
GetCurrentView, 194
GetCurrentWindow, 493
GetCursorColumn, 303
GetCursorRow, 303
GetCustomColour, 91
GetDashes, 499
GetData, 84, 330, 525, 532, 535, 667, 788
GetDatabase, 555
GetDatabaseName, 134
GetDataLeft, 655
GetDataSize, 525
GetDataSource, 134
GetDataSources, 555
GetDate, 788
GetDay, 145, 743
GetDayOfWeek, 145, 743
GetDayOfWeekName, 146
GetDayOfYear, 146
GetDaysInMonth, 146
GetDC, 221, 518
GetDebugMode, 174
GetDefaultConnect, 556
GetDefaultExtension, 205
GetDefaultInfo, 482
GetDefaultItem, 810
GetDefaultMinMargins, 481
GetDefaultSQL, 556
GetDepth, 44, 323
GetDescription, 205, 262
GetDirectory, 206, 250
GetDispatchPtr, 32
GetDocument, 189, 199, 796
GetDocumentManager, 206, 210, 797
GetDocumentName, 206, 210
GetDocuments, 194
GetDocumentTemplate, 210
GetDocumentWindow, 210
GetDouble, 788
GetDragRect, 571
GetDragStatus, 571
GetDropTarget, 810
GetEdge, 571
GetEditable, 304
GetEditControl, 386, 767
GetEditMenu, 109
GetEnabled, 780
GetEnableEffects, 272
GetEnableHelp, 481
GetEnableMargins, 481
GetEnableOrientation, 481
GetEnablePaper, 481
GetEnablePrinter, 481
GetEOL, 737
GetError, 530, 782
GetErrorClass, 134
GetErrorCode, 135, 556
GetErrorCount, 240
GetErrorMessage, 135
GetErrorNumber, 135
GetEventClass, 223
GetEventHandler, 810
GetEventObject, 223
GetEventType, 223
GetEvtHandlerEnabled, 227
GetExcludeList, 731
GetExitOnDelete, 9
GetExtension, 52, 337
GetExtensions, 261
GetExtraLong, 107
GetFaceName, 266
GetFamily, 267
GetFieldData, 557
GetFieldDataPtr, 557
GetFieldRect, 644
GetFieldsCount, 644
GetFileFilter, 206
GetFileHistory, 194
GetFilename, 211, 250
GetFiles, 216, 248
GetFileTypeFromExtension, 446
GetFileTypeFromMimeType, 446
GetFilter, 557
GetFilterIndex, 250
GetFirst, 235, 372
GetFirstChild, 767
GetFirstDayOfMonth, 146
GetFirstEntry, 120
GetFirstGroup, 120
GetFirstLine, 735
GetFirstView, 211
GetFirstVisibleItem, 768
GetFlags, 2, 69, 206, 537
GetFont, 159, 436, 686, 810
GetFontData, 274
GetFontId, 267
GetForce, 86
GetForegroundColour, 810
GetFormat, 219, 248, 253, 728
GetFormatCount, 219, 253, 728
GetFrame, 522, 797
GetFromPage, 511
GetGrandParent, 811
GetGreen, 330
GetH, 569
GetHandle, 811
GetHandlers, 44, 331
GetHDBC, 135
GetHeader, 318
GetHeight, 44, 323, 331, 549, 686
GetHelp, 436

GetHelpString, 424, 431
GetHENV, 135
GetHighlightColour, 691
GetHighlightPen, 691
GetHistoryFile, 255
GetHorizontalTabOffset, 691
GetHorizScrollBar, 304
GetHour, 743
GetHourGMT, 743
GetIcon, 261
GetId, 223, 436, 525, 526, 686, 775, 811
GetID, 739
GetImageCount, 342
GetImageList, 387, 468, 698, 768
GetIncludeList, 731
GetIndent, 768
GetInfo, 135
GetInitialFont, 272
GetInputStream, 291, 317, 530, 782
GetInsertionPoint, 98, 717
GetInstance, 32
GetInt, 107
GetIntPosition, 654
GetItem, 387
GetItemCount, 388, 699
GetItemData, 387, 699, 768
GetItemImage, 699, 768
GetItemPosition, 387
GetItemRect, 387, 699
GetItemSelectedImage, 770
GetItemSpacing, 388
GetItemState, 388
GetItemText, 388, 699, 769
GetJoin, 499
GetJoystick, 359
GetJulianDate, 146
GetKeyCode, 2
GetLabel, 38, 126, 425, 431, 542, 641, 686, 812
GetLabelAlignment, 304
GetLabelBackgroundColour, 304
GetLabelSize, 304
GetLabelTextColour, 304
GetLabelTextFont, 304
GetLabelTop, 432
GetLabelValue, 305
GetLast, 235, 372
GetLastAccess, 655
GetLastChild, 769
GetLastLine, 736
GetLastPosition, 98, 717
GetLastResult, 288
GetLeft, 549
GetLevel, 174
GetLine, 734
GetLineCount, 734
GetLineLength, 717
GetLineSize, 602
GetLineText, 718
GetLineType, 736
GetList, 290
GetLocale, 398
GetLoggingOff, 85
GetLogicalFunction, 159
GetLong, 788
GetManufacturerId, 351
GetMapMode, 159
GetMarginBottomRight, 480
GetMargins, 755
GetMarginTopLeft, 480, 482
GetMarginWidth, 437
GetMask, 45
GetMaskBlue, 331
GetMaskGreen, 331
GetMaskRed, 331
GetMax, 602, 626
GetMaxCommands, 109
GetMaxDocsOpen, 194
GetMaxFiles, 255
GetMaximumSizeX, 577
GetMaximumSizeY, 577
GetMaxPage, 511, 522
GetMaxSize, 755
GetMenu, 432
GetMenuBar, 281
GetMenuCount, 432
GetMenuId, 440
GetMessage, 187, 250
GetMimeType, 261
GetMin, 602, 626
GetMinimumPaneSize, 629
GetMinimumSizeX, 577
GetMinimumSizeY, 577
GetMinMarginBottomRight, 480
GetMinMarginTopLeft, 480
GetMinPage, 511, 522
GetMinute, 743
GetMinuteGMT, 743
GetMonth, 146, 744
GetMonthEnd, 146
GetMonthName, 147
GetMonthStart, 147
GetMovementThreshold, 351
GetName, 52, 102, 337, 399, 437, 532, 736, 789, 812
GetNext, 235
GetNextChild, 769
GetNextEntry, 121
GetNextGroup, 121
GetNextHandler, 227
GetNextItem, 388
GetNextLine, 735
GetNextSibling, 769
GetNextToken, 679
GetNextVisible, 770
GetNoCopies, 511
GetNoHistoryFiles, 195, 256
GetNumberAxes, 351
GetNumberButtons, 351
GetNumberCols, 558
GetNumberFields, 558
GetNumberJoysticks, 352
GetNumberOfEntries, 121
GetNumberOfFiles, 216
GetNumberOfGroups, 121

GetNumberOfLayers, 691
GetNumberOfLines, 718
GetNumberParams, 558
GetNumberRecords, 559
GetObject, 33
GetObjectType, 223
GetODBCVersionFloat, 136
GetODBCVersionString, 136
GetOldSelection, 472
GetOpenCommand, 262
GetOptimization, 160
GetOptions, 559
GetOrientation, 481, 511, 537, 573, 587
GetOutputStream, 290
GetPage, 468
GetPageCount, 468
GetPageImage, 468
GetPageInfo, 518
GetPageSetupData, 484
GetPageSize, 583, 602
GetPageSizeMM, 518
GetPageSizePixels, 518
GetPageText, 469
GetPalette, 44
GetPaperSize, 480
GetParent, 770, 812
GetPassword, 136
GetPath, 121, 186, 251, 782
GetPen, 160
GetPid, 529
GetPixel, 88, 160, 488
GetPointSize, 267
GetPollingMax, 352
GetPollingMin, 352
GetPosition, 216, 352, 359, 459, 549, 587, 811
GetPOVCTSPosition, 353
GetPOVPosition, 352
GetPPIPrinter, 518
GetPPIScreen, 519
GetPrevious, 465
GetPreviousHandler, 227
GetPrevLine, 736
GetPrevSibling, 770
GetPrevVisible, 770
GetPrimaryKeys, 557, 559
GetPrintableName, 211
GetPrintCommand, 262
GetPrintData, 514, 515, 522
GetPrintDC, 514
GetPrintout, 522
GetPrintoutForPrinting, 522
GetPrintPreview, 507
GetPriority, 739
GetProductId, 352
GetProductName, 352
GetProperty, 33
GetProtocol, 781
GetProtocolName, 781
GetRange, 294, 583
GetRect, 70, 569, 813
GetRed, 331
GetRefData, 474
GetRequestedLength, 537
GetResultSet, 559
GetReturnCode, 813
GetRight, 549
GetRootItem, 770
GetRowCount, 469, 699
GetRowHeight, 305
GetRowPosition, 686
GetRows, 305
GetRudderMax, 353
GetRudderMin, 353
GetRudderPosition, 353
GetSashPosition, 630
GetSashVisible, 577
GetScrollPixelsPerUnit, 590
GetScrollPos, 813
GetScrollPosX, 305
GetScrollPosY, 305
GetScrollRange, 814
GetScrollThumb, 813
GetSecond, 744
GetSecondGMT, 744
GetSeconds, 744
GetSelected, 686
GetSelectedItemCount, 389
GetSelectedTabFont, 691
GetSelection, 78, 98, 107, 378, 469, 472, 542, 596, 699, 771
GetSelectionClientData, 596
GetSelections, 378
GetSelEnd, 602
GetSelStart, 603
GetSessionEnding, 85
GetSetChecked, 780
GetSetEnabled, 780
GetSetText, 780
GetShadowColour, 692
GetShadowPen, 693
GetShadowWidth, 294
GetShowHelp, 272
GetSize, 60, 81, 140, 160, 533, 535, 537, 549, 598, 725, 814
GetSkipped, 224
GetSortString, 559
GetSplitMode, 630
GetSQL, 559
GetStatusBar, 281
GetStatusText, 644
GetStipple, 64, 499
GetStream, 175, 505
GetStreamBuf, 175
GetString, 78, 98, 107, 379, 399, 544, 679, 789
GetStringSelection, 78, 99, 379, 542, 596
GetStyle, 64, 187, 251, 267, 499, 731
GetSubMenu, 437
GetSystemColour, 681
GetSystemFont, 681
GetSystemMetric, 682
GetTabFont, 692
GetTabHeight, 692
GetTableName, 560
GetTables, 560

GetTabSelectionHeight, 692
GetTabStyle, 692
GetTabView, 684, 685
GetTabWidth, 692
GetTabWindow, 493
GetText, 725, 780
GetTextBackground, 160
GetTextColour, 389, 437, 693
GetTextExtent, 161, 814
GetTextForeground, 161
GetTextItem, 305
GetThumbLength, 584, 603
GetThumbPosition, 583
GetTickFreq, 603
GetTime, 789
GetTimestamp, 224
GetTimeStampFormat, 403
GetTitle, 181, 211, 281, 425, 815
GetToolBar, 281, 415
GetToolBitmapSize, 754
GetToolClientData, 755
GetToolEnabled, 755
GetToolLongHelp, 756
GetToolPacking, 756
GetToolSeparation, 756
GetToolShortHelp, 756
GetToolSize, 754
GetToolState, 757
GetTop, 549
GetToPage, 512
GetTopItem, 389
GetTopMargin, 693
GetTopWindow, 9
GetType, 52, 337, 532, 535, 560, 789, 794
GetUMax, 353
GetUMin, 353
GetUnderlined, 267
GetUpdateRegion, 815
GetUPosition, 353
GetUsername, 136
GetValue, 72, 99, 294, 546, 604, 627, 718, 727
GetVendorName, 121
GetVerbose, 403
GetVerticalTabTextSpacing, 693
GetVertScrollBar, 305
GetView, 189, 199
GetViewName, 206, 797
GetViewRect, 693
GetVirtualSize, 590
GetVMax, 354
GetVMin, 354
GetVoidPtr, 789
GetVPosition, 354
GetW, 568
GetWeekOfMonth, 147
GetWeekOfYear, 147
GetWeight, 268
GetWidth, 45, 323, 331, 500, 550, 568, 569, 687
GetWildcard, 251
GetWindow, 693, 784
GetWindow1, 630
GetWindow2, 630

GetWindowStyleFlag, 816
GetWritableChar, 667
GetWriteBuf, 667
GetX, 362, 455, 550, 568, 597, 687
GetXMax, 354
GetXMin, 354, 355
GetY, 362, 456, 550, 568, 597, 687
GetYear, 147, 744
GetYearEnd, 147
GetYearStart, 147
GetYMax, 354
GetYMin, 354
GetZMax, 355
GetZoomControl, 508
GetZPosition, 355, 359
GiveFeedback, 218
GoTo, 560
GoToLine, 735
Green, 88
GuessType, 736

H

HasBorder, 577
HasEntry, 122
HasGroup, 122
HashFind, 240, 241
HasMask, 332
HasMoreTokens, 679
HasPage, 519
HasPendingMessages, 403
HasPOV, 355
HasPOV4Dir, 355
HasPOVCTS, 355
HasRudder, 355
HasStream, 175
HasU, 356
HasV, 356
HasZ, 356
HaveRects, 569
height, 367, 549
Helper functions, 445
HitTest, 389, 687, 700, 771
Hostname, 349
How events are processed, 943

I

Icon resource format, 959
Iconize, 182, 281
identifiers, 945
IMPLEMENT_ABSTRACT_CLASS, 880
IMPLEMENT_ABSTRACT_CLASS2, 880, 881
IMPLEMENT_APP, 881
IMPLEMENT_CLASS, 881
IMPLEMENT_CLASS2, 881
IMPLEMENT_DYNAMIC_CLASS, 882
IMPLEMENT_DYNAMIC_CLASS2, 882
Index, 24, 28, 667
IndexOf, 372, 465
Init, 399, 451
InitColours, 645

InitDialog, 491, 816
 Initialization functions, 445
 Initialize, 93, 110, 195, 315, 509, 630
 InitializeClasses, 81
 Initialized, 9
 InitializeModules, 451
 InitStandardHandlers, 45, 332
 InputStreamBuffer, 348
 Insert, 24, 28, 233, 372, 789, 927
 InsertCols, 306
 InsertColumn, 390
 InsertHandler, 45, 332
 InsertItem, 390, 700, 771
 InsertItems, 379
 InsertLine, 737
 InsertPage, 469
 InsertRows, 306
 IntegerValue, 236
 Intersect, 565
 Interval, 748
 Introduction, 901, 942
 Invoke, 34
 InWaitForDataSource, 137
 IsAlive, 739
 IsAscii, 668
 IsBetween, 744
 IsBOF, 560
 IsBold, 771
 IsButton, 359, 456
 IsCheckable, 437
 IsChecked, 75, 425, 432, 437
 IsColNullable, 561
 IsConnected, 609
 IsData, 609
 IsDeleted, 561
 IsDirty, 535
 IsDisconnected, 609
 IsEmpty, 24, 29, 566, 668, 853
 IsEnabled, 426, 433, 437, 816
 IsEOF, 561
 IsExpanded, 772
 IsExpandingEnvVars, 122
 IsFieldDirty, 560, 561
 IsFieldNull, 561
 IsIconInstalled, 703
 IsIconized, 182, 282
 IsKindOf, 81, 474
 IsLeapYear, 148
 IsLoaded, 399
 IsLocked, 462
 IsMain, 739
 IsMaximized, 282
 IsModal, 182
 IsModified, 211, 718
 IsMove, 360
 IsNoWait, 609
 IsNull, 668, 789
 IsNullable, 533
 IsNumber, 668
 IsOfType, 446
 IsOk, 356, 464, 800
 IsOK, 703

IsOpen, 137, 561
 IsOpened, 245, 713, 734
 IsPaused, 740
 IsPreview, 519
 IsRecordingDefaults, 122
 IsRetained, 591, 816
 IsRowDirty, 533
 IsRunning, 740
 IsSameAs, 668
 IsSelected, 772
 IsSelection, 107
 IsSeparator, 438
 IsShown, 816
 IsSplit, 631
 IsSupported, 84
 IsType, 790
 IsVisible, 206, 772
 IsWord, 669
 IsZMove, 360
 Item, 24, 29
 ItemHasChildren, 772

—K—

Key access, 116
 KeyCode, 362
 KeywordSearch, 316
 Kill, 740

—L—

Last, 25, 29, 669
 LastCount, 610
 LastError, 610, 648
 LastRead, 348
 LastWrite, 478
 Layout, 694, 757, 817
 LayoutFrame, 365
 LayoutMDIFrame, 365
 LayoutWindow, 365
 Leave, 127
 Leaving, 456
 left, 367
 Left, 669
 LeftDClick, 456
 LeftDown, 456
 LeftIsDown, 456
 LeftOf, 345
 LeftUp, 456
 Len, 669
 Length, 245, 669
 ListToArray, 677
 Load, 256
 LoadFile, 46, 52, 316, 323, 332, 338, 719
 LoadFromResource, 817
 LoadObject, 211
 LocalHost, 350
 Lock, 462
 LogicalToDeviceX, 161
 LogicalToDeviceXRel, 161
 LogicalToDeviceY, 162
 LogicalToDeviceYRel, 162

Lower, 669, 817
LowerCase, 670

—M—

m_active, 6
m_altDown, 360, 452
m_cancelled, 396
m_checked, 778
m_childDocument, 188, 198
m_childView, 188, 198
m_clientData, 106
m_code, 396, 777
m_col, 396
m_commandInt, 106
m_commandProcessor, 208
m_commandString, 106
m_controlDown, 361, 452
m_count, 476
m_currentView, 190
m_dc, 221
m_defaultDocumentNameCounter, 190
m_defaultExt, 202
m_description, 203
m_directory, 203
m_docClassInfo, 203
m_docs, 190
m_docTypeName, 203
m_documentFile, 208
m_documentManager, 203
m_documentModified, 208
m_documentTemplate, 208
m_documentTitle, 208
m_documentTypeName, 209
m_documentViews, 209
m_eventHandle, 222
m_eventObject, 222
m_eventType, 222
m_extraLong, 106
m_fileFilter, 203
m_fileHistory, 190, 254
m_fileHistoryN, 254
m_fileMaxFiles, 254
m_fileMenu, 255
m_files, 216
m_flags, 191, 203
m_id, 222
m_item, 397, 777
m_itemIndex, 396
m_keyCode, 361
m_leftDown, 453
m_maxDocsOpen, 190
m_menuId, 440
m_metaDown, 361, 453
m_middleDown, 453
m_noFiles, 216
m_oldItem, 777
m_oldItemIndex, 396
m_pid, 529
m_pointDrag, 396, 777
m_pos, 216
m_refData, 473

m_rightDown, 453
m_setChecked, 779
m_setEnabled, 779
m_setText, 779
m_shiftDown, 361, 453
m_skipped, 223
m_text, 779
m_timeStamp, 223
m_viewClassInfo, 204
m_viewDocument, 795
m_viewFrame, 795
m_viewTypeName, 204, 795
m_x, 361, 453
m_y, 361, 454
Macros for template array definition, 18
MainLoop, 10
MakeConnection, 167, 706
MakeDefaultName, 195
MakeKey, 313
MakeLower, 670
MakeModal, 817
MakeNull, 790
MakeString, 790
MakeUpper, 670
Master, 621
Matches, 670
Max, 745
Maximize, 282, 408
MaxX, 162
MaxY, 162
Member, 372, 495, 677, 790
Memory management, 18, 659
Menubar resource format, 958
Message buffering, 401
MessageParameters class, 260
MetaDown, 362, 457
Mid, 670
MiddleDClick, 457
MiddleDown, 457
MiddleIsDown, 457
MiddleUp, 457
Min, 745
MinX, 162
MinY, 162
Miscellaneous, 659
Miscellaneous accessors, 116
MkDir, 288
mnTemplates, 191
Modify, 212
Module definition file, 7
More DDE details, 951
MoreRequested, 319
Move, 561, 818
MoveFirst, 562
MoveLast, 562
MoveNext, 562
MovePrev, 562
Moving, 457

—N—

new, 476

Next, 313, 464
 Notify, 748
 Nth, 236, 373
 NullList, 790
 Number, 78, 99, 373, 380, 542
 Number of elements and simple item access, 19



ODBC SQL data types, 926
 Ok, 4, 46, 65, 89, 132, 162, 257, 258, 324, 333, 442, 489, 500, 523, 609
 OnAcceptConnection, 173, 711
 OnActivate, 10, 189, 199, 282, 306, 818
 OnActivateView, 797
 OnAdvise, 170, 708
 OnApply, 183
 OnBeginDocument, 519
 OnBeginPrinting, 520
 OnCalculateLayout, 574
 OnCancel, 183
 OnChangedViewList, 212
 OnChangeFilename, 797
 OnChangeLabels, 306
 OnChangeSelectionLabel, 306
 OnChar, 719, 731, 819
 OnCharHook, 10, 182, 819
 OnClose, 797, 821
 OnCloseDocument, 212
 OnCloseWindow, 189, 199, 200, 202, 509, 821
 OnCommand, 820
 OnCompareItems, 772
 OnCreate, 212, 798
 OnCreateCell, 306
 OnCreateClient, 415
 OnCreateCommandProcessor, 212
 OnCreateFileHistory, 195
 OnCreatePrintout, 798
 OnCreateStatusBar, 283
 OnCreateTabControl, 693
 OnCreateToolBar, 283
 OnDisconnect, 170, 708
 OnDoubleClickSash, 631
 OnDraw, 591, 687
 OnDrop, 220, 253, 729
 OnDropFiles, 253, 719, 822
 OnDropText, 729
 OnEndDocument, 519
 OnEndPrinting, 520
 OnEndSession, 11
 OnEnter, 220
 OnEraseBackground, 822
 OnEvent, 694
 OnExecute, 170, 709
 OnExit, 10, 451, 740
 OnFileClose, 195
 OnFileNew, 195
 OnFileOpen, 196
 OnFileSave, 196
 OnFileSaveAs, 196
 OnIdle, 11, 824
 OnInit, 12, 451

