

# **User Manual for wxWindows 1.68: a portable C++ GUI toolkit**

Julian Smart, Anthemion Software

October 1997

## Contents

<b>1. Introduction .....</b>	<b>1</b>
1.1. What is wxWindows? .....	1
1.2. Why another cross-platform development tool? .....	1
1.3. wxWindows requirements .....	3
1.4. Availability and location of wxWindows .....	4
1.5. Acknowledgments .....	4
<b>2. Resource guide .....</b>	<b>6</b>
2.1. wxCLIPS .....	6
2.2. WinSock for wxWindows .....	6
2.3. GWX: DevGuide to wxWindows .....	6
2.4. wxXPM .....	6
2.5. CURSES port .....	6
2.6. Imakefiles .....	6
2.7. QDB database .....	7
2.8. MEWEL .....	7
2.9. wxWindows mailing list .....	7
2.10. World Wide Web .....	8
<b>3. Overview and comparison with other GUI models .....</b>	<b>9</b>
3.1. Windows .....	9
3.2. Dialog boxes .....	10
3.3. Menus .....	10
3.4. Events .....	11
3.5. Keyboard input .....	12
3.6. Repainting of windows .....	12
3.7. Scrolling .....	12
3.8. Printing .....	13
3.9. Programmatic versus interactive GUI building .....	13
3.10. Dimensions .....	14
3.11. Colour .....	14
3.12. On-line help .....	15
3.13. User preferences .....	15
3.14. Interprocess Communication .....	15
<b>4. Multi-platform development with wxWindows .....</b>	<b>17</b>
4.1. Code structure .....	17
4.2. Include files .....	17

4.3. Libraries.....	18
4.4. Configuration .....	18
4.5. Makefiles .....	20
4.6. Windows-specific files.....	21
4.7. Memory models and memory allocation .....	22
4.8. Dynamic Link Libraries.....	23
4.9. Conditional compilation .....	23
4.10. Building on-line help.....	23
4.11. C++ issues.....	24
4.12. File handling .....	25
4.13. Large amounts of global data.....	25
<b>5. Utilities supplied with wxWindows .....</b>	<b>27</b>
5.1. wxBuilder.....	27
5.2. wxToolBar .....	27
5.3. wxHelp .....	27
5.4. hyText .....	28
5.5. wxCLIPS.....	28
5.6. PrologIO .....	29
5.7. Tex2RTF .....	29
5.8. wxTreeLayout .....	30
5.9. wxGraphLayout.....	30
5.10. wxImage .....	30
5.11. DIB .....	30
5.12. rcParser.....	30
5.13. MFUTILS .....	30
5.14. Colours .....	30
<b>6. Bugs and future directions.....</b>	<b>31</b>
6.1. Bugs.....	31
6.2. Future directions .....	34
<b>7. Tutorial .....</b>	<b>36</b>
7.1. The demo programs.....	36
<b>8. Programming strategies .....</b>	<b>41</b>
8.1. Strategies for reducing programming errors .....	41
8.2. Strategies for portability .....	41
8.3. Strategies for debugging .....	41
<b>9. Alphabetical class reference .....</b>	<b>44</b>