OnInitDialog, 825
 OnKeyDown, 823
 OnKeyUp, 823
 OnKillFocus, 824
 OnLButtonDClick, 704
 OnLButtonDown, 704
 OnLButtonUp, 704
 OnLeave, 220
 OnLeftClick, 307, 757
 OnLog, 402
 OnMakeConnection, 167, 706
 OnMenuCommand, 196, 284, 825
 OnMenuHighlight, 284, 826
 OnMouseEnter, 758
 OnMouseEvent, 826
 OnMouseMove, 704
 OnMove, 827
 OnNewDocument, 213
 OnOK, 183
 OnOpenDocument, 213
 OnPaint, 506, 591, 827
 OnPoke, 170, 709
 OnPreparePrinting, 520
 OnPrintPage, 520
 OnQueryEndSession, 12
 OnQueryLayoutInfo, 574
 OnQuit, 316
 OnRButtonDClick, 704
 OnRButtonDown, 704
 OnRButtonUp, 704
 OnRequest, 171, 709
 OnRightClick, 307, 758
 OnSashPositionChange, 632
 OnSaveDocument, 213
 OnSaveModified, 213
 OnScroll, 592, 828
 OnSelChange, 470
 OnSelectCell, 307
 OnSelectCellImplementation, 308
 OnSetFocus, 829
 OnSetOptions, 137
 OnSize, 284, 829
 OnStartAdvise, 171, 709
 OnStopAdvise, 171, 709
 OnSysColourChanged, 184, 491, 646, 830
 OnSysRead, 648
 OnSysSeek, 648
 OnSysTell, 648
 OnSysWrite, 648
 OnTabActivate, 694
 OnTabPreActivate, 694
 OnTerminate, 528
 OnUnsplit, 631
 OnUpdate, 798
 OnWaitForDataSource, 137
 Open, 84, 137, 245, 712, 734
 operator, 673
 operator
 =, 746
 operator -, 149, 746
 operator --, 150
 operator !=, 5, 50, 67, 89, 132, 151, 270, 326,

336, 489, 502, 551, 676, 792
operator (), 675
operator [], 674, 792
operator +, 149, 674, 746
operator ++, 150, 569
operator +=, 149, 674, 747
operator <, 150, 676, 745
operator <<, 151, 675, 724
operator <=, 150, 676, 746
operator =, 4, 27, 49, 66, 89, 91, 132, 270, 274, 325, 335, 489, 502, 551, 567, 598, 674, 745, 790, 791
operator -=, 150, 747
operator ==, 4, 50, 66, 89, 132, 151, 270, 326, 336, 489, 502, 551, 675, 746, 791, 792
operator >, 150, 676, 746
operator >=, 150, 676, 746
operator >>, 675
operator bool, 570
operator char, 792
operator const char*, 675
operator double, 793
operator long, 793
operator void*, 793
operator wxDate, 793
operator wxString, 149, 793
operator wxString::+, 674
operator wxTime, 793
operator[], 735
operator=, 22
operatorp[], 27
Options, 117
Other string related functions and classes, 903
OutputStreamBuffer, 478

—P—

Pad, 670
PaintPage, 523
Paste, 99, 720
Path management, 114
Peek, 348, 610
Pending, 13
PercentOf, 345
Play, 442, 800
Pluggable event handlers, 945
Poke, 171, 710
PopEventHandler, 830
PopupMenu, 830
Position, 363, 457
PositionToXY, 720
Positive thinking, 12
Precompiled headers, 9
PrepareDC, 591
Prepend, 670
PrependItem, 772
Print, 516, 523
PrintClasses, 175
PrintDialog, 516
Printf, 671
PrintfV, 671
PrintStatistics, 176

Procedures for writing an ODBC application, 925
ProcessEvent, 227
ProcessMessage, 13
PushEventHandler, 831
Put, 313
PutC, 478
PutChar, 652
PutProperty, 34
Pwd, 289

—Q—

Query, 562
Query database, 445
Quit, 317

—R—

Raise, 832
Read, 122, 123, 241, 246, 348, 610, 651, 794
Read16, 141
Read32, 141
Read8, 141
ReadDouble, 141
ReadFromStream, 241
ReadMailcap, 446
ReadMimeTypes, 447
ReadMsg, 613
Realize, 759
RealValue, 236
Reconnect, 530
RecordCountFinal, 562
Red, 89
Ref, 475
Reference counting and why you shouldn't care about it, 903
Refresh, 832
Register, 619
RegisterModule, 451
RegisterModules, 451
ReleaseCapture, 356
ReleaseMouse, 832
Remove, 25, 29, 100, 342, 671, 720
RemoveAll, 342
RemoveBrush, 69
RemoveChild, 832
RemoveDocument, 196
RemoveFont, 276
RemoveHandler, 47, 333
Removelcon, 705
RemoveLast, 671
RemoveLine, 737
RemoveMenu, 256
RemovePage, 470
RemovePen, 504
RemoveView, 213
Removing items, 19
Rename, 289
Rename entries/groups, 117
RenameEntry, 124
RenameGroup, 124
RenderPage, 523

Replace, 99, 343, 671, 721
 ReplaceWindow, 632
 ReportError, 516
 Requery, 562
 Request, 171, 710
 RequestMore, 319
 Reset, 569
 ResetBuffer, 653
 Resource file, 7
 Resource format design issues, 959
 Restore, 408
 RestoreState, 616
 right, 367
 Right, 672
 RightDClick, 458
 RightDown, 458
 RightIsDown, 458
 RightOf, 346
 RightUp, 458
 Rmdir, 288
 RmFile, 289
 RollbackTrans, 137
 RTTI, 9
 Run, 740

—S—

SameAs, 346
 Save, 213, 256
 SaveAs, 214
 SaveFile, 47, 53, 334, 338, 721
 SaveObject, 214
 SaveState, 616
 Scale, 334
 ScreenToClient, 833
 Scroll, 592
 ScrollList, 391
 ScrollTo, 773
 ScrollWindow, 833
 SearchEventTable, 229
 Searching and replacing, 658
 Searching and sorting, 19
 Seek, 246, 652
 SeekEnd, 246
 SeekI, 349
 SeekO, 478
 Select, 928
 SelectDocumentPath, 197
 SelectDocumentType, 197
 Selected, 380
 SelectItem, 773
 SelectObject, 417
 SelectViewType, 197
 SendCommand, 288
 SendIdleEvents, 13
 Service, 349, 350
 Set, 2, 89, 124, 148, 346, 380, 597
 SetAcceleratorTable, 834
 SetActiveTarget, 402
 SetAlignment, 538, 574
 SetAllowSymbols, 273
 SetAppName, 14, 124

SetAuto3D, 14
 SetAutoLayout, 834
 SetBackground, 163
 SetBackgroundColour, 391, 438, 694, 834
 SetBackgroundMode, 163
 SetBellOnError, 784
 SetBezelFace, 294
 SetBitmap, 61, 638
 SetBitmapDisabled, 58
 SetBitmapFocus, 58
 SetBitmapLabel, 58
 SetBitmaps, 438
 SetBitmapSelected, 59
 SetBrush, 164
 SetBufferIO, 653
 SetCanvas, 523
 SetCanVeto, 86
 SetCap, 500
 SetCapture, 356
 SetCellAlignment, 308
 SetCellBackgroundColour, 308
 SetCellTextColour, 308
 SetCellTextFont, 309
 SetCellValue, 309
 SetChar, 672
 SetCheckpoint, 176
 SetCheckPrevious, 176
 SetChooseFull, 91
 SetChosenFont, 273
 SetClassName, 14
 SetClientData, 100, 107, 229, 236, 380
 SetClientSize, 835
 SetClipboard, 442
 SetClippingRegion, 163
 SetCollate, 512
 SetColour, 65, 91, 273, 500
 SetColPosition, 687
 SetColumn, 391
 SetColumns, 78
 SetColumnWidth, 309, 392
 SetCommandProcessor, 214
 SetConstraints, 836
 SetCurrentPage, 524
 SetCursor, 835
 SetDashes, 500
 SetData, 84, 218, 335, 465, 525, 533, 535, 790
 SetDataSource, 137
 SetDebugMode, 177
 SetDefault, 38
 SetDefaultExtension, 207
 SetDefaultInfo, 483
 SetDefaultMinMargins, 483
 SetDefaultProxy, 782
 SetDefaultSize, 574
 SetDefaultSQL, 563
 SetDepth, 48, 324
 SetDescription, 207
 SetDeviceOrigin, 163
 SetDirectory, 207, 251
 SetDirty, 535
 SetDispatchPtr, 35
 SetDividerPen, 309

SetDocument, 189, 199, 798
SetDocumentManager, 207
SetDocumentName, 214
SetDocumentTemplate, 214
SetDropTarget, 837
SetEditable, 310, 721
SetEditMenu, 110
SetEventHandler, 616, 836
SetEventObject, 224
SetEventType, 224
SetEvtHandlerEnabled, 230
SetExcludeList, 731
SetExitOnDelete, 15
SetExpandEnvVars, 124
SetExtension, 54, 339
SetExtraLong, 108
SetFaceName, 268
SetFamily, 268
SetFieldDirty, 534, 562
SetFieldNull, 563
SetFieldsCount, 646
SetFile, 177
SetFileFilter, 207
SetFilename, 214, 251
SetFilterIndex, 251
SetFirstItem, 381
SetFlags, 70, 207, 538, 611
SetFocus, 837
SetFont, 164, 438, 687, 837
SetForce, 86
SetForegroundColour, 837
SetFormat, 148, 745
SetFrame, 524, 798
SetFromPage, 512
SetGridCursor, 310
SetHeader, 318
SetHeight, 48, 325, 550
SetHelp, 438
SetHelpString, 426, 433
SetHighlightColour, 694
SetHorizontalTabOffset, 695
SetIcon, 284, 705
SetId, 224, 525, 526, 688, 776, 838
SetImageList, 392, 470, 701, 773
SetIncludeList, 731
SetIndent, 773
SetInitialFont, 273
SetInsertionPoint, 100, 722
SetInsertionPointEnd, 100, 722
SetInt, 108
SetIntPosition, 654
SetItem, 392, 393
SetItemBold, 773
SetItemData, 393, 701, 773
SetItemHasChildren, 773
SetItemImage, 393, 701, 774
SetItemPosition, 394
SetItemSelectedImage, 774
SetItemSize, 701
SetItemState, 394
SetItemText, 394, 701, 774
SetJoin, 501
SetLabel, 38, 126, 426, 433, 543, 641, 688
SetLabelAlignment, 310
SetLabelBackgroundColour, 310
SetLabelSize, 310
SetLabelTextColour, 310
SetLabelTextFont, 311
SetLabelTop, 434
SetLabelValue, 311
SetLevel, 177
SetLineSize, 605
SetLoggingOff, 86
SetLogicalFunction, 164
SetLoginTimeout, 138
SetMapMode, 165
SetMarginBottomRight, 482
SetMargins, 760
SetMarginWidth, 438
SetMask, 48, 335
SetMaskColour, 335
SetMaxDocsOpen, 197
SetMaximumSizeX, 578
SetMaximumSizeY, 578
SetMaxPage, 512
SetMenuBar, 285
SetMessage, 187, 251
SetMinimumPaneSize, 633
SetMinimumSizeX, 578
SetMinimumSizeY, 578
SetMinMarginBottomRight, 482
SetMinMarginTopLeft, 482
SetMinPage, 512
SetModal, 184
SetMovementThreshold, 357
SetName, 53, 339, 438, 533, 838
SetNextHandler, 230
SetNoCopies, 512
SetNullable, 533
SetOk, 48, 325
SetOldSelection, 472
SetOptimization, 165
SetOption, 149
SetOptions, 563
SetOrientation, 482, 512, 538, 574
SetPadding, 470, 702
SetPageImage, 471
SetPageSize, 470, 605
SetPageText, 471
SetPalette, 49, 163, 838
SetPaperSize, 482
SetPassword, 138, 289, 531
SetPath, 124, 187, 252
SetPen, 166
SetPid, 529
SetPointSize, 269
SetPosition, 688
SetPreviousHandler, 230
SetPrintout, 524
SetPrintToFile, 513
SetPriority, 740
SetProxy, 783
SetQueryTimeout, 138
SetRange, 273, 295, 604, 627

SetRecordDefaults, 125
SetRect, 70
SetRefData, 475
SetRequestedLength, 538
SetReturnCode, 839
SetRGB, 335
SetRowHeight, 311
SetRowPosition, 688
SetSashBorder, 579
SetSashPosition, 633
SetSashVisible, 578
SetScrollbar, 584, 839
SetScrollbars, 593
SetScrollPos, 840
SetSelected, 688
SetSelectedTabFont, 695
SetSelection, 79, 101, 381, 471, 472, 543, 605, 702, 722
SetSetupDialog, 513
SetShadowColour, 695
SetShadowWidth, 295
SetShowHelp, 273
SetSingleStyle, 394
SetSize, 535, 538, 688, 841
SetSizeHints, 842
SetSplitMode, 633
SetStandardError, 177
SetStatusBar, 286
SetStatusText, 286, 646
SetStatusWidths, 286, 647
SetStipple, 65, 501
SetStream, 178
SetString, 108, 381, 679
SetStringSelection, 79, 382, 543
SetStyle, 66, 187, 252, 269, 501, 731
SetSynchronousMode, 138
SetTabFont, 695
SetTableName, 563
SetTabSelection, 696
SetTabSelectionHeight, 695
SetTabSize, 695
SetTabStyle, 695
SetTabView, 684, 685
SetText, 726, 780
SetTextBackground, 166
SetTextColour, 394, 439, 696
SetTextForeground, 166
SetThumbLength, 606
SetThumbPosition, 584
SetTick, 606
SetTickFreq, 604
SetTimestamp, 224
SetTimeStampFormat, 403
SetTitle, 184, 215, 287, 427, 843
SetToolBar, 287, 416
SetToolBitmapSize, 759
SetToolLongHelp, 760
SetToolPacking, 761
SetToolSeparation, 762
SetToolShortHelp, 761
SetToPage, 513
SetTopMargin, 696
SetTopWindow, 15
SetTraceMask, 403
SetType, 54, 339, 534, 536, 563
SetUnderlined, 269
Setup, 516
SetUser, 289, 531
SetUsername, 138
SetUserScale, 166
SetValue, 72, 101, 295, 546, 606, 627, 722, 727
SetVendorName, 125
SetVerbose, 403
SetVerticalTabTextSpacing, 696
SetView, 189, 199
SetViewer, 316
SetViewName, 799
SetViewRect, 696
SetWeight, 270
SetWidth, 49, 325, 501, 550
SetWildcard, 252
SetWindow, 696, 784
SetWindowStyleFlag, 394
SetX, 550
SetY, 550
SetZoom, 524
SetZoomControl, 508
ShiftDown, 363, 459
Show, 185, 544, 843
ShowModal, 94, 185, 187, 252, 275, 441, 484, 514, 596, 728
ShowPosition, 723
ShowWindowForTab, 493
Shrink, 25, 30, 672
Signal, 111
Simplify the problem, 12
Skip, 224
Sleep, 741
SockAddrLen, 608
SocketEvent, 619
Some advice about using wxString, 902
Sort, 25, 30, 373, 678
SortChildren, 774
SortItems, 394
SplitHorizontally, 634
SplitVertically, 635
sprintf, 672
Start, 748
StartAdvise, 172, 710
StartDoc, 166
StartDrawingOnTop, 580
StartPage, 166
Static functions, 113, 400
std::string compatibility functions, 660
Stop, 748
StopAdvise, 172, 710
Stream, 656
StreamSize, 649
Stricmp, 853
String length, 656
StringValue, 236
Strip, 672
Strlen, 853
Submit, 110

SubString, 672
 Substring extraction, 658
 Subtract, 566
 Supported bitmap file formats, 911

—T—

Tell, 247, 652
 TellI, 349
 TellO, 478
 Templates, 9
 Tests of existence, 116
 The format of a .WXR file, 954
 This, 741
 Tile, 416
 Toggle, 774
 ToggleTool, 762
 top, 367
 TransferDataFromWindow, 843
 TransferDataToWindow, 844
 TransferToWindow, 298, 732, 785
 Trim, 673
 Truncate, 673
 TryLock, 462
 Tuning wxString for your application, 904
 Type, 236
 Type of NULL, 9

—U—

Unconstrained, 345
 Undo, 103, 110
 UngetWriteBuf, 673
 Union, 566
 Unlock, 463
 UnRead, 613
 UnRef, 475
 UnRegister, 620
 Unselect, 774
 Unsplit, 635
 Update, 563, 928
 UpdateAllViews, 215
 UpdateDimensions, 311
 UpdateUI, 427
 Upper, 673
 UpperCase, 673
 Use a debugger, 12
 Use ASSERT, 11
 Use logging functions, 12
 Use relative positioning or constraints, 11
 Use the wxWindows debugging facilities, 12
 Use wxString in preference to character arrays, 11
 Use wxWindows resource files, 11
 UseMenu, 256
 Using the toolbar library, 968
 Using wxExpr, 976

—V—

Validate, 732, 785, 844
 ValidHost, 168, 706

Veto, 86
 ViewStart, 594

—W—

Wait, 111, 614, 615, 621
 WaitForRead, 614
 WaitForWrite, 615
 WaitOnConnect, 618
 WarpPointer, 844
 width, 368, 548
 Window identifiers, 945
 Window layout examples, 922
 WordValue, 237
 Write, 125, 241, 247, 478, 611, 651, 713, 737, 795
 WriteBack, 651
 WriteBitmap, 61
 WriteData, 61, 140, 525, 526, 726
 WriteExpr, 237
 WriteLisp, 241
 WriteLispExpr, 237
 WriteMsg, 612
 WritePrologClause, 237
 WriteString, 726
 WriteText, 723
 Writing values into the string, 658
 WX_CLEAR_ARRAY, 21
 WX_DECLARE_OBJARRAY, 20
 WX_DEFINE_ARRAY, 19
 WX_DEFINE_OBJARRAY, 20
 WX_DEFINE_SORTED_ARRAY, 20
 wxAcceleratorEntry, 1
 wxAcceleratorEntry::GetCommand, 1
 wxAcceleratorEntry::GetFlags, 2
 wxAcceleratorEntry::GetKeyCode, 2
 wxAcceleratorEntry::Set, 2
 wxAcceleratorEntry::wxAcceleratorEntry, 1
 wxAcceleratorTable, 3
 wxAcceleratorTable::~~wxAcceleratorTable, 4
 wxAcceleratorTable::Ok, 4
 wxAcceleratorTable::operator !=, 5
 wxAcceleratorTable::operator =, 4
 wxAcceleratorTable::operator ==, 4
 wxAcceleratorTable::wxAcceleratorTable, 3
 wxActivateEvent, 6
 wxActivateEvent::GetActive, 6
 wxActivateEvent::m_active, 6
 wxActivateEvent::wxActivateEvent, 6
 wxApp, 7
 wxApp::~~wxApp, 7
 wxApp::argc, 7
 wxApp::argv, 7
 wxApp::CreateLogTarget, 7
 wxApp::Dispatch, 8
 wxApp::ExitMainLoop, 9
 wxApp::GetAppName, 8
 wxApp::GetAuto3D, 8
 wxApp::GetClassName, 8
 wxApp::GetExitOnDelete, 9
 wxApp::GetTopWindow, 9
 wxApp::Initialized, 9

wxApp::MainLoop, 10
 wxApp::OnActivate, 10
 wxApp::OnCharHook, 10
 wxApp::OnEndSession, 11
 wxApp::OnExit, 10
 wxApp::OnIdle, 11
 wxApp::OnInit, 12
 wxApp::OnQueryEndSession, 12
 wxApp::Pending, 13
 wxApp::ProcessMessage, 13
 wxApp::SendIdleEvents, 13
 wxApp::SetAppName, 14
 wxApp::SetAuto3D, 14
 wxApp::SetClassName, 14
 wxApp::SetExitOnDelete, 15
 wxApp::SetTopWindow, 15
 wxApp::wxApp, 7
 wxArray, 21
 wxArray copy constructor and assignment
 operator, 21
 wxArray::~~wxArray, 22
 wxArray::Add, 22
 wxArray::Alloc, 22
 wxArray::Clear, 23
 wxArray::Count, 23
 wxArray::Empty, 23
 wxArray::GetCount, 23
 wxArray::Index, 24
 wxArray::Insert, 24
 wxArray::IsEmpty, 24
 wxArray::Item, 24
 wxArray::Last, 25
 wxArray::Remove, 25
 wxArray::Shrink, 25
 wxArray::Sort, 25
 wxArrayString, 26, 27
 wxArrayString::~~wxArrayString, 27
 wxArrayString::Add, 27
 wxArrayString::Alloc, 27
 wxArrayString::Clear, 28
 wxArrayString::Count, 28
 wxArrayString::Empty, 28
 wxArrayString::GetCount, 28
 wxArrayString::Index, 28
 wxArrayString::Insert, 28
 wxArrayString::IsEmpty, 29
 wxArrayString::Item, 29
 wxArrayString::Last, 29
 wxArrayString::operator[], 27
 wxArrayString::operator=, 27
 wxArrayString::Remove (by index), 29
 wxArrayString::Remove (by value), 29
 wxArrayString::Shrink, 30
 wxArrayString::Sort (alphabetically), 30
 wxArrayString::Sort (user defined), 30
 wxArrayString::wxArrayString, 26
 wxASSERT, 889
 wxASSERT_MSG, 889
 wxAutomationObject, 31
 wxAutomationObject::~~wxAutomationObject, 31
 wxAutomationObject::CallMethod, 32
 wxAutomationObject::CreateInstance, 32
 wxAutomationObject::GetDispatchPtr, 32
 wxAutomationObject::GetInstance, 32
 wxAutomationObject::GetObject, 33
 wxAutomationObject::GetProperty, 33
 wxAutomationObject::Invoke, 33
 wxAutomationObject::PutProperty, 34
 wxAutomationObject::SetDispatchPtr, 35
 wxAutomationObject::wxAutomationObject, 31
 wxBeginBusyCursor, 864
 wxBell, 864
 wxBitmap, 39, 40
 wxBitmap::~~wxBitmap, 42
 wxBitmap::AddHandler, 42
 wxBitmap::CleanUpHandlers, 42
 wxBitmap::Create, 42
 wxBitmap::FindHandler, 43
 wxBitmap::GetDepth, 44
 wxBitmap::GetHandlers, 44
 wxBitmap::GetHeight, 44
 wxBitmap::GetMask, 45
 wxBitmap::GetPalette, 44
 wxBitmap::GetWidth, 45
 wxBitmap::InitStandardHandlers, 45
 wxBitmap::InsertHandler, 45
 wxBitmap::LoadFile, 46
 wxBitmap::Ok, 46
 wxBitmap::operator !=, 50
 wxBitmap::operator =, 49
 wxBitmap::operator ==, 50
 wxBitmap::RemoveHandler, 47
 wxBitmap::SaveFile, 47
 wxBitmap::SetDepth, 48
 wxBitmap::SetHeight, 48
 wxBitmap::SetMask, 48
 wxBitmap::SetOk, 48
 wxBitmap::SetPalette, 49
 wxBitmap::SetWidth, 49
 wxBitmap::wxBitmap, 39
wxBITMAP_TYPE_BMP, 40, 328
wxBITMAP_TYPE_BMP_RESOURCE, 40
 wxBITMAP_TYPE_CUR, 130
 wxBITMAP_TYPE_CUR_RESOURCE, 130
wxBITMAP_TYPE_GIF, 40, 321
 wxBITMAP_TYPE_ICO, 130, 321
wxBITMAP_TYPE_ICO_RESOURCE, 321
wxBITMAP_TYPE_JPEG, 328
wxBITMAP_TYPE_PNG, 328
wxBITMAP_TYPE_RESOURCE, 40
wxBITMAP_TYPE_XBM, 40, 130, 321
wxBITMAP_TYPE_XPM, 40, 321
 wxBitmapButton, 55
 wxBitmapButton::~~wxBitmapButton, 56
 wxBitmapButton::Create, 56
 wxBitmapButton::GetBitmapDisabled, 56
 wxBitmapButton::GetBitmapFocus, 57
 wxBitmapButton::GetBitmapLabel, 57
 wxBitmapButton::GetBitmapSelected, 57
 wxBitmapButton::SetBitmapDisabled, 58
 wxBitmapButton::SetBitmapFocus, 58
 wxBitmapButton::SetBitmapLabel, 58
 wxBitmapButton::SetBitmapSelected, 59
 wxBitmapButton::wxBitmapButton, 55

wxBitmapDataObject, 60
 wxBitmapDataObject::GetBitmap, 60
 wxBitmapDataObject::GetSize, 60
 wxBitmapDataObject::SetBitmap, 61
 wxBitmapDataObject::WriteBitmap, 61
 wxBitmapDataObject::WriteData, 61
 wxBitmapDataObject::wxBitmapDataObject, 60
 wxBitmapHandler, 51
 wxBitmapHandler::~wxBitmapHandler, 51
 wxBitmapHandler::Create, 51
 wxBitmapHandler::GetExtension, 52
 wxBitmapHandler::GetName, 52
 wxBitmapHandler::GetType, 52
 wxBitmapHandler::LoadFile, 52
 wxBitmapHandler::SaveFile, 53
 wxBitmapHandler::SetExtension, 54
 wxBitmapHandler::SetName, 53
 wxBitmapHandler::SetType, 54
 wxBitmapHandler::wxBitmapHandler, 51
 wxBrush, 62, 63
 wxBrush::~wxBrush, 63
 wxBrush::GetColour, 64
 wxBrush::GetStipple, 64
 wxBrush::GetStyle, 64
 wxBrush::Ok, 65
 wxBrush::operator !=, 67
 wxBrush::operator =, 66
 wxBrush::operator ==, 66
 wxBrush::SetColour, 65
 wxBrush::SetStipple, 65
 wxBrush::SetStyle, 66
 wxBrush::wxBrush, 62
 wxBrushList, 68
 wxBrushList::AddBrush, 68
 wxBrushList::FindOrCreateBrush, 68
 wxBrushList::RemoveBrush, 69
 wxBrushList::wxBrushList, 68
 wxBU_AUTODRAW, 55
 wxBusyCursor, 36
 wxBusyCursor::~wxBusyCursor, 36
 wxBusyCursor::wxBusyCursor, 35
 wxButton, 37
 wxButton::~wxButton, 37
 wxButton::Create, 37
 wxButton::GetLabel, 38
 wxButton::SetDefault, 38
 wxButton::SetLabel, 38
 wxButton::wxButton, 36
 wxCalculateLayoutEvent, 69
 wxCalculateLayoutEvent::GetFlags, 69
 wxCalculateLayoutEvent::GetRect, 69
 wxCalculateLayoutEvent::SetFlags, 70
 wxCalculateLayoutEvent::SetRect, 70
 wxCalculateLayoutEvent::wxCalculateLayoutEvent, 69
 wxCAPTION, 179, 276, 406, 411, 447
 wxCB_DROPDOWN, 94
 wxCB_READONLY, 94
 wxCB_SIMPLE, 94
 wxCB_SORT, 94
 wxCHECK, 890
 wxCHECK_MSG, 890
 wxCheckBox, 71
 wxCheckBox::~wxCheckBox, 72
 wxCheckBox::Create, 72
 wxCheckBox::GetValue, 72
 wxCheckBox::SetValue, 72
 wxCheckBox::wxCheckBox, 71
 wxCheckListBox, 73
 wxCheckListBox::~wxCheckListBox, 74
 wxCheckListBox::Check, 74
 wxCheckListBox::IsChecked, 75
 wxCheckListBox::wxCheckListBox, 73
 wxChoice, 76
 wxChoice::~wxChoice, 76
 wxChoice::Append, 77
 wxChoice::Clear, 77
 wxChoice::Create, 77
 wxChoice::FindString, 77
 wxChoice::GetColumns, 77
 wxChoice::GetSelection, 78
 wxChoice::GetString, 78
 wxChoice::GetStringSelection, 78
 wxChoice::Number, 78
 wxChoice::SetColumns, 78
 wxChoice::SetSelection, 79
 wxChoice::SetStringSelection, 79
 wxChoice::wxChoice, 75
 wxClassInfo, 80, 961
 wxClassInfo::CreateObject, 80
 wxClassInfo::FindClass, 80
 wxClassInfo::GetBaseClassName1, 80
 wxClassInfo::GetBaseClassName2, 80
 wxClassInfo::GetClassName, 81
 wxClassInfo::GetSize, 81
 wxClassInfo::InitializeClasses, 81
 wxClassInfo::IsKindOf, 81
 wxClassInfo::wxClassInfo, 80
 wxClientDC, 82
 wxClientDC::wxClientDC, 82
 wxCLIP_CHILDREN, 801
 wxClipboard, 83
 wxClipboard::~wxClipboard, 83
 wxClipboard::AddData, 83
 wxClipboard::Clear, 83
 wxClipboard::Close, 83
 wxClipboard::GetData, 83
 wxClipboard::IsSupported, 84
 wxClipboard::Open, 84
 wxClipboard::SetData, 84
 wxClipboard::wxClipboard, 83
 wxClipboardOpen, 861
 wxCloseClipboard, 862
 wxCloseEvent, 85
 wxCloseEvent::CanVeto, 85
 wxCloseEvent::GetForce, 86
 wxCloseEvent::GetLoggingOff, 85
 wxCloseEvent::GetSessionEnding, 85
 wxCloseEvent::SetCanVeto, 86
 wxCloseEvent::SetForce, 86
 wxCloseEvent::SetLoggingOff, 86
 wxCloseEvent::Veto, 86
 wxCloseEvent::wxCloseEvent, 85
 wxColour, 87

wxColour::Blue, 88
wxColour::GetPixel, 88
wxColour::Green, 88
wxColour::Ok, 88
wxColour::operator !=, 89
wxColour::operator =, 89
wxColour::operator ==, 89
wxColour::Red, 89
wxColour::Set, 89
wxColour::wxColour, 87
wxColourData, 90
wxColourData::~~wxColourData, 90
wxColourData::GetChooseFull, 90
wxColourData::GetColour, 90
wxColourData::GetCustomColour, 91
wxColourData::operator =, 91
wxColourData::SetChooseFull, 91
wxColourData::SetColour, 91
wxColourData::SetCustomColour, 91
wxColourData::wxColourData, 90
wxColourDatabase, 92
wxColourDatabase::FindColour, 92
wxColourDatabase::FindName, 93
wxColourDatabase::Initialize, 93
wxColourDatabase::wxColourDatabase, 92
wxColourDialog, 93
wxColourDialog overview, 917
wxColourDialog::~~wxColourDialog, 93
wxColourDialog::GetColourData, 94
wxColourDialog::ShowModal, 94
wxColourDialog::wxColourDialog, 93
wxColourDisplay, 858
wxComboBox, 95
wxComboBox::~~wxComboBox, 96
wxComboBox::Append, 96
wxComboBox::Clear, 96
wxComboBox::Copy, 97
wxComboBox::Create, 97
wxComboBox::Cut, 97
wxComboBox::Delete, 97
wxComboBox::FindString, 97
wxComboBox::GetClientData, 98
wxComboBox::GetInsertionPoint, 98
wxComboBox::GetLastPosition, 98
wxComboBox::GetSelection, 98
wxComboBox::GetString, 98
wxComboBox::GetStringSelection, 99
wxComboBox::GetValue, 99
wxComboBox::Number, 99
wxComboBox::Paste, 99
wxComboBox::Remove, 100
wxComboBox::Replace, 99
wxComboBox::SetClientData, 100
wxComboBox::SetInsertionPoint, 100
wxComboBox::SetInsertionPointEnd, 100
wxComboBox::SetSelection, 101
wxComboBox::SetValue, 101
wxComboBox::wxComboBox, 95
wxCommand, 102
wxCommand overview, 940
wxCommand::~~wxCommand, 102
wxCommand::CanUndo, 102
wxCommand::Do, 102
wxCommand::GetName, 102
wxCommand::Undo, 103
wxCommand::wxCommand, 102
wxCommandEvent, 106
wxCommandEvent::Checked, 106
wxCommandEvent::GetClientData, 106
wxCommandEvent::GetExtraLong, 107
wxCommandEvent::GetInt, 107
wxCommandEvent::GetSelection, 107
wxCommandEvent::GetString, 107
wxCommandEvent::IsSelection, 107
wxCommandEvent::m_clientData, 106
wxCommandEvent::m_commandInt, 106
wxCommandEvent::m_commandString, 106
wxCommandEvent::m_extraLong, 106
wxCommandEvent::SetClientData, 107
wxCommandEvent::SetExtraLong, 108
wxCommandEvent::SetInt, 108
wxCommandEvent::SetString, 108
wxCommandEvent::wxCommandEvent, 106
wxCommandProcessor, 108
wxCommandProcessor overview, 940
wxCommandProcessor::~~wxCommandProcessor, 109
wxCommandProcessor::CanUndo, 109
wxCommandProcessor::ClearCommands, 109
wxCommandProcessor::Do, 109
wxCommandProcessor::GetCommands, 109
wxCommandProcessor::GetEditMenu, 109
wxCommandProcessor::GetMaxCommands, 109
wxCommandProcessor::Initialize, 110
wxCommandProcessor::SetEditMenu, 110
wxCommandProcessor::Submit, 110
wxCommandProcessor::Undo, 110
wxCommandProcessor::wxCommandProcessor, 108
wxConcatFiles, 849
wxCondition, 111
wxCondition::~~wxCondition, 111
wxCondition::Broadcast, 111
wxCondition::Signal, 111
wxCondition::Wait, 111
wxCondition::wxCondition, 111
wxConfigBase, 118
wxConfigBase::~~wxConfigBase, 119
wxConfigBase::Create, 119
wxConfigBase::DeleteAll, 119
wxConfigBase::DeleteEntry, 119
wxConfigBase::DeleteGroup, 119
wxConfigBase::DontCreateOnDemand, 119
wxConfigBase::Exists, 120
wxConfigBase::Flush, 120
wxConfigBase::Get, 120
wxConfigBase::GetAppName, 120
wxConfigBase::GetFirstEntry, 120
wxConfigBase::GetFirstGroup, 120
wxConfigBase::GetNextEntry, 121
wxConfigBase::GetNextGroup, 121
wxConfigBase::GetNumberOfEntries, 121
wxConfigBase::GetNumberOfGroups, 121
wxConfigBase::GetPath, 121

wxConfigBase::GetVendorName, 121
 wxConfigBase::HasEntry, 122
 wxConfigBase::HasGroup, 122
 wxConfigBase::IsExpandingEnvVars, 122
 wxConfigBase::IsRecordingDefaults, 122
 wxConfigBase::Read, 122
 wxConfigBase::RenameEntry, 124
 wxConfigBase::RenameGroup, 124
 wxConfigBase::Set, 124
 wxConfigBase::SetAppName, 124
 wxConfigBase::SetExpandingEnvVars, 124
 wxConfigBase::SetPath, 124
 wxConfigBase::SetRecordDefaults, 125
 wxConfigBase::SetVendorName, 125
 wxConfigBase::Write, 125
 wxConfigBase::wxConfigBase, 118
 wxControl::Command, 126
 wxControl::GetLabel, 126
 wxControl::SetLabel, 126
 wxCopyFile, 849
 wxCreateDynamicObject, 865
 wxCriticalSection, 127
 wxCriticalSection::~wxCriticalSection, 127
 wxCriticalSection::Enter, 127
 wxCriticalSection::Leave, 127
 wxCriticalSection::wxCriticalSection, 127
 wxCriticalSectionLocker, 128
 wxCriticalSectionLocker::~wxCriticalSectionLocker, 128
 wxCriticalSectionLocker::wxCriticalSectionLocker, 128
 wxCursor, 129, 130
 wxCursor::~wxCursor, 131
 wxCursor::Ok, 131
 wxCursor::operator !=, 132
 wxCursor::operator =, 132
 wxCursor::operator ==, 132
 wxCursor::wxCursor, 129
 wxDatabase, 133
 wxDatabase overview, 925
 wxDatabase::~wxDatabase, 133
 wxDatabase::BeginTrans, 133
 wxDatabase::Cancel, 133
 wxDatabase::CanTransact, 133
 wxDatabase::CanUpdate, 133
 wxDatabase::Close, 133
 wxDatabase::CommitTrans, 134
 wxDatabase::ErrorOccured, 134
 wxDatabase::ErrorSnapshot, 134
 wxDatabase::GetDatabaseName, 134
 wxDatabase::GetDataSource, 134
 wxDatabase::GetErrorClass, 134
 wxDatabase::GetErrorCode, 134
 wxDatabase::GetErrorMessage, 135
 wxDatabase::GetErrorNumber, 135
 wxDatabase::GetHDBC, 135
 wxDatabase::GetHENV, 135
 wxDatabase::GetInfo, 135
 wxDatabase::GetODBCVersionFloat, 136
 wxDatabase::GetODBCVersionString, 136
 wxDatabase::GetPassword, 136
 wxDatabase::GetUsername, 136
 wxDatabase::InWaitForDataSource, 137
 wxDatabase::IsOpen, 137
 wxDatabase::OnSetOptions, 137
 wxDatabase::OnWaitForDataSource, 137
 wxDatabase::Open, 137
 wxDatabase::RollbackTrans, 137
 wxDatabase::SetDataSource, 137
 wxDatabase::SetLoginTimeout, 138
 wxDatabase::SetPassword, 138
 wxDatabase::SetQueryTimeout, 138
 wxDatabase::SetSynchronousMode, 138
 wxDatabase::SetUsername, 138
 wxDatabase::wxDatabase, 132
 wxDataInputStream, 140, 142
 wxDataInputStream::~wxDataInputStream, 141
 wxDataInputStream::Read16, 141
 wxDataInputStream::Read32, 141
 wxDataInputStream::Read8, 141
 wxDataInputStream::ReadDouble, 141
 wxDataInputStream::ReadLine, 141
 wxDataInputStream::ReadString, 141, 142
 wxDataInputStream::wxDataInputStream, 140
 wxDataObject, 139
 wxDataObject::~wxDataObject, 140
 wxDataObject::GetSize, 140
 wxDataObject::WriteData, 140
 wxDataObject::wxDataObject, 139
 wxDataOutputStream::~wxDataOutputStream, 142
 wxDataOutputStream::Write16, 142
 wxDataOutputStream::Write32, 142
 wxDataOutputStream::Write8, 142
 wxDataOutputStream::WriteDouble, 143
 wxDataOutputStream::WriteLine, 143
 wxDataOutputStream::WriteString, 143
 wxDataOutputStream::wxDataOutputStream, 142
 wxDate, 143, 144, 745
 wxDate::~wxDate, 144
 wxDate::AddMonths, 144
 wxDate::AddWeeks, 145
 wxDate::AddYears, 145
 wxDate::FormatDate, 145
 wxDate::GetDay, 145
 wxDate::GetDayOfWeek, 145
 wxDate::GetDayOfWeekName, 146
 wxDate::GetDayOfYear, 146
 wxDate::GetDaysInMonth, 146
 wxDate::GetFirstDayOfMonth, 146
 wxDate::GetJulianDate, 146
 wxDate::GetMonth, 146
 wxDate::GetMonthEnd, 146
 wxDate::GetMonthName, 147
 wxDate::GetMonthStart, 147
 wxDate::GetWeekOfMonth, 147
 wxDate::GetWeekOfYear, 147
 wxDate::GetYear, 147
 wxDate::GetYearEnd, 147
 wxDate::GetYearStart, 147
 wxDate::IsLeapYear, 147
 wxDate::operator -, 149
 wxDate::operator --, 150
 wxDate::operator !=, 151

wxDate::operator +, 149
wxDate::operator ++, 150
wxDate::operator +=, 149
wxDate::operator <, 150
wxDate::operator <=, 151
wxDate::operator <=, 150
wxDate::operator -=, 150
wxDate::operator ==, 151
wxDate::operator >, 150
wxDate::operator >=, 150
wxDate::operator wxString, 149
wxDate::Set, 148
wxDate::SetFormat, 148
wxDate::SetOption, 148
wxDate::wxDate, 143
wxDC, 152
wxDC::~~wxDC, 152
wxDC::BeginDrawing, 152
wxDC::Blit, 152
wxDC::Clear, 153
wxDC::CrossHair, 153
wxDC::DestroyClippingRegion, 154
wxDC::DeviceToLogicalX, 154
wxDC::DeviceToLogicalXRel, 154
wxDC::DeviceToLogicalY, 154
wxDC::DeviceToLogicalYRel, 154
wxDC::DrawArc, 154
wxDC::DrawBitmap, 155
wxDC::DrawEllipse, 155
wxDC::DrawEllipticArc, 155
wxDC::DrawIcon, 155
wxDC::DrawLine, 155
wxDC::DrawLines, 156
wxDC::DrawPoint, 156
wxDC::DrawPolygon, 156
wxDC::DrawRectangle, 157
wxDC::DrawRoundedRectangle, 157
wxDC::DrawSpline, 157
wxDC::DrawText, 157
wxDC::EndDoc, 158
wxDC::EndDrawing, 158
wxDC::EndPage, 158
wxDC::FloodFill, 158
wxDC::GetBackground, 158
wxDC::GetBrush, 158
wxDC::GetCharHeight, 159
wxDC::GetCharWidth, 159
wxDC::GetClippingBox, 159
wxDC::GetFont, 159
wxDC::GetLogicalFunction, 159
wxDC::GetMapMode, 159
wxDC::GetOptimization, 160
wxDC::GetPen, 160
wxDC::GetPixel, 160
wxDC::GetSize, 160
wxDC::GetTextBackground, 160
wxDC::GetTextExtent, 161
wxDC::GetTextForeground, 161
wxDC::LogicalToDeviceX, 161
wxDC::LogicalToDeviceXRel, 161
wxDC::LogicalToDeviceY, 162
wxDC::LogicalToDeviceYRel, 162
wxDC::MaxX, 162
wxDC::MaxY, 162
wxDC::MinX, 162
wxDC::MinY, 162
wxDC::Ok, 162
wxDC::SetBackground, 163
wxDC::SetBackgroundMode, 163
wxDC::SetBrush, 164
wxDC::SetClippingRegion, 163
wxDC::SetDeviceOrigin, 163
wxDC::SetFont, 164
wxDC::SetLogicalFunction, 164
wxDC::SetMapMode, 165
wxDC::SetOptimization, 165
wxDC::SetPalette, 163
wxDC::SetPen, 165
wxDC::SetTextBackground, 166
wxDC::SetTextForeground, 166
wxDC::SetUserScale, 166
wxDC::StartDoc, 166
wxDC::StartPage, 166
wxDC::wxDC, 151
wxDDECleanUp, 865
wxDDEClient, 167
wxDDEClient::MakeConnection, 167
wxDDEClient::OnMakeConnection, 167
wxDDEClient::ValidHost, 168
wxDDEClient::wxDDEClient, 167
wxDDEConnection, 169
wxDDEConnection::Advise, 169
wxDDEConnection::Disconnect, 170
wxDDEConnection::Execute, 169
wxDDEConnection::OnAdvise, 170
wxDDEConnection::OnDisconnect, 170
wxDDEConnection::OnExecute, 170
wxDDEConnection::OnPoke, 170
wxDDEConnection::OnRequest, 171
wxDDEConnection::OnStartAdvise, 171
wxDDEConnection::OnStopAdvise, 171
wxDDEConnection::Poke, 171
wxDDEConnection::Request, 171
wxDDEConnection::StartAdvise, 172
wxDDEConnection::StopAdvise, 172
wxDDEConnection::wxDDEConnection, 169
wxDDEInitialize, 865
wxDDEServer, 172
wxDDEServer::Create, 172
wxDDEServer::OnAcceptConnection, 173
wxDDEServer::wxDDEServer, 172
WXDEBUG_NEW, 878
wxDebugContext overview, 931
wxDebugContext::Check, 173
wxDebugContext::Dump, 174
wxDebugContext::GetCheckPrevious, 174
wxDebugContext::GetDebugMode, 174
wxDebugContext::GetLevel, 174
wxDebugContext::GetStream, 175
wxDebugContext::GetStreamBuf, 175
wxDebugContext::HasStream, 175
wxDebugContext::PrintClasses, 175
wxDebugContext::PrintStatistics, 176
wxDebugContext::SetCheckpoint, 176