9.1. wxApp: wxObject .....	44
9.2. wxBitmap: wxObject .....	48
9.3. wxBrush: wxObject .....	51
9.4. wxBrushList: wxList .....	53
9.5. wxButton: wxItem .....	54
9.6. wxButtonBar: wxToolBar .....	55
9.7. wxCanvas: wxWindow .....	57
9.8. wxCanvasDC: wxDC .....	68
9.9. wxCheckBox: wxItem .....	68
9.10. wxChoice: wxItem .....	69
9.11. wxClassInfo .....	72
9.12. wxClient: wxIPCObject .....	74
9.13. wxClipboard: wxObject .....	75
9.14. wxClipboardClient: wxObject .....	76
9.15. wxColour: wxObject .....	76
9.16. wxColourData: wxObject .....	78
9.17. wxColourDatabase: wxObject .....	79
9.18. wxColourDialog: wxDialogBox .....	80
9.19. wxColourMap: wxObject .....	81
9.20. wxComboBox: wxItem .....	82
9.21. wxCommand: wxObject .....	86
9.22. wxCommandEvent: wxEvent .....	88
9.23. wxCommandProcessor: wxObject .....	89
9.24. wxConnection: wxObject .....	91
9.25. wxCursor: wxBitmap .....	94
9.26. wxDatabase: wxObject .....	96
9.27. wxDate: wxObject .....	101
9.28. wxDC: wxObject .....	108
9.29. wxDebugContext .....	120
9.30. wxDebugStreamBuf: streambuf .....	123
9.31. wxDialogBox: wxPanel .....	123
9.32. wxDocChildFrame: wxFrame .....	126
9.33. wxDocManager: wxEvtHandler .....	128
9.34. wxDocParentFrame: wxFrame .....	134
9.35. wxDocTemplate: wxObject .....	135
9.36. wxDocument: wxEvtHandler .....	140
9.37. wxEnhDialogBox: wxDialogBox .....	146
9.38. wxEvent: wxObject .....	149
9.39. wxEvtHandler: wxObject .....	150
9.40. wxFileHistory: wxObject .....	156

9.41. wxFont: wxObject .....	158
9.42. wxFontData: wxObject .....	160
9.43. wxFontDialog: wxDialogBox .....	162
9.44. wxFontList: wxList .....	163
9.45. wxFontNameDirectory: wxObject .....	164
9.46. wxForm: wxObject .....	166
9.47. wxFormItem: wxObject .....	172
9.48. wxFrame: wxWindow .....	172
9.49. wxFunction .....	180
9.50. wxGauge: wxItem .....	180
9.51. wxGroupBox: wxItem .....	182
9.52. wxIcon: wxBitmap .....	183
9.53. wxHashTable: wxObject .....	185
9.54. wxHelpInstance: wxClient .....	187
9.55. wxIndividualLayoutConstraint: wxObject .....	189
9.56. wxIntPoint: wxObject .....	191
9.57. wxItem: wxWindow .....	191
9.58. wxKeyEvent: wxEvent .....	193
9.59. wxLayoutConstraints: wxObject .....	196
9.60. wxList: wxObject .....	197
9.61. wxListBox: wxItem .....	201
9.62. wxMemoryDC: wxCanvasDC .....	205
9.63. wxMenu: wxWindow .....	206
9.64. wxMenuBar: wxWindow .....	209
9.65. wxMessage: wxItem .....	211
9.66. wxMetaFile: wxObject .....	212
9.67. wxMetaFileDC: wxDC .....	213
9.68. wxMouseEvent: wxEvent .....	214
9.69. wxMultiText: wxText .....	219
9.70. wxNode: wxObject .....	221
9.71. wxObject .....	221
9.72. wxPageSetupData: wxObject .....	223
9.73. wxPageSetupDialog: wxDialogBox .....	227
9.74. wxPanel: wxCanvas .....	228
9.75. wxPanelDC: wxDC .....	235
9.76. wxPathList: wxList .....	235
9.77. wxPen: wxObject .....	236
9.78. wxPenList: wxList .....	239
9.79. wxPoint: wxObject .....	240
9.80. wxPostScriptDC: wxDC .....	240