-
- wxDebugContext::SetCheckPrevious, 176
 - wxDebugContext::SetDebugMode, 177
 - wxDebugContext::SetFile, 177
 - wxDebugContext::SetLevel, 177
 - wxDebugContext::SetStandardError, 177
 - wxDebugContext::SetStream, 178
 - wxDebugMsg, 865
 - wxDEFAULT_DIALOG_STYLE, 179
 - wxDEFAULT_FRAME_STYLE, 276, 406, 411, 447
 - wxDialog, 180
 - wxDialog::~wxDialog, 181
 - wxDialog::Centre, 181
 - wxDialog::Create, 181
 - wxDialog::EndModal, 181
 - wxDialog::GetTitle, 181
 - wxDialog::Iconize, 182
 - wxDialog::IsIconized, 182
 - wxDialog::IsModal, 182
 - wxDialog::OnApply, 183
 - wxDialog::OnCancel, 183
 - wxDialog::OnCharHook, 182
 - wxDialog::OnOK, 183
 - wxDialog::OnSysColourChanged, 184
 - wxDialog::SetModal, 184
 - wxDialog::SetTitle, 184
 - wxDialog::Show, 185
 - wxDialog::ShowModal, 185
 - wxDialog::wxDialog, 180
 - wxDirDialog, 186
 - wxDirDialog overview, 920
 - wxDirDialog::~wxDirDialog, 186
 - wxDirDialog::GetMessage, 187
 - wxDirDialog::GetPath, 186
 - wxDirDialog::GetStyle, 187
 - wxDirDialog::SetMessage, 187
 - wxDirDialog::SetPath, 187
 - wxDirDialog::SetStyle, 187
 - wxDirDialog::ShowModal, 187
 - wxDirDialog::wxDirDialog, 186
 - wxDirExists, 847
 - wxDisplayDepth, 858
 - wxDisplaySize, 866
 - wxDocChildFrame, 188
 - wxDocChildFrame::~wxDocChildFrame, 188
 - wxDocChildFrame::GetDocument, 189
 - wxDocChildFrame::GetView, 189
 - wxDocChildFrame::m_childDocument, 188
 - wxDocChildFrame::m_childView, 188
 - wxDocChildFrame::OnActivate, 189
 - wxDocChildFrame::OnCloseWindow, 189
 - wxDocChildFrame::SetDocument, 189
 - wxDocChildFrame::SetView, 189
 - wxDocChildFrame::wxDocChildFrame, 188
 - wxDocManager, 191
 - wxDocManager overview, 939
 - wxDocManager::~wxDocManager, 191
 - wxDocManager::ActivateView, 191
 - wxDocManager::AddDocument, 191
 - wxDocManager::AddFileToHistory, 192
 - wxDocManager::AssociateTemplate, 192
 - wxDocManager::CreateDocument, 192
 - wxDocManager::CreateView, 192
 - wxDocManager::DisassociateTemplate, 192
 - wxDocManager::FileHistoryAddFilesToMenu, 193
 - wxDocManager::FileHistoryLoad, 193
 - wxDocManager::FileHistoryRemoveMenu, 193
 - wxDocManager::FileHistorySave, 193
 - wxDocManager::FileHistoryUseMenu, 193
 - wxDocManager::FindTemplateForPath, 194
 - wxDocManager::GetCurrentDocument, 194
 - wxDocManager::GetCurrentView, 194
 - wxDocManager::GetDocuments, 194
 - wxDocManager::GetFileHistory, 194
 - wxDocManager::GetMaxDocsOpen, 194
 - wxDocManager::GetNoHistoryFiles, 194
 - wxDocManager::Initialize, 195
 - wxDocManager::m_currentView, 190
 - wxDocManager::m_defaultDocumentNameCounter, 190
 - wxDocManager::m_docs, 190
 - wxDocManager::m_fileHistory, 190
 - wxDocManager::m_flags, 191
 - wxDocManager::m_maxDocsOpen, 190
 - wxDocManager::m_templates, 191
 - wxDocManager::MakeDefaultName, 195
 - wxDocManager::OnCreateFileHistory, 195
 - wxDocManager::OnFileClose, 195
 - wxDocManager::OnFileNew, 195
 - wxDocManager::OnFileOpen, 196
 - wxDocManager::OnFileSave, 196
 - wxDocManager::OnFileSaveAs, 196
 - wxDocManager::OnMenuCommand, 196
 - wxDocManager::RemoveDocument, 196
 - wxDocManager::SelectDocumentPath, 196
 - wxDocManager::SelectDocumentType, 197
 - wxDocManager::SelectViewType, 197
 - wxDocManager::SetMaxDocsOpen, 197
 - wxDocManager::wxDocManager, 191
 - wxDocMDIChildFrame, 198
 - wxDocMDIChildFrame::~wxDocMDIChildFrame, 198
 - wxDocMDIChildFrame::GetDocument, 199
 - wxDocMDIChildFrame::GetView, 199
 - wxDocMDIChildFrame::m_childDocument, 198
 - wxDocMDIChildFrame::m_childView, 198
 - wxDocMDIChildFrame::OnActivate, 199
 - wxDocMDIChildFrame::OnCloseWindow, 199
 - wxDocMDIChildFrame::SetDocument, 199
 - wxDocMDIChildFrame::SetView, 199
 - wxDocMDIChildFrame::wxDocMDIChildFrame, 198
 - wxDocMDIParentFrame::~wxDocMDIParentFrame, 200
 - wxDocMDIParentFrame::OnCloseWindow, 200
 - wxDocMDIParentFrame::wxDocMDIParentFrame, 200
 - wxDocParentFrame, 200, 201
 - wxDocParentFrame::~wxDocParentFrame, 201
 - wxDocParentFrame::OnCloseWindow, 202
 - wxDocParentFrame::wxDocParentFrame, 201
 - wxDocTemplate, 204
 - wxDocTemplate overview, 938
-

wxDocTemplate::~wxDocTemplate, 205
 wxDocTemplate::CreateDocument, 205
 wxDocTemplate::CreateView, 205
 wxDocTemplate::GetDefaultExtension, 205
 wxDocTemplate::GetDescription, 205
 wxDocTemplate::GetDirectory, 205
 wxDocTemplate::GetDocumentManager, 206
 wxDocTemplate::GetDocumentName, 206
 wxDocTemplate::GetFileFilter, 206
 wxDocTemplate::GetFlags, 206
 wxDocTemplate::GetViewName, 206
 wxDocTemplate::IsVisible, 206
 wxDocTemplate::m_defaultExt, 202
 wxDocTemplate::m_description, 202
 wxDocTemplate::m_directory, 203
 wxDocTemplate::m_docClassInfo, 203
 wxDocTemplate::m_docTypeName, 203
 wxDocTemplate::m_documentManager, 203
 wxDocTemplate::m_fileFilter, 203
 wxDocTemplate::m_flags, 203
 wxDocTemplate::m_viewClassInfo, 203
 wxDocTemplate::m_viewTypeName, 204
 wxDocTemplate::SetDefaultExtension, 206
 wxDocTemplate::SetDescription, 207
 wxDocTemplate::SetDirectory, 207
 wxDocTemplate::SetDocumentManager, 207
 wxDocTemplate::SetFileFilter, 207
 wxDocTemplate::SetFlags, 207
 wxDocTemplate::wxDocTemplate, 204
 wxDocument, 209
 wxDocument overview, 937
 wxDocument::~wxDocument, 209
 wxDocument::AddView, 209
 wxDocument::Close, 209
 wxDocument::DeleteAllViews, 210
 wxDocument::GetCommandProcessor, 210
 wxDocument::GetDocumentManager, 210
 wxDocument::GetDocumentName, 210
 wxDocument::GetDocumentTemplate, 210
 wxDocument::GetDocumentWindow, 210
 wxDocument::GetFilename, 210
 wxDocument::GetFirstView, 211
 wxDocument::GetPrintableName, 211
 wxDocument::GetTitle, 211
 wxDocument::IsModified, 211
 wxDocument::LoadObject, 211
 wxDocument::m_commandProcessor, 208
 wxDocument::m_documentFile, 208
 wxDocument::m_documentModified, 208
 wxDocument::m_documentTemplate, 208
 wxDocument::m_documentTitle, 208
 wxDocument::m_documentTypeName, 209
 wxDocument::m_documentViews, 209
 wxDocument::Modify, 212
 wxDocument::OnChangedViewList, 212
 wxDocument::OnCloseDocument, 212
 wxDocument::OnCreate, 212
 wxDocument::OnCreateCommandProcessor, 212
 wxDocument::OnNewDocument, 213
 wxDocument::OnOpenDocument, 213
 wxDocument::OnSaveDocument, 213
 wxDocument::OnSaveModified, 213
 wxDocument::RemoveView, 213
 wxDocument::Save, 213
 wxDocument::SaveAs, 214
 wxDocument::SaveObject, 214
 wxDocument::SetCommandProcessor, 214
 wxDocument::SetDocumentName, 214
 wxDocument::SetDocumentTemplate, 214
 wxDocument::SetFilename, 214
 wxDocument::SetTitle, 215
 wxDocument::UpdateAllViews, 215
 wxDocument::wxDocument, 209
 wxDOUBLE_BORDER, 801
 wxDragResult, 217
 wxDropFilesEvent, 216
 wxDropFilesEvent::GetFiles, 216
 wxDropFilesEvent::GetNumberOfFiles, 216
 wxDropFilesEvent::GetPosition, 216
 wxDropFilesEvent::m_files, 216
 wxDropFilesEvent::m_noFiles, 216
 wxDropFilesEvent::m_pos, 216
 wxDropFilesEvent::wxDropFilesEvent, 216
 wxDropSource, 217
 wxDropSource::~wxDropSource, 218
 wxDropSource::DoDragDrop, 218
 wxDropSource::GiveFeedback, 218
 wxDropSource::SetData, 218
 wxDropSource::wxDropSource, 217
 wxDropTarget, 219
 wxDropTarget::~wxDropTarget, 219
 wxDropTarget::GetFormat, 219
 wxDropTarget::GetFormatCount, 219
 wxDropTarget::OnDrop, 220
 wxDropTarget::OnEnter, 220
 wxDropTarget::OnLeave, 220
 wxDropTarget::wxDropTarget, 219
 wxEdge, 344
 wxEmptyClipboard, 862
 wxEndBusyCursor, 867
 wxEntry, 866
 wxEnumClipboardFormats, 862
 wxEraseEvent, 221
 wxEraseEvent::GetDC, 221
 wxEraseEvent::m_dc, 221
 wxEraseEvent::wxEraseEvent, 221
 wxError, 867
 wxEvent, 222
 wxEvent::GetEventClass, 223
 wxEvent::GetEventObject, 223
 wxEvent::GetEventType, 223
 wxEvent::GetId, 223
 wxEvent::GetObjectType, 223
 wxEvent::GetSkipped, 224
 wxEvent::GetTimestamp, 224
 wxEvent::m_eventHandle, 222
 wxEvent::m_eventObject, 222
 wxEvent::m_eventType, 222
 wxEvent::m_id, 222
 wxEvent::m_skipped, 223
 wxEvent::m_timeStamp, 223
 wxEvent::SetEventObject, 224
 wxEvent::SetEventType, 224

wxEvtHandler::SetId, 224
wxEvtHandler::SetTimestamp, 224
wxEvtHandler::Skip, 224
wxEvtHandler::wxEvtHandler, 222
wxEvtHandler::~wxEvtHandler, 225
wxEvtHandler::Connect, 225
wxEvtHandler::Default, 226
wxEvtHandler::GetClientData, 226
wxEvtHandler::GetEvtHandlerEnabled, 227
wxEvtHandler::GetNextHandler, 227
wxEvtHandler::GetPreviousHandler, 227
wxEvtHandler::ProcessEvent, 227
wxEvtHandler::SearchEventTable, 229
wxEvtHandler::SetClientData, 229
wxEvtHandler::SetEvtHandlerEnabled, 230
wxEvtHandler::SetNextHandler, 230
wxEvtHandler::SetPreviousHandler, 230
wxEvtHandler::wxEvtHandler, 225
wxExecute, 867
wxExit, 868
wxExpr, 231, 232
wxExpr compilation, 976
wxExpr for data file manipulation, 974
wxExpr::~wxEvtHandler, 232
wxExpr::AddAttributeValue, 232
wxExpr::AddAttributeValueString, 232
wxExpr::AddAttributeValueStringList, 233
wxExpr::AddAttributeValueWord, 233
wxExpr::Append, 233
wxExpr::Arg, 233
wxExpr::AttributeValue, 234
wxExpr::Copy, 234
wxExpr::DeleteAttributeValue, 235
wxExpr::Functor, 235
wxExpr::GetAttributeValue, 233
wxExpr::GetAttributeValueStringList, 234
wxExpr::GetClientData, 235
wxExpr::GetFirst, 235
wxExpr::GetLast, 235
wxExpr::GetNext, 235
wxExpr::Insert, 233
wxExpr::IntegerValue, 236
wxExpr::Nth, 236
wxExpr::RealValue, 236
wxExpr::SetClientData, 236
wxExpr::StringValue, 236
wxExpr::Type, 236
wxExpr::WordValue, 237
wxExpr::WriteExpr, 237
wxExpr::WriteLispExpr, 237
wxExpr::WritePrologClause, 237
wxExpr::wxExpr, 231
wxExprCleanUp, 238
wxExprDatabase, 238, 239
wxExprDatabase::~wxEvtHandler, 239
wxExprDatabase::Append, 239
wxExprDatabase::BeginFind, 239
wxExprDatabase::ClearDatabase, 239
wxExprDatabase::FindClause, 239
wxExprDatabase::FindClauseByFunctor, 240
wxExprDatabase::GetErrorCount, 240
wxExprDatabase::HashFind, 240
wxExprDatabase::Read, 241
wxExprDatabase::ReadFromString, 241
wxExprDatabase::Write, 241
wxExprDatabase::WriteLisp, 241
wxExprDatabase::wxExprDatabase, 238
wxExprIsFunctor, 237
wxFAIL, 889
wxFAIL_MSG, 889
wxFatalError, 868
wxFile, 243
wxFile::~wxEvtHandler, 243
wxFile::Access, 243
wxFile::Attach, 244
wxFile::Close, 244
wxFile::Create, 244
wxFile::Detach, 244
wxFile::Eof, 244
wxFile::Exists, 245
wxFile::fd, 244
wxFile::Flush, 245
wxFile::IsOpened, 245
wxFile::Length, 245
wxFile::Open, 245
wxFile::Read, 246
wxFile::Seek, 246
wxFile::SeekEnd, 246
wxFile::Tell, 247
wxFile::Write, 247
wxFile::wxFile, 243
wxFileDataObject, 248
wxFileDataObject::AddFile, 248
wxFileDataObject::GetFiles, 248
wxFileDataObject::GetFormat, 248
wxFileDataObject::wxFileDataObject, 248
wxFileDialog, 249
wxFileDialog overview, 919
wxFileDialog::~wxEvtHandler, 250
wxFileDialog::GetDirectory, 250
wxFileDialog::GetFilename, 250
wxFileDialog::GetFilterIndex, 250
wxFileDialog::GetMessage, 250
wxFileDialog::GetPath, 251
wxFileDialog::GetStyle, 251
wxFileDialog::GetWildcard, 251
wxFileDialog::SetDirectory, 251
wxFileDialog::SetFilename, 251
wxFileDialog::SetFilterIndex, 251
wxFileDialog::SetMessage, 251
wxFileDialog::SetPath, 252
wxFileDialog::SetStyle, 252
wxFileDialog::SetWildcard, 252
wxFileDialog::ShowModal, 252
wxFileDialog::wxFileDialog, 249
wxFileDropTarget, 253
wxFileDropTarget::GetFormat, 253
wxFileDropTarget::GetFormatCount, 253
wxFileDropTarget::OnDrop, 253
wxFileDropTarget::OnDropFiles, 253
wxFileDropTarget::wxFileDropTarget, 253
wxFileExists, 847
wxFileHistory, 255

- wxFileHistory overview, 940
- wxFileHistory::~~wxFileHistory, 255
- wxFileHistory::AddFilesToMenu, 255
- wxFileHistory::AddFileToHistory, 255
- wxFileHistory::GetHistoryFile, 255
- wxFileHistory::GetMaxFiles, 255
- wxFileHistory::GetNoHistoryFiles, 256
- wxFileHistory::Load, 256
- wxFileHistory::m_fileHistory, 254
- wxFileHistory::m_fileHistoryN, 254
- wxFileHistory::m_fileMaxFiles, 254
- wxFileHistory::m_fileMenu, 255
- wxFileHistory::RemoveMenu, 256
- wxFileHistory::Save, 256
- wxFileHistory::UseMenu, 256
- wxFileHistory::wxFileHistory, 255
- wxFileInputStream, 257
- wxFileInputStream::~~wxFileInputStream, 257
- wxFileInputStream::Ok, 257
- wxFileInputStream::wxFileInputStream, 257
- wxFileNameFromPath, 847
- wxFileOutputStream, 258
- wxFileOutputStream::~~wxFileOutputStream, 258
- wxFileOutputStream::Ok, 258
- wxFileOutputStream::wxFileOutputStream, 258
- wxFileSelector, 854
- wxFileStream, 259
- wxFileStream::wxFileStream, 259
- wxFileType, 261
- wxFileType::~~wxFileType, 261
- wxFileType::ExpandCommand, 262
- wxFileType::GetDescription, 262
- wxFileType::GetExtensions, 261
- wxFileType::GetIcon, 261
- wxFileType::GetMimeType, 261
- wxFileType::GetOpenCommand, 262
- wxFileType::GetPrintCommand, 262
- wxFileType::wxFileType, 261
- wxFilterInputStream, 263
- wxFilterInputStream::wxFilterInputStream, 263
- wxFilterOutputStream, 263
- wxFilterOutputStream::wxFilterOutputStream, 263
- wxFindFirstFile, 848
- wxFindMenuItemId, 869
- wxFindWindowByLabel, 869
- wxFindWindowByName, 869
- wxFocusEvent, 264
- wxFocusEvent::wxFocusEvent, 264
- wxFont, 265
- wxFont::~~wxFont, 266
- wxFont::GetFaceName, 266
- wxFont::GetFamily, 267
- wxFont::GetFontId, 267
- wxFont::GetPointSize, 267
- wxFont::GetStyle, 267
- wxFont::GetUnderlined, 267
- wxFont::GetWeight, 268
- wxFont::operator !=, 270
- wxFont::operator =, 270
- wxFont::operator ==, 270
- wxFont::SetFaceName, 268
- wxFont::SetFamily, 268
- wxFont::SetPointSize, 269
- wxFont::SetStyle, 269
- wxFont::SetUnderlined, 269
- wxFont::SetWeight, 270
- wxFont::wxFont, 265
- wxFontData, 271
- wxFontData::~~wxFontData, 271
- wxFontData::EnableEffects, 271
- wxFontData::GetAllowSymbols, 271
- wxFontData::GetChosenFont, 272
- wxFontData::GetColour, 272
- wxFontData::GetEnableEffects, 272
- wxFontData::GetInitialFont, 272
- wxFontData::GetShowHelp, 272
- wxFontData::operator =, 274
- wxFontData::SetAllowSymbols, 273
- wxFontData::SetChosenFont, 273
- wxFontData::SetColour, 273
- wxFontData::SetInitialFont, 273
- wxFontData::SetRange, 273
- wxFontData::SetShowHelp, 273
- wxFontData::wxFontData, 271
- wxFontDialog, 274
- wxFontDialog overview, 918
- wxFontDialog::~~wxFontDialog, 274
- wxFontDialog::GetFontData, 274
- wxFontDialog::ShowModal, 275
- wxFontDialog::wxFontDialog, 274
- wxFontList, 275
- wxFontList::AddFont, 275
- wxFontList::FindOrCreateFont, 276
- wxFontList::RemoveFont, 276
- wxFontList::wxFontList, 275
- wxFrame, 277
- wxFrame::~~wxFrame, 278
- wxFrame::Centre, 278
- wxFrame::Command, 279
- wxFrame::Create, 279
- wxFrame::CreateStatusBar, 279
- wxFrame::CreateToolBar, 280
- wxFrame::GetMenuBar, 281
- wxFrame::GetStatusBar, 281
- wxFrame::GetTitle, 281
- wxFrame::GetToolBar, 281
- wxFrame::Iconize, 281
- wxFrame::IsIconized, 282
- wxFrame::IsMaximized, 282
- wxFrame::Maximize, 282
- wxFrame::OnActivate, 282
- wxFrame::OnCreateStatusBar, 283
- wxFrame::OnCreateToolBar, 283
- wxFrame::OnMenuCommand, 284
- wxFrame::OnMenuHighlight, 284
- wxFrame::OnSize, 284
- wxFrame::SetIcon, 284
- wxFrame::SetMenuBar, 285
- wxFrame::SetStatusBar, 286
- wxFrame::SetStatusText, 286
- wxFrame::SetStatusWidths, 286
- wxFrame::SetTitle, 287
- wxFrame::SetToolBar, 287

wxFrame::wxFrame, 277
wxFTP::ChDir, 288
wxFTP::GetInputStream, 291
wxFTP::GetLastResult, 288
wxFTP::GetList, 290
wxFTP::GetOutputStream, 290
wxFTP::MkDir, 288
wxFTP::Pwd, 289
wxFTP::Rename, 289
wxFTP::Rmdir, 288
wxFTP::RmFile, 289
wxFTP::SendCommand, 288
wxFTP::SetPassword, 289
wxFTP::SetUser, 289
wxGA_HORIZONTAL, 292
wxGA_PROGRESSBAR, 292
wxGA_VERTICAL, 292
wxGauge, 292
wxGauge::~wxGauge, 293
wxGauge::Create, 293
wxGauge::GetBezelFace, 293
wxGauge::GetRange, 294
wxGauge::GetShadowWidth, 294
wxGauge::GetValue, 294
wxGauge::SetBezelFace, 294
wxGauge::SetRange, 295
wxGauge::SetShadowWidth, 295
wxGauge::SetValue, 295
wxGauge::wxGauge, 292
wxGDIObject, 296
wxGDIObject::wxGDIObject, 296
wxGenericValidator, 297
wxGenericValidator::~wxGenericValidator, 297
wxGenericValidator::Clone, 298
wxGenericValidator::TransferFromWindow, 298
wxGenericValidator::TransferToWindow, 298
wxGenericValidator::wxGenericValidator, 297
wxGetActiveWindow, 870
wxGetClipboardData, 862
wxGetClipboardFormatName, 863
wxGetCwd, 849
wxGetDisplayName, 870
wxGetElapsedTime, 870
wxGetEmailAddress, 849
wxGetFreeMemory, 871
wxGetHomeDir, 870
wxGetHostName, 849, 870
wxGetMousePosition, 871
wxGetMultipleChoice, 855
wxGetOSDirectory, 848
wxGetOsVersion, 871
wxGetPrinterCommand, 859
wxGetPrinterFile, 859
wxGetPrinterMode, 859
wxGetPrinterOptions, 859
wxGetPrinterOrientation, 860
wxGetPrinterPreviewCommand, 860
wxGetPrinterScaling, 860
wxGetPrinterTranslation, 860
wxGetResource, 872
wxGetSingleChoice, 856
wxGetSingleChoiceData, 856
wxGetSingleChoiceIndex, 856
wxGetTempFileName, 850
wxGetTextFromUser, 855
wxGetTranslation, 854
wxGetUserId, 850, 872
wxGetUserName, 850, 873
wxGetWorkingDirectory, 850
wxGrid, 299
wxGrid::AdjustScrollbars, 299
wxGrid::AppendCols, 299
wxGrid::AppendRows, 299
wxGrid::BeginBatch, 299
wxGrid::CellHitTest, 299
wxGrid::CreateGrid, 300
wxGrid::CurrentCellVisible, 300
wxGrid::DeleteCols, 300
wxGrid::DeleteRows, 300
wxGrid::EndBatch, 300
wxGrid::GetBatchCount, 301
wxGrid::GetCell, 301
wxGrid::GetCellAlignment, 301
wxGrid::GetCellBackgroundColour, 301
wxGrid::GetCells, 302
wxGrid::GetCellTextColour, 302
wxGrid::GetCellTextFont, 302
wxGrid::GetCellValue, 303
wxGrid::GetCols, 303
wxGrid::GetColumnWidth, 303
wxGrid::GetCurrentRect, 303
wxGrid::GetCursorColumn, 303
wxGrid::GetCursorRow, 303
wxGrid::GetEditable, 303
wxGrid::GetHorizScrollBar, 304
wxGrid::GetLabelAlignment, 304
wxGrid::GetLabelBackgroundColour, 304
wxGrid::GetLabelSize, 304
wxGrid::GetLabelTextColour, 304
wxGrid::GetLabelTextFont, 304
wxGrid::GetLabelValue, 305
wxGrid::GetRowHeight, 305
wxGrid::GetRows, 305
wxGrid::GetScrollPosX, 305
wxGrid::GetScrollPosY, 305
wxGrid::GetTextItem, 305
wxGrid::GetVertScrollBar, 305
wxGrid::InsertCols, 306
wxGrid::InsertRows, 306
wxGrid::OnActivate, 306
wxGrid::OnCellLeftClick, 307
wxGrid::OnCellRightClick, 307
wxGrid::OnChangeLabels, 306
wxGrid::OnChangeSelectionLabel, 306
wxGrid::OnCreateCell, 306
wxGrid::OnLabelLeftClick, 307
wxGrid::OnLabelRightClick, 307
wxGrid::OnSelectCell, 307
wxGrid::OnSelectCellImplementation, 308
wxGrid::SetCellAlignment, 308
wxGrid::SetCellBackgroundColour, 308
wxGrid::SetCellTextColour, 308
wxGrid::SetCellTextFont, 309
wxGrid::SetCellValue, 309

wxGrid::SetColumnWidth, 309
wxGrid::SetDividerPen, 309
wxGrid::SetEditable, 309
wxGrid::SetGridCursor, 310
wxGrid::SetLabelAlignment, 310
wxGrid::SetLabelBackgroundColour, 310
wxGrid::SetLabelSize, 310
wxGrid::SetLabelTextColour, 310
wxGrid::SetLabelTextFont, 310
wxGrid::SetLabelValue, 311
wxGrid::SetRowHeight, 311
wxGrid::UpdateDimensions, 311
wxGrid::wxGrid, 298
wxHashTable, 312
wxHashTable::~wxHashTable, 312
wxHashTable::BeginFind, 312
wxHashTable::Clear, 312
wxHashTable::Delete, 313
wxHashTable::Get, 313
wxHashTable::MakeKey, 313
wxHashTable::Next, 313
wxHashTable::Put, 313
wxHashTable::wxHashTable, 312
wxHelpController, 314
wxHelpController::~wxHelpController, 314
wxHelpController::DisplayBlock, 315
wxHelpController::DisplayContents, 315
wxHelpController::DisplaySection, 315
wxHelpController::Initialize, 315
wxHelpController::KeywordSearch, 316
wxHelpController::LoadFile, 316
wxHelpController::OnQuit, 316
wxHelpController::Quit, 317
wxHelpController::SetViewer, 316
wxHelpController::wxHelpController, 314
wxHSCROLL, 411, 714, 801
wxHTTP::GetInputStream, 317
wxHTTP::SetHeader, 318
wxIcon, 320, 321
wxIcon::~wxIcon, 323
wxIcon::GetDepth, 323
wxIcon::GetHeight, 323
wxIcon::GetWidth, 323
wxIcon::LoadFile, 323
wxIcon::Ok, 324
wxIcon::operator !=, 326
wxIcon::operator =, 325
wxIcon::operator ==, 326
wxIcon::SetDepth, 324
wxIcon::SetHeight, 324
wxIcon::SetOk, 325
wxIcon::SetWidth, 325
wxIcon::wxIcon, 320
wxICONIZE, 276, 406, 411, 447
wxID, 945
wxIdleEvent, 319
wxIdleEvent::MoreRequested, 319
wxIdleEvent::RequestMore, 319
wxIdleEvent::wxIdleEvent, 318
wxImage, 327
wxImage::~wxImage, 328
wxImage::AddHandler, 328
wxImage::CleanUpHandlers, 328
wxImage::ConvertToBitmap, 329
wxImage::Create, 329
wxImage::Destroy, 329
wxImage::FindHandler, 329
wxImage::GetBlue, 330
wxImage::GetData, 330
wxImage::GetGreen, 330
wxImage::GetHandlers, 331
wxImage::GetHeight, 331
wxImage::GetMaskBlue, 331
wxImage::GetMaskGreen, 331
wxImage::GetMaskRed, 331
wxImage::GetRed, 330
wxImage::GetWidth, 331
wxImage::HasMask, 332
wxImage::InitStandardHandlers, 332
wxImage::InsertHandler, 332
wxImage::LoadFile, 332
wxImage::Ok, 333
wxImage::operator !=, 336
wxImage::operator =, 335
wxImage::operator ==, 336
wxImage::RemoveHandler, 333
wxImage::SaveFile, 334
wxImage::Scale, 334
wxImage::SetData, 335
wxImage::SetMask, 335
wxImage::SetMaskColour, 335
wxImage::SetRGB, 335
wxImage::wxImage, 327
wxIMAGE_LIST_NORMAL, 387, 768
wxIMAGE_LIST_SMALL, 387, 768
wxIMAGE_LIST_STATE, 387, 768
wxImageHandler, 337
wxImageHandler::~wxImageHandler, 337
wxImageHandler::GetExtension, 337
wxImageHandler::GetName, 337
wxImageHandler::GetType, 337
wxImageHandler::LoadFile, 338
wxImageHandler::SaveFile, 338
wxImageHandler::SetExtension, 339
wxImageHandler::SetName, 339
wxImageHandler::SetType, 339
wxImageHandler::wxImageHandler, 337
wxImageList, 340
wxImageList::Add, 340
wxImageList::Create, 341
wxImageList::Draw, 341
wxImageList::GetImageCount, 342
wxImageList::Remove, 342
wxImageList::RemoveAll, 342
wxImageList::Replace, 342
wxImageList::wxImageList, 340
wxIndividualLayoutConstraint, 344
wxIndividualLayoutConstraint::Above, 344
wxIndividualLayoutConstraint::Absolute, 345
wxIndividualLayoutConstraint::AsIs, 345
wxIndividualLayoutConstraint::Below, 345
wxIndividualLayoutConstraint::LeftOf, 345
wxIndividualLayoutConstraint::PercentOf, 345
wxIndividualLayoutConstraint::RightOf, 346

wxIndividualLayoutConstraint::SameAs, 346
 wxIndividualLayoutConstraint::Set, 346
 wxIndividualLayoutConstraint::Unconstrained, 345
 wxIndividualLayoutConstraint::wxIndividualLayoutConstraint, 344
 wxInitDialogEvent, 347
 wxInitDialogEvent::wxInitDialogEvent, 347
 wxInputStream, 347
 wxInputStream::~wxInputStream, 347
 wxInputStream::GetC, 347
 wxInputStream::InputStreamBuffer, 348
 wxInputStream::LastRead, 348
 wxInputStream::Peek, 348
 wxInputStream::Read, 348
 wxInputStream::SeekI, 349
 wxInputStream::TellI, 349
 wxInputStream::wxInputStream, 347
 wxIPCFFormat, 168, 707
 wxIPv4address::Hostname, 349
 wxIPv4address::LocalHost, 350
 wxIPv4address::Service, 349, 350
 wxIsAbsolutePath, 848
 wxIsBusy, 873
 wxIsClipboardFormatAvailable, 863
 wxIsWild, 851
 wxJoystick, 351
 wxJoystick::~~wxJoystick, 351
 wxJoystick::GetButtonState, 351
 wxJoystick::GetManufacturerId, 351
 wxJoystick::GetMovementThreshold, 351
 wxJoystick::GetNumberAxes, 351
 wxJoystick::GetNumberButtons, 351
 wxJoystick::GetNumberJoysticks, 352
 wxJoystick::GetPollingMax, 352
 wxJoystick::GetPollingMin, 352
 wxJoystick::GetPosition, 352
 wxJoystick::GetPOVCTSPosition, 353
 wxJoystick::GetPOVPosition, 352
 wxJoystick::GetProductId, 352
 wxJoystick::GetProductName, 352
 wxJoystick::GetRudderMax, 353
 wxJoystick::GetRudderMin, 353
 wxJoystick::GetRudderPosition, 353
 wxJoystick::GetUMax, 353
 wxJoystick::GetUMin, 353
 wxJoystick::GetUPosition, 353
 wxJoystick::GetVMax, 354
 wxJoystick::GetVMin, 354
 wxJoystick::GetVPosition, 354
 wxJoystick::GetXMax, 354
 wxJoystick::GetXMin, 354
 wxJoystick::GetYMax, 354
 wxJoystick::GetYMin, 354
 wxJoystick::GetZMax, 355
 wxJoystick::GetZMin, 355
 wxJoystick::GetZPosition, 355
 wxJoystick::HasPOV, 355
 wxJoystick::HasPOV4Dir, 355
 wxJoystick::HasPOVCTS, 355
 wxJoystick::HasRudder, 355
 wxJoystick::HasU, 356
 wxJoystick::HasV, 356
 wxJoystick::HasZ, 356
 wxJoystick::IsOk, 356
 wxJoystick::ReleaseCapture, 356
 wxJoystick::SetCapture, 356
 wxJoystick::SetMovementThreshold, 357
 wxJoystick::wxJoystick, 351
 wxJoystickEvent, 358
 wxJoystickEvent::ButtonDown, 358
 wxJoystickEvent::ButtonIsDown, 358
 wxJoystickEvent::ButtonUp, 358
 wxJoystickEvent::GetButtonChange, 359
 wxJoystickEvent::GetButtonState, 359
 wxJoystickEvent::GetJoystick, 359
 wxJoystickEvent::GetPosition, 359
 wxJoystickEvent::GetZPosition, 359
 wxJoystickEvent::IsButton, 359
 wxJoystickEvent::IsMove, 359
 wxJoystickEvent::IsZMove, 360
 wxJoystickEvent::wxJoystickEvent, 358
 wxKeyEvent, 361
 wxKeyEvent::AltDown, 362
 wxKeyEvent::ControlDown, 362
 wxKeyEvent::GetX, 362
 wxKeyEvent::GetY, 362
 wxKeyEvent::KeyCode, 362
 wxKeyEvent::m_altDown, 360
 wxKeyEvent::m_controlDown, 361
 wxKeyEvent::m_keyCode, 361
 wxKeyEvent::m_metaDown, 361
 wxKeyEvent::m_shiftDown, 361
 wxKeyEvent::m_x, 361
 wxKeyEvent::m_y, 361
 wxKeyEvent::MetaDown, 362
 wxKeyEvent::Position, 362
 wxKeyEvent::ShiftDown, 363
 wxKeyEvent::wxKeyEvent, 361
 wxKill, 873
 wxLayoutAlgorithm, 365
 wxLayoutAlgorithm::~~wxLayoutAlgorithm, 365
 wxLayoutAlgorithm::LayoutFrame, 365
 wxLayoutAlgorithm::LayoutMDIFrame, 365
 wxLayoutAlgorithm::LayoutWindow, 365
 wxLayoutAlgorithm::wxLayoutAlgorithm, 365
 wxLayoutConstraints, 366
 wxLayoutConstraints::bottom, 367
 wxLayoutConstraints::centreX, 367
 wxLayoutConstraints::centreY, 367
 wxLayoutConstraints::height, 367
 wxLayoutConstraints::left, 367
 wxLayoutConstraints::right, 367
 wxLayoutConstraints::top, 367
 wxLayoutConstraints::width, 368
 wxLayoutConstraints::wxLayoutConstraints, 366
 wxLB_ALWAYS_SB, 374
 wxLB_EXTENDED, 374
 wxLB_HSCROLL, 374
 wxLB_MULTIPLE, 374
 wxLB_NEEDED_SB, 374
 wxLB_SINGLE, 374
 wxLB_SORT, 374
 wxLC_ALIGN_LEFT, 382

wxLC_ALIGN_TOP, 382
wxLC_AUTOARRANGE, 382
wxLC_EDIT_LABELS, 382
wxLC_ICON, 382
wxLC_LIST, 382
wxLC_NO_HEADER, 383
wxLC_REPORT, 382
wxLC_SINGLE_SEL, 383
wxLC_SMALL_ICON, 382
wxLC_SORT_ASCENDING, 383
wxLC_SORT_DESCENDING, 383
wxLC_USER_TEXT, 382
wxList, 370
wxList::~~wxList, 370
wxList::Append, 371
wxList::Clear, 371
wxList::DeleteContents, 371
wxList::DeleteNode, 371
wxList::DeleteObject, 371
wxList::Find, 372
wxList::GetFirst, 372
wxList::GetLast, 372
wxList::IndexOf, 372
wxList::Insert, 372
wxList::Member, 372
wxList::Nth, 372
wxList::Number, 373
wxList::Sort, 373
wxList::wxList, 370
wxListBox, 375
wxListBox::~~wxListBox, 376
wxListBox::Append, 376
wxListBox::Clear, 376
wxListBox::Create, 376
wxListBox::Delete, 376
wxListBox::Deselect, 377
wxListBox::FindString, 377
wxListBox::GetClientData, 377
wxListBox::GetSelection, 378
wxListBox::GetSelections, 378
wxListBox::GetString, 379
wxListBox::GetStringSelection, 379
wxListBox::InsertItems, 379
wxListBox::Number, 379
wxListBox::Selected, 380
wxListBox::Set, 380
wxListBox::SetClientData, 380
wxListBox::SetFirstItem, 381
wxListBox::SetSelection, 381
wxListBox::SetString, 381
wxListBox::SetStringSelection, 382
wxListBox::wxListBox, 374
wxListCtrl, 383
wxListCtrl::~~wxListCtrl, 384
wxListCtrl::Arrange, 384
wxListCtrl::Create, 385
wxListCtrl::DeleteAllItems, 385
wxListCtrl::DeleteColumn, 385
wxListCtrl::DeleteItem, 385
wxListCtrl::Edit, 385
wxListCtrl::EnsureVisible, 385
wxListCtrl::FindItem, 385
wxListCtrl::GetColumn, 386
wxListCtrl::GetColumnWidth, 386
wxListCtrl::GetCountPerPage, 386
wxListCtrl::GetEditControl, 386
wxListCtrl::GetImageList, 387
wxListCtrl::GetItem, 387
wxListCtrl::GetItemCount, 388
wxListCtrl::GetItemData, 387
wxListCtrl::GetItemPosition, 387
wxListCtrl::GetItemRect, 387
wxListCtrl::GetItemSpacing, 388
wxListCtrl::GetItemState, 388
wxListCtrl::GetItemText, 388
wxListCtrl::GetNextItem, 388
wxListCtrl::GetSelectedItemCount, 389
wxListCtrl::GetTextColour, 389
wxListCtrl::GetTopItem, 389
wxListCtrl::HitTest, 389
wxListCtrl::InsertColumn, 390
wxListCtrl::InsertItem, 390
wxListCtrl::ScrollList, 391
wxListCtrl::SetBackgroundColour, 391
wxListCtrl::SetColumn, 391
wxListCtrl::SetColumnWidth, 392
wxListCtrl::SetImageList, 392
wxListCtrl::SetItem, 392
wxListCtrl::SetItemData, 393
wxListCtrl::SetItemImage, 393
wxListCtrl::SetItemPosition, 393
wxListCtrl::SetItemState, 394
wxListCtrl::SetItemText, 394
wxListCtrl::SetSingleStyle, 394
wxListCtrl::SetTextColour, 394
wxListCtrl::SetWindowStyleFlag, 394
wxListCtrl::SortItems, 394
wxListCtrl::wxListCtrl, 383
wxListEvent, 396
wxListEvent::m_cancelled, 396
wxListEvent::m_code, 396
wxListEvent::m_col, 396
wxListEvent::m_item, 397
wxListEvent::m_itemIndex, 396
wxListEvent::m_oldItemIndex, 396
wxListEvent::m_pointDrag, 396
wxListEvent::wxListEvent, 396
wxLoadUserResource, 873
wxLocale, 397
wxLocale::~~wxLocale, 398
wxLocale::AddCatalog, 398
wxLocale::AddCatalogLookupPathPrefix, 398
wxLocale::GetLocale, 398
wxLocale::GetName, 399
wxLocale::GetString, 399
wxLocale::Init, 398
wxLocale::IsLoaded, 399
wxLocale::wxLocale, 397
wxLog::DontCreateOnDemand, 402
wxLog::Flush, 402
wxLog::GetActiveTarget, 402
wxLog::GetTimeStampFormat, 403
wxLog::GetTraceMask, 403
wxLog::GetVerbose, 403

wxLog::HasPendingMessages, 402
 wxLog::OnLog, 402
 wxLog::SetActiveTarget, 402
 wxLog::SetTimeStampFormat, 403
 wxLog::SetTraceMask, 403
 wxLog::SetVerbose, 403
 wxLogDebug, 888
 wxLogError, 887
 wxLogFatalError, 887
 wxLogMessage, 887
 wxLogStatus, 888
 wxLogSysError, 888
 wxLogTrace, 888
 wxLogVerbose, 888
 wxLogWarning, 887
 wxMakeMetafilePlaceable, 858
 wxMask, 404
 wxMask::~~wxMask, 405
 wxMask::Create, 405
 wxMask::wxMask, 404
 wxMatchWild, 851
 wxMAXIMIZE, 277, 406, 411, 447
 wxMAXIMIZE_BOX, 277, 406, 411, 447
 wxMDIChildFrame, 406, 407
 wxMDIChildFrame::~~wxMDIChildFrame, 407
 wxMDIChildFrame::Activate, 407
 wxMDIChildFrame::Create, 408
 wxMDIChildFrame::Maximize, 408
 wxMDIChildFrame::Restore, 408
 wxMDIChildFrame::wxMDIChildFrame, 406
 wxMDIClientWindow, 409
 wxMDIClientWindow::~~wxMDIClientWindow, 410
 wxMDIClientWindow::CreateClient, 410
 wxMDIClientWindow::wxMDIClientWindow, 409
 wxMDIParentFrame, 411, 412
 wxMDIParentFrame::~~wxMDIParentFrame, 412
 wxMDIParentFrame::ActivateNext, 413
 wxMDIParentFrame::ActivatePrevious, 413
 wxMDIParentFrame::ArrangeIcons, 413
 wxMDIParentFrame::Cascade, 413
 wxMDIParentFrame::Create, 413
 wxMDIParentFrame::GetActiveChild, 415
 wxMDIParentFrame::GetClientSize, 414
 wxMDIParentFrame::GetClientWindow, 415
 wxMDIParentFrame::GetToolBar, 415
 wxMDIParentFrame::OnCreateClient, 415
 wxMDIParentFrame::SetToolBar, 416
 wxMDIParentFrame::Tile, 416
 wxMDIParentFrame::wxMDIParentFrame, 411
 wxMemoryDC, 417
 wxMemoryDC::SelectObject, 417
 wxMemoryDC::wxMemoryDC, 417
 wxMemoryInputStream, 418
 wxMemoryInputStream::~~wxMemoryInputStream, 418
 wxMemoryInputStream::wxMemoryInputStream, 418
 wxMemoryOutputStream, 419
 wxMemoryOutputStream::~~wxMemoryOutputStream, 419
 wxMemoryOutputStream::wxMemoryOutputStream, 419
 wxMenu, 420
 wxMenu::~~wxMenu, 421
 wxMenu::Append, 421
 wxMenu::AppendSeparator, 422
 wxMenu::Break, 422
 wxMenu::Check, 423
 wxMenu::Enable, 423
 wxMenu::FindItem, 423
 wxMenu::FindItemForId, 424
 wxMenu::GetHelpString, 424
 wxMenu::GetLabel, 425
 wxMenu::GetTitle, 425
 wxMenu::IsChecked, 425
 wxMenu::IsEnabled, 426
 wxMenu::SetHelpString, 426
 wxMenu::SetLabel, 426
 wxMenu::SetTitle, 427
 wxMenu::UpdateUI, 427
 wxMenu::wxMenu, 420
 wxMenuBar, 428
 wxMenuBar::~~wxMenuBar, 428
 wxMenuBar::Append, 429
 wxMenuBar::Check, 429
 wxMenuBar::Enable, 429
 wxMenuBar::EnableTop, 430
 wxMenuBar::FindItemById, 431
 wxMenuBar::FindMenuItem, 430
 wxMenuBar::GetHelpString, 431
 wxMenuBar::GetLabel, 431
 wxMenuBar::GetLabelTop, 432
 wxMenuBar::GetMenu, 432
 wxMenuBar::GetMenuCount, 432
 wxMenuBar::IsChecked, 432
 wxMenuBar::IsEnabled, 433
 wxMenuBar::SetHelpString, 433
 wxMenuBar::SetLabel, 433
 wxMenuBar::SetLabelTop, 434
 wxMenuBar::wxMenuBar, 428
 wxMenuEvent, 440
 wxMenuEvent::GetMenuId, 440
 wxMenuEvent::m_menuId, 440
 wxMenuEvent::wxMenuEvent, 440
 wxMenuItem, 435
 wxMenuItem::~~wxMenuItem, 435
 wxMenuItem::Check, 435
 wxMenuItem::DeleteSubMenu, 436
 wxMenuItem::Enable, 436
 wxMenuItem::GetBackgroundColour, 436
 wxMenuItem::GetBitmap, 436
 wxMenuItem::GetFont, 436
 wxMenuItem::GetHelp, 436
 wxMenuItem::GetId, 436
 wxMenuItem::GetMarginWidth, 437
 wxMenuItem::GetName, 437
 wxMenuItem::GetSubMenu, 437
 wxMenuItem::GetTextColour, 437
 wxMenuItem::IsCheckable, 437
 wxMenuItem::IsChecked, 437
 wxMenuItem::IsEnabled, 437
 wxMenuItem::IsSeparator, 438
 wxMenuItem::SetBackgroundColour, 438
 wxMenuItem::SetBitmaps, 438

wxMenuItem::SetFont, 438
wxMenuItem::SetHelp, 438
wxMenuItem::SetMarginWidth, 438
wxMenuItem::SetName, 438
wxMenuItem::SetTextColour, 439
wxMenuItem::wxMenuItem, 435
wxMessageBox, 857
wxMessageDialog, 441
wxMessageDialog overview, 920
wxMessageDialog::~wxMessageDialog, 441
wxMessageDialog::ShowModal, 441
wxMessageDialog::wxMessageDialog, 440
wxMetafile, 442
wxMetafile::~wxMetafile, 442
wxMetafile::Ok, 442
wxMetafile::Play, 442
wxMetafile::SetClipboard, 442
wxMetafile::wxMetafile, 442
wxMetafileDC, 444
wxMetafileDC::~wxMetafileDC, 444
wxMetafileDC::Close, 444
wxMetafileDC::wxMetafileDC, 444
wxMimeTypeManager, 446
wxMimeTypeManager::~wxMimeTypeManager, 446
wxMimeTypeManager::GetFileTypeFromExtension, 446
wxMimeTypeManager::GetFileTypeFromMimeType, 446
wxMimeTypeManager::IsOfType, 446
wxMimeTypeManager::ReadMailcap, 446
wxMimeTypeManager::ReadMimeType, 447
wxMimeTypeManager::wxMimeTypeManager, 446
wxMiniFrame, 448
wxMiniFrame::~wxMiniFrame, 449
wxMiniFrame::Create, 449
wxMiniFrame::wxMiniFrame, 448
wxMINIMIZE, 276, 406, 411, 447
wxMINIMIZE_BOX, 276, 406, 411, 447
wxMkdir, 851
wxModule, 450
wxModule::~wxModule, 450
wxModule::CleanupModules, 450
wxModule::Exit, 450
wxModule::Init, 451
wxModule::InitializeModules, 451
wxModule::OnExit, 451
wxModule::OnInit, 451
wxModule::RegisterModule, 451
wxModule::RegisterModules, 451
wxModule::wxModule, 450
wxMouseEvent, 454
wxMouseEvent::AltDown, 454
wxMouseEvent::Button, 454
wxMouseEvent::ButtonDClick, 454
wxMouseEvent::ButtonDown, 455
wxMouseEvent::ButtonUp, 455
wxMouseEvent::ControlDown, 455
wxMouseEvent::Dragging, 455
wxMouseEvent::Entering, 455
wxMouseEvent::GetX, 455
wxMouseEvent::GetY, 455
wxMouseEvent::IsButton, 456
wxMouseEvent::Leaving, 456
wxMouseEvent::LeftDClick, 456
wxMouseEvent::LeftDown, 456
wxMouseEvent::LeftIsDown, 456
wxMouseEvent::LeftUp, 456
wxMouseEvent::m_altDown, 452
wxMouseEvent::m_controlDown, 452
wxMouseEvent::m_leftDown, 452, 453
wxMouseEvent::m_metaDown, 453
wxMouseEvent::m_middleDown, 453
wxMouseEvent::m_rightDown, 453
wxMouseEvent::m_shiftDown, 453
wxMouseEvent::m_x, 453
wxMouseEvent::m_y, 453
wxMouseEvent::MetaDown, 457
wxMouseEvent::MiddleDClick, 457
wxMouseEvent::MiddleDown, 457
wxMouseEvent::MiddleIsDown, 457
wxMouseEvent::MiddleUp, 457
wxMouseEvent::Moving, 457
wxMouseEvent::Position, 457
wxMouseEvent::RightDClick, 458
wxMouseEvent::RightDown, 458
wxMouseEvent::RightIsDown, 458
wxMouseEvent::RightUp, 458
wxMouseEvent::ShiftDown, 459
wxMouseEvent::wxMouseEvent, 454
wxMoveEvent, 459
wxMoveEvent::GetPosition, 459
wxMoveEvent::wxMoveEvent, 459
wxMultipleChoiceDialog overview, 921
wxMutex, 461
wxMutex::~wxMutex, 462
wxMutex::IsLocked, 462
wxMutex::Lock, 462
wxMutex::TryLock, 462
wxMutex::Unlock, 462
wxMutex::wxMutex, 461
wxMutexLocker, 463
wxMutexLocker::~wxMutexLocker, 463
wxMutexLocker::IsOk, 464
wxMutexLocker::wxMutexLocker, 463
wxNewId, 864
wxNO_3D, 179, 801
wxNodeBase::GetData, 464
wxNodeBase::GetNext, 464
wxNodeBase::IndexOf, 465
wxNodeBase::Previous, 464
wxNodeBase::SetData, 465
wxNotebook, 466
wxNotebook::~wxNotebook, 466
wxNotebook::AddPage, 467
wxNotebook::AdvanceSelection, 467
wxNotebook::Create, 467
wxNotebook::DeleteAllPages, 468
wxNotebook::DeletePage, 468
wxNotebook::GetImageList, 468
wxNotebook::GetPage, 468
wxNotebook::GetPageCount, 468
wxNotebook::GetPageImage, 468

wxNotebook::GetPageText, 468
 wxNotebook::GetRowCount, 469
 wxNotebook::GetSelection, 469
 wxNotebook::InsertPage, 469
 wxNotebook::OnSelChange, 470
 wxNotebook::RemovePage, 470
 wxNotebook::SetImageList, 470
 wxNotebook::SetPadding, 470
 wxNotebook::SetPageImage, 470
 wxNotebook::SetPageSize, 470
 wxNotebook::SetPageText, 471
 wxNotebook::SetSelection, 471
 wxNotebook::wxNotebook, 466
 wxNotebookEvent, 472
 wxNotebookEvent::GetOldSelection, 472
 wxNotebookEvent::GetSelection, 472
 wxNotebookEvent::SetOldSelection, 472
 wxNotebookEvent::SetSelection, 472
 wxNotebookEvent::wxNotebookEvent, 472
 wxNow, 874
 wxObjArray, 21, 22
 wxObjArray::Detach, 23
 wxObject, 473
 wxObject::~wxObject, 473
 wxObject::Dump, 473
 wxObject::GetClassInfo, 474
 wxObject::GetRefData, 474
 wxObject::IsKindOf, 474
 wxObject::m_refData, 473
 wxObject::operator delete, 476
 wxObject::operator new, 476
 wxObject::Ref, 475
 wxObject::SetRefData, 475
 wxObject::UnRef, 475
 wxObject::wxObject, 473
 wxObjectRefData, 477
 wxObjectRefData::~wxObjectRefData, 477
 wxObjectRefData::m_count, 476
 wxObjectRefData::wxObjectRefData, 477
 wxOnAssert, 889
 wxOpenClipboard, 863
 wxOutputStream, 477
 wxOutputStream::~wxOutputStream, 477
 wxOutputStream::LastWrite, 478
 wxOutputStream::OutputStreamBuffer, 477
 wxOutputStream::PutC, 478
 wxOutputStream::SeekO, 478
 wxOutputStream::TellO, 478
 wxOutputStream::Write, 478
 wxOutputStream::wxOutputStream, 477
 wxPageSetupData, 479
 wxPageSetupData::~wxPageSetupData, 479
 wxPageSetupData::EnableHelp, 479
 wxPageSetupData::EnableMargins, 479
 wxPageSetupData::EnableOrientation, 479
 wxPageSetupData::EnablePaper, 480
 wxPageSetupData::EnablePrinter, 480
 wxPageSetupData::GetDefaultInfo, 482
 wxPageSetupData::GetDefaultMinMargins, 481
 wxPageSetupData::GetEnableHelp, 481
 wxPageSetupData::GetEnableMargins, 481
 wxPageSetupData::GetEnableOrientation, 481
 wxPageSetupData::GetEnablePaper, 481
 wxPageSetupData::GetEnablePrinter, 481
 wxPageSetupData::GetMarginBottomRight, 480
 wxPageSetupData::GetMarginTopLeft, 480
 wxPageSetupData::GetMinMarginBottomRight, 480
 wxPageSetupData::GetMinMarginTopLeft, 480
 wxPageSetupData::GetOrientation, 481
 wxPageSetupData::GetPaperSize, 480
 wxPageSetupData::SetDefaultInfo, 483
 wxPageSetupData::SetDefaultMinMargins, 483
 wxPageSetupData::SetMarginBottomRight, 482
 wxPageSetupData::SetMarginTopLeft, 482
 wxPageSetupData::SetMinMarginBottomRight, 482
 wxPageSetupData::SetMinMarginTopLeft, 482
 wxPageSetupData::SetOrientation, 482
 wxPageSetupData::SetPaperSize, 482
 wxPageSetupData::wxPageSetupData, 479
 wxPageSetupDialog, 484
 wxPageSetupDialog::~wxPageSetupDialog, 484
 wxPageSetupDialog::GetPageSetupData, 484
 wxPageSetupDialog::ShowModal, 484
 wxPageSetupDialog::wxPageSetupDialog, 483
 wxPaintDC, 485
 wxPaintDC::wxPaintDC, 485
 wxPaintEvent, 485
 wxPaintEvent::wxPaintEvent, 485
 wxPalette, 486
 wxPalette::~wxPalette, 487
 wxPalette::Create, 487
 wxPalette::GetPixel, 488
 wxPalette::GetRGB, 488
 wxPalette::Ok, 489
 wxPalette::operator !=, 489
 wxPalette::operator =, 489
 wxPalette::operator ==, 489
 wxPalette::wxPalette, 486
 wxPanel, 490
 wxPanel::~wxPanel, 491
 wxPanel::Create, 491
 wxPanel::InitDialog, 491
 wxPanel::OnSysColourChanged, 491
 wxPanel::wxPanel, 490
 wxPanelTabView, 492
 wxPanelTabView::~wxPanelTabView, 493
 wxPanelTabView::AddTabWindow, 493
 wxPanelTabView::ClearWindows, 493
 wxPanelTabView::GetCurrentWindow, 493
 wxPanelTabView::GetTabWindow, 493
 wxPanelTabView::ShowWindowForTab, 493
 wxPanelTabView::wxPanelTabView, 492
 wxPathList, 494
 wxPathList::Add, 494
 wxPathList::AddEnvList, 494
 wxPathList::EnsureFileAccessible, 494
 wxPathList::FindAbsoluteValidPath, 495
 wxPathList::FindValidPath, 495
 wxPathList::Member, 495
 wxPathList::wxPathList, 494
 wxPathOnly, 848
 wxPen, 496, 497

wxPen::~wxPen, 498
wxPen::GetCap, 498
wxPen::GetColour, 498
wxPen::GetDashes, 499
wxPen::GetJoin, 499
wxPen::GetStipple, 499
wxPen::GetStyle, 499
wxPen::GetWidth, 500
wxPen::Ok, 500
wxPen::operator !=, 502
wxPen::operator =, 502
wxPen::operator ==, 502
wxPen::SetCap, 500
wxPen::SetColour, 500
wxPen::SetDashes, 500
wxPen::SetJoin, 501
wxPen::SetStipple, 501
wxPen::SetStyle, 501
wxPen::SetWidth, 501
wxPen::wxPen, 496
wxPenList, 503
wxPenList::AddPen, 503
wxPenList::FindOrCreatePen, 503
wxPenList::RemovePen, 504
wxPenList::wxPenList, 503
wxPoint, 504
wxPoint::wxPoint, 504
wxPoint::x, 504
wxPoint::y, 504
wxPostDelete, 874
wxPostScriptDC, 505
wxPostScriptDC::GetStream, 505
wxPostScriptDC::wxPostScriptDC, 505
wxPreviewCanvas, 506
wxPreviewCanvas::~wxPreviewCanvas, 506
wxPreviewCanvas::OnPaint, 506
wxPreviewCanvas::wxPreviewCanvas, 506
wxPreviewControlBar, 507
wxPreviewControlBar::~wxPreviewControlBar, 507
wxPreviewControlBar::CreateButtons, 507
wxPreviewControlBar::GetPrintPreview, 507
wxPreviewControlBar::GetZoomControl, 508
wxPreviewControlBar::SetZoomControl, 508
wxPreviewControlBar::wxPreviewControlBar, 507
wxPreviewFrame, 508
wxPreviewFrame::~wxPreviewFrame, 508
wxPreviewFrame::CreateCanvas, 509
wxPreviewFrame::CreateControlBar, 509
wxPreviewFrame::Initialize, 509
wxPreviewFrame::OnCloseWindow, 509
wxPreviewFrame::wxPreviewFrame, 508
wxPrintData, 510
wxPrintData::~wxPrintData, 510
wxPrintData::EnableHelp, 510
wxPrintData::EnablePageNumbers, 510
wxPrintData::EnablePrintToFile, 510
wxPrintData::EnableSelection, 510
wxPrintData::GetAllPages, 511
wxPrintData::GetCollate, 511
wxPrintData::GetFromPage, 511
wxPrintData::GetMaxPage, 511
wxPrintData::GetMinPage, 511
wxPrintData::GetNoCopies, 511
wxPrintData::GetOrientation, 511
wxPrintData::GetToPage, 511
wxPrintData::SetCollate, 512
wxPrintData::SetFromPage, 512
wxPrintData::SetMaxPage, 512
wxPrintData::SetMinPage, 512
wxPrintData::SetNoCopies, 512
wxPrintData::SetOrientation, 512
wxPrintData::SetPrintToFile, 512
wxPrintData::SetSetupDialog, 513
wxPrintData::SetToPage, 513
wxPrintData::wxPrintData, 510
wxPrintDialog, 513
wxPrintDialog overview, 919
wxPrintDialog::~wxPrintDialog, 514
wxPrintDialog::GetPrintData, 514
wxPrintDialog::GetPrintDC, 514
wxPrintDialog::ShowModal, 514
wxPrintDialog::wxPrintDialog, 513
wxPrinter, 515
wxPrinter::~wxPrinter, 515
wxPrinter::Abort, 515
wxPrinter::CreateAbortWindow, 515
wxPrinter::GetPrintData, 515
wxPrinter::Print, 516
wxPrinter::PrintDialog, 516
wxPrinter::ReportError, 516
wxPrinter::Setup, 516
wxPrinter::wxPrinter, 515
wxPrinterDC, 517
wxPrinterDC::wxPrinterDC, 517
wxPrintout, 517
wxPrintout::~wxPrintout, 517
wxPrintout::GetDC, 518
wxPrintout::GetPageInfo, 518
wxPrintout::GetPageSizeMM, 518
wxPrintout::GetPageSizePixels, 518
wxPrintout::GetPPIPrinter, 518
wxPrintout::GetPPIScreen, 519
wxPrintout::HasPage, 519
wxPrintout::IsPreview, 519
wxPrintout::OnBeginDocument, 519
wxPrintout::OnBeginPrinting, 520
wxPrintout::OnEndDocument, 519
wxPrintout::OnEndPrinting, 520
wxPrintout::OnPreparePrinting, 520
wxPrintout::OnPrintPage, 520
wxPrintout::wxPrintout, 517
wxPrintPreview, 521
wxPrintPreview::~wxPrintPreview, 521
wxPrintPreview::DrawBlankPage, 521
wxPrintPreview::GetCanvas, 521
wxPrintPreview::GetCurrentPage, 522
wxPrintPreview::GetFrame, 522
wxPrintPreview::GetMaxPage, 522
wxPrintPreview::GetMinPage, 522
wxPrintPreview::GetPrintData, 522
wxPrintPreview::GetPrintout, 522
wxPrintPreview::GetPrintoutForPrinting, 522
wxPrintPreview::Ok, 523

-
- wxPrintPreview::PaintPage, 523
 - wxPrintPreview::Print, 523
 - wxPrintPreview::RenderPage, 523
 - wxPrintPreview::SetCanvas, 523
 - wxPrintPreview::SetCurrentPage, 523
 - wxPrintPreview::SetFrame, 524
 - wxPrintPreview::SetPrintout, 524
 - wxPrintPreview::SetZoom, 524
 - wxPrintPreview::wxPrintPreview, 521
 - wxPrivateDataObject, 524
 - wxPrivateDataObject::~~wxPrivateDataObject, 525
 - wxPrivateDataObject::GetData, 525
 - wxPrivateDataObject::GetId, 525
 - wxPrivateDataObject::GetSize, 525
 - wxPrivateDataObject::SetData, 525
 - wxPrivateDataObject::SetId, 525
 - wxPrivateDataObject::WriteData, 525, 526
 - wxPrivateDataObject::wxPrivateDataObject, 524
 - wxPrivateDropTarget, 526
 - wxPrivateDropTarget::GetId, 526
 - wxPrivateDropTarget::SetId, 526
 - wxPrivateDropTarget::wxPrivateDropTarget, 526
 - wxProcess, 527
 - wxProcess::~~wxProcess, 527
 - wxProcess::Detach, 527
 - wxProcess::OnTerminate, 528
 - wxProcess::wxProcess, 527
 - wxProcessEvent, 529
 - wxProcessEvent::GetPid, 529
 - wxProcessEvent::m_pid, 529
 - wxProcessEvent::SetPid, 529
 - wxProcessEvent::wxProcessEvent, 529
 - wxProtocol::Abort, 530
 - wxProtocol::GetContentType, 531
 - wxProtocol::GetError, 530
 - wxProtocol::GetInputStream, 530
 - wxProtocol::Reconnect, 529
 - wxProtocol::SetPassword, 531
 - wxProtocol::SetUser, 531
 - wxQueryCol, 532
 - wxQueryCol overview, 925
 - wxQueryCol::~~wxQueryCol, 532
 - wxQueryCol::AppendField, 533
 - wxQueryCol::BindVar, 532
 - wxQueryCol::FillVar, 532
 - wxQueryCol::GetData, 532
 - wxQueryCol::GetName, 532
 - wxQueryCol::GetSize, 533
 - wxQueryCol::GetType, 532
 - wxQueryCol::IsNullable, 533
 - wxQueryCol::IsRowDirty, 533
 - wxQueryCol::SetData, 533
 - wxQueryCol::SetFieldDirty, 534
 - wxQueryCol::SetName, 533
 - wxQueryCol::SetNullable, 533
 - wxQueryCol::SetType, 534
 - wxQueryCol::wxQueryCol, 532
 - wxQueryField, 534
 - wxQueryField overview, 926
 - wxQueryField::~~wxQueryField, 534
 - wxQueryField::AllocData, 534
 - wxQueryField::ClearData, 535
 - wxQueryField::GetData, 535
 - wxQueryField::GetSize, 535
 - wxQueryField::GetType, 535
 - wxQueryField::IsDirty, 535
 - wxQueryField::SetData, 535
 - wxQueryField::SetDirty, 535
 - wxQueryField::SetSize, 535
 - wxQueryField::SetType, 536
 - wxQueryField::wxQueryField, 534
 - wxQueryLayoutInfoEvent, 537
 - wxQueryLayoutInfoEvent::GetAlignment, 537
 - wxQueryLayoutInfoEvent::GetFlags, 537
 - wxQueryLayoutInfoEvent::GetOrientation, 537
 - wxQueryLayoutInfoEvent::GetRequestedLength, 537
 - wxQueryLayoutInfoEvent::GetSize, 537
 - wxQueryLayoutInfoEvent::SetAlignment, 538
 - wxQueryLayoutInfoEvent::SetFlags, 538
 - wxQueryLayoutInfoEvent::SetOrientation, 538
 - wxQueryLayoutInfoEvent::SetRequestedLength, 538
 - wxQueryLayoutInfoEvent::SetSize, 538
 - wxQueryLayoutInfoEvent::wxQueryLayoutInfoEvent, 537
 - wxRA_SPECIFY_COLS, 539
 - wxRA_SPECIFY_ROWS, 539
 - wxRadioBox, 539
 - wxRadioBox::~~wxRadioBox, 540
 - wxRadioBox::Create, 540
 - wxRadioBox::Enable, 541
 - wxRadioBox::FindString, 541
 - wxRadioBox::GetLabel, 542
 - wxRadioBox::GetSelection, 542
 - wxRadioBox::GetString, 544
 - wxRadioBox::GetStringSelection, 542
 - wxRadioBox::Number, 542
 - wxRadioBox::SetLabel, 543
 - wxRadioBox::SetSelection, 543
 - wxRadioBox::SetStringSelection, 543
 - wxRadioBox::Show, 544
 - wxRadioBox::wxRadioBox, 539
 - wxRadioButton, 545
 - wxRadioButton::~~wxRadioButton, 546
 - wxRadioButton::Create, 546
 - wxRadioButton::GetValue, 546
 - wxRadioButton::SetValue, 546
 - wxRadioButton::wxRadioButton, 545
 - wxRAISED_BORDER, 801
 - wxRealPoint, 547
 - wxRealPoint::wxRealPoint, 547
 - wxRecordSet, 551
 - wxRecordSet overview, 926
 - wxRecordSet::~~wxRecordSet, 552
 - wxRecordSet::AddNew, 552
 - wxRecordSet::BeginQuery, 552
 - wxRecordSet::BindVar, 552
 - wxRecordSet::CanAppend, 552
 - wxRecordSet::Cancel, 553
 - wxRecordSet::CanRestart, 553
 - wxRecordSet::CanScroll, 553
 - wxRecordSet::CanTransact, 553
-

wxRecordSet::CanUpdate, 553
wxRecordSet::ConstructDefaultSQL, 553
wxRecordSet::Delete, 553
wxRecordSet::Edit, 554
wxRecordSet::EndQuery, 554
wxRecordSet::ExecuteSQL, 554
wxRecordSet::FillVars, 554
wxRecordSet::GetColName, 554
wxRecordSet::GetColType, 554
wxRecordSet::GetColumns, 555
wxRecordSet::GetCurrentRecord, 555
wxRecordSet::GetDatabase, 555
wxRecordSet::GetDataSources, 555
wxRecordSet::GetDefaultConnect, 556
wxRecordSet::GetDefaultSQL, 556
wxRecordSet::GetErrorCode, 556
wxRecordSet::GetFieldData, 557
wxRecordSet::GetFieldDataPtr, 557
wxRecordSet::GetFilter, 557
wxRecordSet::GetForeignKeys, 557
wxRecordSet::GetNumberCols, 558
wxRecordSet::GetNumberFields, 558
wxRecordSet::GetNumberParams, 558
wxRecordSet::GetNumberRecords, 559
wxRecordSet::GetOptions, 559
wxRecordSet::GetPrimaryKeys, 559
wxRecordSet::GetResultSet, 559
wxRecordSet::GetSortString, 559
wxRecordSet::GetSQL, 559
wxRecordSet::GetTableName, 560
wxRecordSet::GetTables, 560
wxRecordSet::GetType, 560
wxRecordSet::GoTo, 560
wxRecordSet::IsBOF, 560
wxRecordSet::IsColNullable, 561
wxRecordSet::IsDeleted, 561
wxRecordSet::IsEOF, 561
wxRecordSet::IsFieldDirty, 560
wxRecordSet::IsFieldNull, 561
wxRecordSet::IsOpen, 561
wxRecordSet::Move, 561
wxRecordSet::MoveFirst, 562
wxRecordSet::MoveLast, 562
wxRecordSet::MoveNext, 562
wxRecordSet::MovePrev, 562
wxRecordSet::Query, 562
wxRecordSet::RecordCountFinal, 562
wxRecordSet::Requery, 562
wxRecordSet::SetDefaultSQL, 563
wxRecordSet::SetFieldDirty, 562
wxRecordSet::SetFieldNull, 563
wxRecordSet::SetOptions, 563
wxRecordSet::SetTableName, 563
wxRecordSet::SetType, 563
wxRecordSet::Update, 563
wxRecordSet::wxRecordSet, 551
wxRect, 548
wxRect::GetBottom, 549
wxRect::GetHeight, 549
wxRect::GetLeft, 549
wxRect::GetPosition, 549
wxRect::GetRight, 549
wxRect::GetSize, 549
wxRect::GetTop, 549
wxRect::GetWidth, 550
wxRect::GetX, 550
wxRect::GetY, 550
wxRect::height, 548
wxRect::operator !=, 551
wxRect::operator =, 551
wxRect::operator ==, 551
wxRect::SetHeight, 550
wxRect::SetWidth, 550
wxRect::SetX, 550
wxRect::SetY, 550
wxRect::width, 548
wxRect::wxRect, 548
wxRect::x, 548
wxRect::y, 548
wxRegion, 564
wxRegion::~~wxRegion, 564
wxRegion::Clear, 564
wxRegion::Contains, 564
wxRegion::GetBox, 565
wxRegion::Intersect, 565
wxRegion::IsEmpty, 566
wxRegion::operator =, 567
wxRegion::Union, 566
wxRegion::wxRegion, 564
wxRegion::Xor, 567
wxRegionIterator, 568
wxRegionIterator::GetH, 569
wxRegionIterator::GetHeight, 569
wxRegionIterator::GetRect, 569
wxRegionIterator::GetW, 568
wxRegionIterator::GetWidth, 568
wxRegionIterator::GetX, 568
wxRegionIterator::GetY, 568
wxRegionIterator::HaveRects, 569
wxRegionIterator::operator ++, 569
wxRegionIterator::operator bool, 570
wxRegionIterator::Reset, 569
wxRegionIterator::wxRegionIterator, 568
wxRegisterClipboardFormat, 863
wxRegisterId, 864
wxRelationship, 344
wxRemoveFile, 851
wxRenameFile, 851
wxRESIZE_BORDER, 179, 277, 406, 411, 447
wxResourceAddIdentifier, 883
wxResourceClear, 883
wxResourceCreateBitmap, 884
wxResourceCreateIcon, 884
wxResourceCreateMenuBar, 884
wxResourceGetIdentifier, 885
wxResourceParseData, 885
wxResourceParseFile, 886
wxResourceParseString, 886
wxResourceRegisterBitmapData, 886
wxRETAINED, 588
wxRmdir, 851
wxSashEvent, 571
wxSashEvent::GetDragRect, 571
wxSashEvent::GetDragStatus, 571

-
- wxSashEvent::GetEdge, 571
 - wxSashEvent::wxSashEvent, 571
 - wxSashLayoutWindow, 572, 573
 - wxSashLayoutWindow::~~wxSashLayoutWindow, 573
 - wxSashLayoutWindow::GetAlignment, 573
 - wxSashLayoutWindow::GetOrientation, 573
 - wxSashLayoutWindow::OnCalculateLayout, 574
 - wxSashLayoutWindow::OnQueryLayoutInfo, 574
 - wxSashLayoutWindow::SetAlignment, 574
 - wxSashLayoutWindow::SetDefaultSize, 574
 - wxSashLayoutWindow::SetOrientation, 574
 - wxSashLayoutWindow::wxSashLayoutWindow, 572
 - wxSashWindow, 576
 - wxSashWindow::~~wxSashWindow, 576
 - wxSashWindow::GetMaximumSizeX, 577
 - wxSashWindow::GetMaximumSizeY, 577
 - wxSashWindow::GetMinimumSizeX, 577
 - wxSashWindow::GetMinimumSizeY, 577
 - wxSashWindow::GetSashVisible, 577
 - wxSashWindow::HasBorder, 577
 - wxSashWindow::SetMaximumSizeX, 578
 - wxSashWindow::SetMaximumSizeY, 578
 - wxSashWindow::SetMinimumSizeX, 578
 - wxSashWindow::SetMinimumSizeY, 578
 - wxSashWindow::SetSashBorder, 579
 - wxSashWindow::SetSashVisible, 578
 - wxSashWindow::wxSashWindow, 576
 - wxSB_HORIZONTAL, 581
 - wxSB_SIZEGRIP, 642
 - wxSB_VERTICAL, 581
 - wxScreenDC, 580
 - wxScreenDC::EndDrawingOnTop, 580
 - wxScreenDC::StartDrawingOnTop, 580
 - wxScreenDC::wxScreenDC, 580
 - wxScrollBar, 582
 - wxScrollBar::~~wxScrollBar, 583
 - wxScrollBar::Create, 583
 - wxScrollBar::GetPageSize, 583
 - wxScrollBar::GetRange, 583
 - wxScrollBar::GetThumbLength, 584
 - wxScrollBar::GetThumbPosition, 583
 - wxScrollBar::SetScrollbar, 584
 - wxScrollBar::SetThumbPosition, 584
 - wxScrollBar::wxScrollBar, 582
 - wxScrolledWindow, 588
 - wxScrolledWindow::~~wxScrolledWindow, 589
 - wxScrolledWindow::Create, 589
 - wxScrolledWindow::EnableScrolling, 589
 - wxScrolledWindow::GetScrollPixelsPerUnit, 590
 - wxScrolledWindow::GetVirtualSize, 590
 - wxScrolledWindow::IsRetained, 591
 - wxScrolledWindow::OnDraw, 591
 - wxScrolledWindow::OnPaint, 591
 - wxScrolledWindow::OnScroll, 592
 - wxScrolledWindow::PrepareDC, 591
 - wxScrolledWindow::Scroll, 592
 - wxScrolledWindow::SetScrollbars, 593
 - wxScrolledWindow::ViewStart, 594
 - wxScrolledWindow::wxScrolledWindow, 588
 - wxScrollEvent, 587
 - wxScrollEvent::GetOrientation, 587
 - wxScrollEvent::GetPosition, 587
 - wxScrollEvent::wxScrollEvent, 587
 - wxSetClipboardData, 863
 - wxSetCursor, 859
 - wxSetDisplayName, 875
 - wxSetPrinterCommand, 860
 - wxSetPrinterFile, 860
 - wxSetPrinterMode, 860
 - wxSetPrinterOptions, 861
 - wxSetPrinterOrientation, 861
 - wxSetPrinterPreviewCommand, 861
 - wxSetPrinterScaling, 861
 - wxSetPrinterTranslation, 861
 - wxSetWorkingDirectory, 851
 - wxShell, 875
 - wxSIMPLE_BORDER, 277, 801
 - wxSingleChoiceDialog, 595
 - wxSingleChoiceDialog overview, 920
 - wxSingleChoiceDialog::~~wxSingleChoiceDialog, 595
 - wxSingleChoiceDialog::GetSelection, 596
 - wxSingleChoiceDialog::GetSelectionClientData, 596
 - wxSingleChoiceDialog::GetStringSelection, 596
 - wxSingleChoiceDialog::ShowModal, 596
 - wxSingleChoiceDialog::wxSingleChoiceDialog, 595
 - wxSize, 597
 - wxSize::GetX, 597
 - wxSize::GetY, 597
 - wxSize::operator =, 598
 - wxSize::Set, 597
 - wxSize::wxSize, 597
 - wxSize::x, 597
 - wxSize::y, 597
 - wxSizeEvent, 598
 - wxSizeEvent::GetSize, 598
 - wxSizeEvent::wxSizeEvent, 598
 - wxSL_AUTOTICKS, 599
 - wxSL_HORIZONTAL, 599
 - wxSL_LABELS, 599
 - wxSL_LEFT, 599
 - wxSL_RIGHT, 599
 - wxSL_SELRANGE, 599
 - wxSL_TOP, 599
 - wxSL_VERTICAL, 599
 - wxSleep, 875
 - wxSlider, 600
 - wxSlider::~~wxSlider, 601
 - wxSlider::ClearSel, 601
 - wxSlider::ClearTicks, 601
 - wxSlider::Create, 601
 - wxSlider::GetLineSize, 601
 - wxSlider::GetMax, 602
 - wxSlider::GetMin, 602
 - wxSlider::GetPageSize, 602
 - wxSlider::GetSelEnd, 602
 - wxSlider::GetSelStart, 603
 - wxSlider::GetThumbLength, 603
 - wxSlider::GetTickFreq, 603
 - wxSlider::GetValue, 604
-

wxSlider::SetLineSize, 604
 wxSlider::SetPageSize, 605
 wxSlider::SetRange, 604
 wxSlider::SetSelection, 605
 wxSlider::SetThumbLength, 606
 wxSlider::SetTick, 606
 wxSlider::SetTickFreq, 604
 wxSlider::SetValue, 606
 wxSlider::wxSlider, 600
 wxSocketAddress, 607
 wxSocketAddress::~~wxSocketAddress, 607
 wxSocketAddress::Build, 607
 wxSocketAddress::Clear, 607
 wxSocketAddress::Disassemble, 608
 wxSocketAddress::SockAddrLen, 608
 wxSocketAddress::wxSocketAddress, 607
 wxSocketBase, 608
 wxSocketBase::~~wxSocketBase, 609
 wxSocketBase::Discard, 614
 wxSocketBase::Error, 609
 wxSocketBase::IsConnected, 609
 wxSocketBase::IsData, 609
 wxSocketBase::IsDisconnected, 609
 wxSocketBase::IsNoWait, 609
 wxSocketBase::LastCount, 610
 wxSocketBase::LastError, 610
 wxSocketBase::Ok, 609
 wxSocketBase::Peek, 610
 wxSocketBase::Read, 610
 wxSocketBase::ReadMsg, 613
 wxSocketBase::RestoreState, 616
 wxSocketBase::SaveState, 616
 wxSocketBase::SetEventHandler, 616
 wxSocketBase::SetFlags, 611
 wxSocketBase::UnRead, 613
 wxSocketBase::Wait, 614
 wxSocketBase::WaitForLost, 615
 wxSocketBase::WaitForRead, 614
 wxSocketBase::WaitForWrite, 615
 wxSocketBase::Write, 611
 wxSocketBase::WriteMsg, 612
 wxSocketBase::wxSocketBase, 608
 wxSocketClient, 617
 wxSocketClient::~~wxSocketClient, 617
 wxSocketClient::Connect, 617
 wxSocketClient::WaitOnConnect, 618
 wxSocketClient::wxSocketClient, 617
 wxSocketEvent, 619
 wxSocketEvent::SocketEvent, 619
 wxSocketEvent::wxSocketEvent, 619
 wxSocketHandler, 619
 wxSocketHandler::~~wxSocketHandler, 619
 wxSocketHandler::Count, 620
 wxSocketHandler::CreateClient, 620
 wxSocketHandler::CreateServer, 620
 wxSocketHandler::Master, 621
 wxSocketHandler::Register, 619
 wxSocketHandler::UnRegister, 620
 wxSocketHandler::Wait, 621
 wxSocketHandler::wxSocketHandler, 619
 wxSocketHandler::YieldSock, 621
 wxSocketInputStream, 623, 624
 wxSocketInputStream::wxSocketInputStream, 623
 wxSocketOutputStream::wxSocketOutputStream, 624
 wxSocketServer, 622
 wxSocketServer::~~wxSocketServer, 622
 wxSocketServer::Accept, 622
 wxSocketServer::AcceptWith, 622
 wxSocketServer::wxSocketServer, 622
 wxSortedArray, 21, 22
 wxSP_3D, 628
 wxSP_ARROW_KEYS, 624
 wxSP_BORDER, 628
 wxSP_HORIZONTAL, 624
 wxSP_NOBORDER, 628
 wxSP_VERTICAL, 624
 wxSP_WRAP, 624
 wxSpinButton, 625
 wxSpinButton::~~wxSpinButton, 626
 wxSpinButton::Create, 626
 wxSpinButton::GetMax, 626
 wxSpinButton::GetMin, 626
 wxSpinButton::GetValue, 626
 wxSpinButton::SetRange, 627
 wxSpinButton::SetValue, 627
 wxSpinButton::wxSpinButton, 625
 wxSplitPath, 852
 wxSplitterWindow, 628
 wxSplitterWindow::~~wxSplitterWindow, 629
 wxSplitterWindow::Create, 629
 wxSplitterWindow::GetMinimumPaneSize, 629
 wxSplitterWindow::GetSashPosition, 630
 wxSplitterWindow::GetSplitMode, 630
 wxSplitterWindow::GetWindow1, 630
 wxSplitterWindow::GetWindow2, 630
 wxSplitterWindow::Initialize, 630
 wxSplitterWindow::IsSplit, 631
 wxSplitterWindow::OnDoubleClickSash, 631
 wxSplitterWindow::OnSashPositionChange, 632
 wxSplitterWindow::OnUnsplit, 631
 wxSplitterWindow::ReplaceWindow, 632
 wxSplitterWindow::SetMinimumPaneSize, 633
 wxSplitterWindow::SetSashPosition, 633
 wxSplitterWindow::SetSplitMode, 633
 wxSplitterWindow::SplitHorizontally, 634
 wxSplitterWindow::SplitVertically, 635
 wxSplitterWindow::Unsplit, 635
 wxSplitterWindow::wxSplitterWindow, 628
 wxStartTimer, 876
 wxSTATIC_BORDER, 801
 wxStaticBitmap, 636, 637
 wxStaticBitmap::Create, 637
 wxStaticBitmap::GetBitmap, 637
 wxStaticBitmap::SetBitmap, 638
 wxStaticBitmap::wxStaticBitmap, 636
 wxStaticBox, 639
 wxStaticBox::~~wxStaticBox, 639
 wxStaticBox::Create, 639
 wxStaticBox::wxStaticBox, 638
 wxStaticText, 640
 wxStaticText::Create, 641
 wxStaticText::GetLabel, 641

wxStaticText::SetLabel, 641
wxStaticText::wxStaticText, 640
wxStatusBar, 642
wxStatusBar::~~wxStatusBar, 643
wxStatusBar::Create, 643
wxStatusBar::DrawField, 644
wxStatusBar::DrawFieldText, 645
wxStatusBar::GetFieldRect, 643
wxStatusBar::GetFieldsCount, 644
wxStatusBar::GetStatusText, 644
wxStatusBar::InitColours, 645
wxStatusBar::OnSysColourChanged, 646
wxStatusBar::SetFieldsCount, 646
wxStatusBar::SetStatusText, 646
wxStatusBar::SetStatusWidths, 647
wxStatusBar::wxStatusBar, 642
wxSTAY_ON_TOP, 179, 277, 406, 411, 447
wxStreamBase, 647
wxStreamBase::~~wxStreamBase, 648
wxStreamBase::LastError, 648
wxStreamBase::OnSysRead, 648
wxStreamBase::OnSysSeek, 648
wxStreamBase::OnSysTell, 648
wxStreamBase::OnSysWrite, 648
wxStreamBase::StreamSize, 649
wxStreamBase::wxStreamBase, 647
wxStreamBuffer, 649, 650
wxStreamBuffer::~~wxStreamBuffer, 650
wxStreamBuffer::FillBuffer, 655
wxStreamBuffer::Fixed, 655
wxStreamBuffer::Flushable, 655
wxStreamBuffer::FlushBuffer, 655
wxStreamBuffer::GetBufferEnd, 654
wxStreamBuffer::GetBufferPos, 654
wxStreamBuffer::GetBufferStart, 654
wxStreamBuffer::GetChar, 652
wxStreamBuffer::GetDataLeft, 655
wxStreamBuffer::GetIntPosition, 654
wxStreamBuffer::GetLastAccess, 655
wxStreamBuffer::PutChar, 652
wxStreamBuffer::Read, 650
wxStreamBuffer::ResetBuffer, 653
wxStreamBuffer::Seek, 652
wxStreamBuffer::SetBufferIO, 653
wxStreamBuffer::SetIntPosition, 654
wxStreamBuffer::Stream, 656
wxStreamBuffer::Tell, 652
wxStreamBuffer::Write, 651
wxStreamBuffer::WriteBack, 651
wxStreamBuffer::wxStreamBuffer, 649
wxString, 663
wxString::~~wxString, 663
wxString::AfterFirst, 664
wxString::AfterLast, 664
wxString::Alloc, 663
wxString::Append, 664
wxString::BeforeFirst, 664
wxString::BeforeLast, 664
wxString::c_str, 665
wxString::Clear, 665
wxString::Cmp, 665
wxString::CmpNoCase, 665
wxString::CompareTo, 665
wxString::Contains, 666
wxString::Empty, 666
wxString::Find, 666
wxString::First, 666
wxString::Freq, 666
wxString::GetChar, 667
wxString::GetData, 667
wxString::GetWritableChar, 667
wxString::GetWriteBuf, 667
wxString::Index, 667
wxString::IsAscii, 668
wxString::IsEmpty, 668
wxString::IsNull, 668
wxString::IsNumber, 668
wxString::IsSameAs, 668
wxString::IsWord, 669
wxString::Last, 669
wxString::Left, 669
wxString::Len, 669
wxString::Length, 669
wxString::Lower, 669
wxString::LowerCase, 670
wxString::MakeLower, 670
wxString::MakeUpper, 670
wxString::Matches, 670
wxString::Mid, 670
wxString::operator (), 675
wxString::operator [], 674
wxString::operator +=, 674
wxString::operator <<, 675
wxString::operator =, 674
wxString::operator >>, 675
wxString::operator const char*, 675
wxString::operator!, 673
wxString::Pad, 670
wxString::Prepend, 670
wxString::Printf, 671
wxString::PrintfV, 671
wxString::Remove, 671
wxString::RemoveLast, 671
wxString::Replace, 671
wxString::Right, 672
wxString::SetChar, 672
wxString::Shrink, 672
wxString::sprintf, 672
wxString::Strip, 672
wxString::SubString, 672
wxString::Trim, 673
wxString::Truncate, 673
wxString::UngetWriteBuf, 673
wxString::Upper, 673
wxString::UpperCase, 673
wxString::wxString, 662
wxStringEq, 853
wxStringList, 676, 677
wxStringList::~~wxStringList, 677
wxStringList::Add, 677
wxStringList::Clear, 677
wxStringList::Delete, 677
wxStringList::ListToArray, 677
wxStringList::Member, 677

wxStringList::Sort, 677
wxStringList::wxStringList, 676
wxStringMatch, 853
wxStringTokenizer, 678
wxStringTokenizer::~~wxStringTokenizer, 678
wxStringTokenizer::CountTokens, 678
wxStringTokenizer::GetNextToken, 679
wxStringTokenizer::GetString, 679
wxStringTokenizer::HasMoreTokens, 678
wxStringTokenizer::SetString, 679
wxStringTokenizer::wxStringTokenizer, 678
wxStripMenuCodes, 875
wxSUNKEN_BORDER, 801
wxSW_3D, 575
wxSysColourChanged, 680
wxSysColourChangedEvent::wxSysColourChanged, 680
wxSYSTEM_MENU, 179, 277, 406, 411, 447
wxSystemSettings, 680
wxSystemSettings::GetSystemColour, 681
wxSystemSettings::GetSystemFont, 681
wxSystemSettings::GetSystemMetric, 682
wxSystemSettings::wxSystemSettings, 680
wxTAB_TRAVERSAL, 801
wxTabbedDialog, 684
wxTabbedDialog::~~wxTabbedDialog, 684
wxTabbedDialog::GetTabView, 684
wxTabbedDialog::SetTabView, 684
wxTabbedDialog::wxTabbedDialog, 684
wxTabbedPanel, 685
wxTabbedPanel::GetTabView, 685
wxTabbedPanel::SetTabView, 685
wxTabbedPanel::wxTabbedPanel, 685
wxTabControl, 685
wxTabControl::GetColPosition, 686
wxTabControl::GetFont, 686
wxTabControl::GetHeight, 686
wxTabControl::GetId, 686
wxTabControl::GetLabel, 686
wxTabControl::GetRowPosition, 686
wxTabControl::GetSelected, 686
wxTabControl::GetWidth, 687
wxTabControl::GetX, 687
wxTabControl::GetY, 687
wxTabControl::HitTest, 687
wxTabControl::OnDraw, 687
wxTabControl::SetColPosition, 687
wxTabControl::SetFont, 687
wxTabControl::SetId, 688
wxTabControl::SetLabel, 688
wxTabControl::SetPosition, 688
wxTabControl::SetRowPosition, 688
wxTabControl::SetSelected, 688
wxTabControl::SetSize, 688
wxTabControl::wxTabControl, 685
wxTabCtrl, 697
wxTabCtrl::~~wxTabCtrl, 698
wxTabCtrl::Create, 698
wxTabCtrl::DeleteAllItems, 698
wxTabCtrl::DeleteItem, 698
wxTabCtrl::GetCurFocus, 698
wxTabCtrl::GetImageList, 698
wxTabCtrl::GetItemCount, 699
wxTabCtrl::GetItemData, 699
wxTabCtrl::GetItemImage, 699
wxTabCtrl::GetItemRect, 699
wxTabCtrl::GetItemText, 699
wxTabCtrl::GetRowCount, 699
wxTabCtrl::GetSelection, 699
wxTabCtrl::HitTest, 700
wxTabCtrl::InsertItem, 700
wxTabCtrl::SetImageList, 701
wxTabCtrl::SetItemData, 701
wxTabCtrl::SetItemImage, 701
wxTabCtrl::SetItemSize, 701
wxTabCtrl::SetItemText, 701
wxTabCtrl::SetPadding, 701
wxTabCtrl::SetSelection, 702
wxTabCtrl::wxTabCtrl, 697
wxTabEvent, 703
wxTabEvent::wxTabEvent, 703
wxTabView, 689
wxTabView::AddTab, 689
wxTabView::CalculateTabWidth, 689
wxTabView::ClearTabs, 690
wxTabView::Draw, 690
wxTabView::FindTabControlForId, 690
wxTabView::FindTabControlForPosition, 690
wxTabView::GetBackgroundBrush, 690
wxTabView::GetBackgroundColour, 691
wxTabView::GetBackgroundPen, 691
wxTabView::GetHighlightColour, 691
wxTabView::GetHighlightPen, 691
wxTabView::GetHorizontalTabOffset, 691
wxTabView::GetNumberOfLayers, 691
wxTabView::GetSelectedTabFont, 691
wxTabView::GetShadowColour, 692
wxTabView::GetShadowPen, 693
wxTabView::GetTabFont, 692
wxTabView::GetTabHeight, 692
wxTabView::GetTabSelectionHeight, 692
wxTabView::GetTabStyle, 692
wxTabView::GetTabWidth, 692
wxTabView::GetTextColour, 692
wxTabView::GetTopMargin, 693
wxTabView::GetVerticalTabTextSpacing, 693
wxTabView::GetViewRect, 693
wxTabView::GetWindow, 693
wxTabView::Layout, 694
wxTabView::OnCreateTabControl, 693
wxTabView::OnEvent, 694
wxTabView::OnTabActivate, 694
wxTabView::OnTabPreActivate, 694
wxTabView::SetBackgroundColour, 694
wxTabView::SetHighlightColour, 694
wxTabView::SetHorizontalTabOffset, 695
wxTabView::SetSelectedTabFont, 695
wxTabView::SetShadowColour, 695
wxTabView::SetTabFont, 695
wxTabView::SetTabSelection, 696
wxTabView::SetTabSelectionHeight, 695
wxTabView::SetTabSize, 695
wxTabView::SetTabStyle, 695
wxTabView::SetTextColour, 696

wxTabView::SetTopMargin, 696
wxTabView::SetVerticalTabTextSpacing, 696
wxTabView::SetViewRect, 696
wxTabView::SetWindow, 696
wxTabView::wxTabView, 689
wxTaskBarIcon, 703
wxTaskBarIcon::~~wxTaskBarIcon, 703
wxTaskBarIcon::IsIconInstalled, 703
wxTaskBarIcon::IsOK, 703
wxTaskBarIcon::OnLButtonDClick, 704
wxTaskBarIcon::OnLButtonDown, 704
wxTaskBarIcon::OnLButtonUp, 704
wxTaskBarIcon::OnMouseMove, 704
wxTaskBarIcon::OnRButtonDClick, 704
wxTaskBarIcon::OnRButtonDown, 704
wxTaskBarIcon::OnRButtonUp, 704
wxTaskBarIcon::RemoveIcon, 705
wxTaskBarIcon::SetIcon, 705
wxTaskBarIcon::wxTaskBarIcon, 703
wxTB_3DBUTTONS, 749
wxTB_FLAT, 749
wxTB_HORIZONTAL, 749
wxTB_VERTICAL, 749
wxTCPClient, 705
wxTCPClient::MakeConnection, 706
wxTCPClient::OnMakeConnection, 706
wxTCPClient::ValidHost, 706
wxTCPClient::wxTCPClient, 705
wxTCPConnection, 707
wxTCPConnection::Advise, 708
wxTCPConnection::Disconnect, 708
wxTCPConnection::Execute, 708
wxTCPConnection::OnAdvise, 708
wxTCPConnection::OnDisconnect, 708
wxTCPConnection::OnExecute, 709
wxTCPConnection::OnPoke, 709
wxTCPConnection::OnRequest, 709
wxTCPConnection::OnStartAdvise, 709
wxTCPConnection::OnStopAdvise, 709
wxTCPConnection::Poke, 710
wxTCPConnection::Request, 710
wxTCPConnection::StartAdvise, 710
wxTCPConnection::StopAdvise, 710
wxTCPConnection::wxTCPConnection, 707
wxTCPServer, 711
wxTCPServer::Create, 711
wxTCPServer::OnAcceptConnection, 711
wxTCPServer::wxTCPServer, 711
wxTE_MULTILINE, 714
wxTE_PASSWORD, 714
wxTE_PROCESS_ENTER, 714
wxTE_READONLY, 714
wxTempFile, 712
wxTempFile::~~wxTempFile, 713
wxTempFile::Commit, 713
wxTempFile::Discard, 713
wxTempFile::IsOpened, 713
wxTempFile::Open, 712
wxTempFile::Write, 713
wxTempFile::wxTempFile, 712
wxTextCtrl, 715
wxTextCtrl::~~wxTextCtrl, 716
wxTextCtrl::AppendText, 723
wxTextCtrl::Clear, 716
wxTextCtrl::Copy, 716
wxTextCtrl::Create, 716
wxTextCtrl::Cut, 716
wxTextCtrl::DiscardEdits, 717
wxTextCtrl::GetInsertionPoint, 717
wxTextCtrl::GetLastPosition, 717
wxTextCtrl::GetLineLength, 717
wxTextCtrl::GetLineText, 718
wxTextCtrl::GetNumberOfLines, 718
wxTextCtrl::GetValue, 718
wxTextCtrl::IsModified, 718
wxTextCtrl::LoadFile, 718
wxTextCtrl::OnChar, 719
wxTextCtrl::OnDropFiles, 719
wxTextCtrl::operator <<, 724
wxTextCtrl::Paste, 720
wxTextCtrl::PositionToXY, 720
wxTextCtrl::Remove, 720
wxTextCtrl::Replace, 721
wxTextCtrl::SaveFile, 721
wxTextCtrl::SetEditable, 721
wxTextCtrl::SetInsertionPoint, 722
wxTextCtrl::SetInsertionPointEnd, 722
wxTextCtrl::SetSelection, 722
wxTextCtrl::SetValue, 722
wxTextCtrl::ShowPosition, 723
wxTextCtrl::WriteText, 723
wxTextCtrl::wxTextCtrl, 715
wxTextCtrl::XYToPosition, 724
wxTextDataObject, 725
wxTextDataObject::GetSize, 725
wxTextDataObject::GetText, 725
wxTextDataObject::SetText, 726
wxTextDataObject::WriteData, 726
wxTextDataObject::WriteString, 726
wxTextDataObject::wxTextDataObject, 725
wxTextDropTarget, 728
wxTextDropTarget::GetFormat, 728
wxTextDropTarget::GetFormatCount, 728
wxTextDropTarget::OnDrop, 729
wxTextDropTarget::OnDropText, 729
wxTextDropTarget::wxTextDropTarget, 728
wxTextEntryDialog, 727
wxTextEntryDialog overview, 920
wxTextEntryDialog::~~wxTextEntryDialog, 727
wxTextEntryDialog::GetValue, 727
wxTextEntryDialog::SetValue, 727
wxTextEntryDialog::ShowModal, 728
wxTextEntryDialog::wxTextEntryDialog, 727
wxTextFile, 733
wxTextFile::~~wxTextFile, 737
wxTextFile::AddLine, 736
wxTextFile::Close, 734
wxTextFile::Eof, 735
wxTextFile::Exists, 733
wxTextFile::GetCurrentLine, 735
wxTextFile::GetEOL, 737
wxTextFile::GetFirstLine, 735
wxTextFile::GetLastLine, 736
wxTextFile::GetLine, 734

wxTextFile::GetLineCount, 734
wxTextFile::GetLineType, 736
wxTextFile::GetName, 736
wxTextFile::GetNextLine, 735
wxTextFile::GetPrevLine, 736
wxTextFile::GoToLine, 735
wxTextFile::GuessType, 736
wxTextFile::InsertLine, 736
wxTextFile::IsOpened, 734
wxTextFile::Open, 734
wxTextFile::operator[], 735
wxTextFile::RemoveLine, 737
wxTextFile::Write, 737
wxTextFile::wxTextFile, 733
wxTextValidator, 730
wxTextValidator::~~wxTextValidator, 730
wxTextValidator::Clone, 730
wxTextValidator::GetExcludeList, 731
wxTextValidator::GetIncludeList, 731
wxTextValidator::GetStyle, 731
wxTextValidator::OnChar, 731
wxTextValidator::SetExcludeList, 731
wxTextValidator::SetIncludeList, 731
wxTextValidator::SetStyle, 731
wxTextValidator::TransferFromWindow, 732
wxTextValidator::TransferToWindow, 732
wxTextValidator::Validate, 732
wxTextValidator::wxTextValidator, 730
wxTHICK_FRAME, 179, 277, 406, 411, 447
wxThread, 738
wxThread::~~wxThread, 738
wxThread::Create, 738
wxThread::Delete, 739
wxThread::GetID, 739
wxThread::GetPriority, 739
wxThread::IsAlive, 739
wxThread::IsMain, 739
wxThread::IsPaused, 740
wxThread::IsRunning, 740
wxThread::Kill, 740
wxThread::OnExit, 740
wxThread::Run, 740
wxThread::SetPriority, 740
wxThread::Sleep, 741
wxThread::This, 741
wxThread::wxThread, 738
wxThread::Yield, 741
wxTime, 742, 743
wxTime::FormatTime, 744
wxTime::GetDay, 743
wxTime::GetDayOfWeek, 743
wxTime::GetHour, 743
wxTime::GetHourGMT, 743
wxTime::GetMinute, 743
wxTime::GetMinuteGMT, 743
wxTime::GetMonth, 744
wxTime::GetSecond, 744
wxTime::GetSecondGMT, 744
wxTime::GetSeconds, 744
wxTime::GetYear, 744
wxTime::IsBetween, 744
wxTime::Max, 745
wxTime::Min, 745
wxTime::operator -, 746
wxTime::operator !=, 746
wxTime::operator +, 746
wxTime::operator +=", 747
wxTime::operator <, 745
wxTime::operator <=, 746
wxTime::operator =, 745
wxTime::operator -=, 747
wxTime::operator ==, 746
wxTime::operator >, 746
wxTime::operator >=, 746
wxTime::operator char*, 745
wxTime::operator wxDate, 745
wxTime::SetFormat, 745
wxTime::wxTime, 742
wxTimer, 747
wxTimer::~~wxTimer, 747
wxTimer::Interval, 747
wxTimer::Notify, 748
wxTimer::Start, 748
wxTimer::Stop, 748
wxTimer::wxTimer, 747
wxTINY_CAPTION_HORIZ, 447
wxTINY_CAPTION_VERT, 447
wxToLower, 876
wxToolBar, 750
wxToolBar::~~wxToolBar, 751
wxToolBar::AddSeparator, 751
wxToolBar::AddTool, 751
wxToolBar::CreateTools, 752
wxToolBar::DrawTool, 753
wxToolBar::EnableTool, 753
wxToolBar::FindToolForPosition, 753
wxToolBar::GetMargins, 754
wxToolBar::GetMaxSize, 755
wxToolBar::GetToolBitmapSize, 754
wxToolBar::GetToolClientData, 755
wxToolBar::GetToolEnabled, 755
wxToolBar::GetToolLongHelp, 756
wxToolBar::GetToolPacking, 756
wxToolBar::GetToolSeparation, 756
wxToolBar::GetToolShortHelp, 756
wxToolBar::GetToolSize, 754
wxToolBar::GetToolState, 757
wxToolBar::Layout, 757
wxToolBar::OnLeftClick, 757
wxToolBar::OnMouseEnter, 758
wxToolBar::OnRightClick, 758
wxToolBar::Realize, 759
wxToolBar::SetMargins, 760
wxToolBar::SetToolBitmapSize, 759
wxToolBar::SetToolLongHelp, 760
wxToolBar::SetToolPacking, 761
wxToolBar::SetToolSeparation, 762
wxToolBar::SetToolShortHelp, 761
wxToolBar::ToggleTool, 762
wxToolBar::wxToolBar, 750
wxToUpper, 876
wxTR_EDIT_LABELS, 763
wxTR_HAS_BUTTONS, 763
wxTrace, 876

WXTRACE, 882
 wxTraceLevel, 877
 WXTRACELEVEL, 883
 wxTransferFileToStream, 852
 wxTransferStreamToFile, 852
 wxTRANSPARENT_WINDOW, 801
 wxTreeCtrl, 763, 764, 765
 wxTreeCtrl::~wxTreeCtrl, 764
 wxTreeCtrl::AddRoot, 764
 wxTreeCtrl::AppendItem, 765
 wxTreeCtrl::Collapse, 765
 wxTreeCtrl::CollapseAndReset, 765
 wxTreeCtrl::Create, 765
 wxTreeCtrl::Delete, 765
 wxTreeCtrl::DeleteAllItems, 765
 wxTreeCtrl::EditLabel, 766
 wxTreeCtrl::EndEditLabel, 766
 wxTreeCtrl::EnsureVisible, 766
 wxTreeCtrl::Expand, 766
 wxTreeCtrl::GetBoundingRect, 767
 wxTreeCtrl::GetChildrenCount, 767
 wxTreeCtrl::GetCount, 767
 wxTreeCtrl::GetEditControl, 767
 wxTreeCtrl::GetFirstChild, 767
 wxTreeCtrl::GetFirstVisibleItem, 768
 wxTreeCtrl::GetImageList, 768
 wxTreeCtrl::GetIndent, 768
 wxTreeCtrl::GetItemData, 768
 wxTreeCtrl::GetItemImage, 768
 wxTreeCtrl::GetItemSelectedImage, 770
 wxTreeCtrl::GetItemText, 768
 wxTreeCtrl::GetLastChild, 769
 wxTreeCtrl::GetNextChild, 769
 wxTreeCtrl::GetNextSibling, 769
 wxTreeCtrl::GetNextVisible, 769
 wxTreeCtrl::GetParent, 770
 wxTreeCtrl::GetPrevSibling, 770
 wxTreeCtrl::GetPrevVisible, 770
 wxTreeCtrl::GetRootItem, 770
 wxTreeCtrl::GetSelection, 770
 wxTreeCtrl::HitTest, 771
 wxTreeCtrl::InsertItem, 771
 wxTreeCtrl::IsBold, 771
 wxTreeCtrl::IsExpanded, 771
 wxTreeCtrl::IsSelected, 772
 wxTreeCtrl::IsVisible, 772
 wxTreeCtrl::ItemHasChildren, 772
 wxTreeCtrl::OnCompareItems, 772
 wxTreeCtrl::PrependItem, 772
 wxTreeCtrl::ScrollTo, 773
 wxTreeCtrl::SelectItem, 773
 wxTreeCtrl::SetImageList, 773
 wxTreeCtrl::SetIndent, 773
 wxTreeCtrl::SetItemBold, 773
 wxTreeCtrl::SetItemData, 773
 wxTreeCtrl::SetItemHasChildren, 773
 wxTreeCtrl::SetItemImage, 774
 wxTreeCtrl::SetItemSelectedImage, 774
 wxTreeCtrl::SetItemText, 774
 wxTreeCtrl::SortChildren, 774
 wxTreeCtrl::Toggle, 774
 wxTreeCtrl::Unselect, 774
 wxTreeCtrl::wxTreeCtrl, 763
 wxTreeEvent, 776
 wxTreeEvent::m_code, 777
 wxTreeEvent::m_itemIndex, 777
 wxTreeEvent::m_oldItem, 777
 wxTreeEvent::m_pointDrag, 777
 wxTreeEvent::wxTreeEvent, 776
 wxTreeItemData, 775
 wxTreeItemData::~wxTreeItemData, 775
 wxTreeItemData::GetId, 775
 wxTreeItemData::SetId, 776
 wxTreeItemData::wxTreeItemData, 775
 wxUnix2DosFilename, 849
 wxUpdateUIEvent, 778
 wxUpdateUIEvent::Check, 779
 wxUpdateUIEvent::Enable, 779
 wxUpdateUIEvent::GetChecked, 779
 wxUpdateUIEvent::GetEnabled, 780
 wxUpdateUIEvent::GetSetChecked, 780
 wxUpdateUIEvent::GetSetEnabled, 780
 wxUpdateUIEvent::GetSetText, 780
 wxUpdateUIEvent::GetText, 780
 wxUpdateUIEvent::m_checked, 778
 wxUpdateUIEvent::m_enabled, 778
 wxUpdateUIEvent::m_setChecked, 779
 wxUpdateUIEvent::m_setEnabled, 779
 wxUpdateUIEvent::m_setText, 779
 wxUpdateUIEvent::m_text, 779
 wxUpdateUIEvent::SetText, 780
 wxUpdateUIEvent::wxUpdateUIEvent, 778
 wxURL, 781
 wxURL::~wxURL, 781
 wxURL::GetError, 782
 wxURL::GetInputStream, 782
 wxURL::GetPath, 782
 wxURL::GetProtocol, 781
 wxURL::GetProtocolName, 781
 wxURL::SetDefaultProxy, 782
 wxURL::SetProxy, 783
 wxURL::wxURL, 781
 wxValidator, 784
 wxValidator::~wxValidator, 784
 wxValidator::Clone, 784
 wxValidator::GetWindow, 784
 wxValidator::SetBellOnError, 784
 wxValidator::SetWindow, 784
 wxValidator::TransferFromWindow, 785
 wxValidator::TransferToWindow, 785
 wxValidator::Validate, 785
 wxValidator::wxValidator, 784
 wxVariant, 786, 787
 wxVariant::~wxVariant, 787
 wxVariant::Append, 787
 wxVariant::ClearList, 787
 wxVariant::Delete, 788
 wxVariant::GetBool, 788
 wxVariant::GetChar, 788
 wxVariant::GetCount, 787
 wxVariant::GetData, 788
 wxVariant::GetDate, 788
 wxVariant::GetDouble, 788
 wxVariant::GetLong, 788

wxVariant::GetName, 789
wxVariant::GetString, 789
wxVariant::GetTime, 789
wxVariant::GetType, 789
wxVariant::GetVoidPtr, 789
wxVariant::Insert, 789
wxVariant::IsNull, 789
wxVariant::IsType, 790
wxVariant::MakeNull, 790
wxVariant::MakeString, 790
wxVariant::Member, 790
wxVariant::NullList, 790
wxVariant::operator !=, 792
wxVariant::operator [], 792
wxVariant::operator =, 790
wxVariant::operator ==, 791
wxVariant::operator char, 792
wxVariant::operator double, 793
wxVariant::operator void*, 793
wxVariant::operator wxDate, 793
wxVariant::operator wxString, 793
wxVariant::operator wxTime, 793
wxVariant::SetData, 790
wxVariant::wxVariant, 786
wxVariantData, 794
wxVariantData::Copy, 794
wxVariantData::Eq, 794
wxVariantData::GetType, 794
wxVariantData::Read, 794
wxVariantData::Write, 794
wxVariantData::wxVariantData, 794
wxView, 796
wxView overview, 938
wxView::~~wxView, 796
wxView::Activate, 796
wxView::Close, 796
wxView::GetDocument, 796
wxView::GetDocumentManager, 796
wxView::GetFrame, 797
wxView::GetViewName, 797
wxView::m_viewDocument, 795
wxView::m_viewFrame, 795
wxView::m_viewTypeName, 795
wxView::OnActivateView, 797
wxView::OnChangeFilename, 797
wxView::OnClose, 797
wxView::OnCreate, 797
wxView::OnCreatePrintout, 798
wxView::OnUpdate, 798
wxView::SetDocument, 798
wxView::SetFrame, 798
wxView::SetViewName, 799
wxView::wxView, 796
wxVSCROLL, 411, 801
wxWave, 799
wxWave::~~wxWave, 799
wxWave::Create, 799
wxWave::IsOk, 800
wxWave::Play, 800
wxWave::wxWave, 799
wxWindow, 801
wxWindow::~~wxWindow, 802
wxWindow::AddChild, 802
wxWindow::CaptureMouse, 802
wxWindow::Center, 803
wxWindow::Centre, 803
wxWindow::Clear, 803
wxWindow::ClientToScreen, 803
wxWindow::Close, 804
wxWindow::ConvertDialogToPixels, 805
wxWindow::ConvertPixelsToDialog, 806
wxWindow::Destroy, 806
wxWindow::DestroyChildren, 807
wxWindow::DragAcceptFiles, 807
wxWindow::Enable, 807
wxWindow::FindFocus, 807
wxWindow::FindWindow, 808
wxWindow::Fit, 808
wxWindow::GetBackgroundColour, 808
wxWindow::GetCharHeight, 808
wxWindow::GetCharWidth, 809
wxWindow::GetChildren, 809
wxWindow::GetClientSize, 809
wxWindow::GetConstraints, 809
wxWindow::GetDefaultItem, 810
wxWindow::GetDropTarget, 810
wxWindow::GetEventHandler, 810
wxWindow::GetFont, 810
wxWindow::GetForegroundColour, 810
wxWindow::GetGrandParent, 811
wxWindow::GetHandle, 811
wxWindow::GetId, 811
wxWindow::GetLabel, 812
wxWindow::GetName, 812
wxWindow::GetParent, 812
wxWindow::GetPosition, 811
wxWindow::GetRect, 813
wxWindow::GetReturnCode, 813
wxWindow::GetScrollPos, 813
wxWindow::GetScrollRange, 814
wxWindow::GetScrollThumb, 813
wxWindow::GetSize, 814
wxWindow::GetTextExtent, 814
wxWindow::GetTitle, 815
wxWindow::GetUpdateRegion, 815
wxWindow::GetWindowStyleFlag, 816
wxWindow::InitDialog, 816
wxWindow::IsEnabled, 816
wxWindow::IsRetained, 816
wxWindow::IsShown, 816
wxWindow::Layout, 816
wxWindow::LoadFromResource, 817
wxWindow::Lower, 817
wxWindow::MakeModal, 817
wxWindow::Move, 818
wxWindow::OnActivate, 818
wxWindow::OnChar, 819
wxWindow::OnCharHook, 819
wxWindow::OnClose, 821
wxWindow::OnCloseWindow, 821
wxWindow::OnCommand, 820
wxWindow::OnDropFiles, 822
wxWindow::OnEraseBackground, 822
wxWindow::OnIdle, 824

wxWindow::OnInitDialog, 825
 wxWindow::OnKeyDown, 823
 wxWindow::OnKeyUp, 823
 wxWindow::OnKillFocus, 824
 wxWindow::OnMenuCommand, 825
 wxWindow::OnMenuHighlight, 826
 wxWindow::OnMouseEvent, 826
 wxWindow::OnMove, 827
 wxWindow::OnPaint, 827
 wxWindow::OnScroll, 828
 wxWindow::OnSetFocus, 829
 wxWindow::OnSize, 829
 wxWindow::OnSysColourChanged, 830
 wxWindow::PopEventHandler, 830
 wxWindow::PopupMenu, 830
 wxWindow::PushEventHandler, 831
 wxWindow::Raise, 832
 wxWindow::Refresh, 832
 wxWindow::ReleaseMouse, 832
 wxWindow::RemoveChild, 832
 wxWindow::ScreenToClient, 833
 wxWindow::ScrollWindow, 833
 wxWindow::SetAcceleratorTable, 834
 wxWindow::SetAutoLayout, 834
 wxWindow::SetBackgroundColour, 834
 wxWindow::SetClientSize, 835
 wxWindow::SetConstraints, 836
 wxWindow::SetCursor, 835
 wxWindow::SetDropTarget, 837
 wxWindow::SetEventHandler, 836
 wxWindow::SetFocus, 837
 wxWindow::SetFont, 837
 wxWindow::SetForegroundColour, 837
 wxWindow::SetId, 838
 wxWindow::SetName, 838

wxWindow::SetPalette, 838
 wxWindow::SetReturnCode, 839
 wxWindow::SetScrollbar, 839
 wxWindow::SetScrollPos, 840
 wxWindow::SetSize, 841
 wxWindow::SetSizeHints, 842
 wxWindow::SetTitle, 843
 wxWindow::Show, 843
 wxWindow::TransferDataFromWindow, 843
 wxWindow::TransferDataToWindow, 844
 wxWindow::Validate, 844
 wxWindow::WarpPointer, 844
 wxWindow::wxWindow, 801
 wxWindowDC, 845
 wxWindowDC::wxWindowDC, 845
 wxWindows 1.xx compatibility functions, 659
 wxWindows predefined command identifiers, 941
 wxWriteResource, 877
 wxYield, 878

—X—

x, 547
 x, 504, 548, 597
 Xor, 567
 XYToPosition, 724

—Y—

y, 505, 547, 597
 y, 548
 Yield, 741
 YieldSock, 621