9.81. wxPreviewCanvas: wxCanvas.....	241
9.82. wxPreviewControlBar: wxPanel.....	241
9.83. wxPreviewFrame: wxFrame .....	244
9.84. wxPrintData: wxObject.....	245
9.85. wxPrintDialog: wxDialogBox.....	248
9.86. wxPrinter: wxObject .....	249
9.87. wxPrinterDC: wxDC .....	250
9.88. wxPrintout: wxObject .....	251
9.89. wxPrintPreview: wxObject .....	253
9.90. wxQueryCol: wxObject.....	256
9.91. wxQueryField: wxObject .....	259
9.92. wxRadioBox: wxItem.....	260
9.93. wxRadioButton: wxItem.....	263
9.94. wxRecordSet: wxObject.....	264
9.95. wxScreenDC: wxCanvasDC.....	275
9.96. wxScrollBar: wxItem.....	276
9.97. wxServer: wxIPCObject .....	277
9.98. wxSlider: wxItem.....	278
9.99. wxSplitterWindow: wxCanvas.....	280
9.100. wxString: wxObject .....	286
9.101. wxStringList: wxList.....	297
9.102. wxText: wxItem .....	299
9.103. wxTextWindow: wxWindow .....	302
9.104. wxTimer: wxObject.....	307
9.105. wxToolBar: wxPanel .....	308
9.106. wxTypeTree: wxList .....	313
9.107. wxUpdateIterator: wxObject .....	314
9.108. wxView: wxEvtHandler.....	315
9.109. wxWindow: wxObject .....	319
<b>10. Functions .....</b>	<b>328</b>
10.1. File functions .....	328
10.2. String functions .....	332
10.3. Dialog functions .....	333
10.4. GDI functions.....	335
10.5. System event functions .....	336
10.6. Printer settings.....	338
10.7. Clipboard functions .....	340
10.8. Miscellaneous functions .....	342
10.9. Macros.....	352

---

10.10. wxWindows resource functions .....	355
<b>11. Classes by category.....</b>	<b>359</b>
11.1. Managed windows .....	359
11.2. Subwindows.....	359
11.3. Common dialogs.....	359
11.4. Panel items.....	359
11.5. Window layout .....	360
11.6. Device contexts .....	360
11.7. Graphics device interface.....	360
11.8. Events .....	361
11.9. Data structures .....	361
11.10. Run-time class information system.....	361
11.11. Debugging features.....	361
11.12. Interprocess communication.....	362
11.13. Document/view framework .....	362
11.14. Printing framework .....	362
11.15. Database classes.....	363
11.16. Miscellaneous .....	363
11.17. wxString member functions .....	363
<b>12. Topic overviews .....</b>	<b>367</b>
12.1. Window styles.....	367
12.2. Run time class information overview .....	370
12.3. Document/view overview.....	372
12.4. Printing overview.....	377
12.5. Interprocess communication overview.....	378
12.6. Font overview .....	381
12.7. Device context overview.....	383
12.8. wxApp overview.....	383
12.9. Bitmaps overview.....	384
12.10. Dialog box overview .....	385
12.11. Common dialogs overview .....	385
12.12. Constraints overview .....	388
12.13. Event handling overview .....	390
12.14. Toolbar overview.....	391
12.15. Database classes overview.....	393
12.16. Debugging overview.....	397
12.17. wxString overview .....	399
12.18. Writing a wxWindows application: a rough guide .....	413
12.19. The wxWindows resource system .....	414

12.20. Notes on using the reference .....	419
12.21. wxSplitterWindow overview .....	420
<b>References .....</b>	<b>423</b>
<b>Index .....</b>	<b>425</b>



## **Copyright notice**

Copyright (c) 1996 Artificial Intelligence Applications Institute, The University of Edinburgh

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

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL THE ARTIFICIAL INTELLIGENCE APPLICATIONS INSTITUTE OR THE UNIVERSITY OF EDINBURGH BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

## 1. Introduction

### 1.1. What is wxWindows?

wxWindows is a class library for C++ providing GUI (Graphical User Interface) and other facilities on more than one platform. It currently supports subsets of Open Look (XView), Motif and MS Windows (including Windows NT). It contains around 60 classes, with 650 public functions.

wxWindows was originally developed at the Artificial Intelligence Applications Institute, University of Edinburgh, for internal use on a medium-sized project: a hypertext-based knowledge-acquisition and diagramming tool called HARDY. wxWindows has been released into the public domain in the hope that others will also find it useful.

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 exist for cross-platform development (such as CommonView and XVT++), none met all of the following criteria:

1. low price
2. source availability
3. simplicity of programming
4. support for GCC (GNU C++)
5. support for interprocess communication

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 brothers, and 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. wxWindows is likely to evolve to allow more sophisticated GUI use on an increasing number of platforms.

In writing wxWindows, completeness has inevitably been traded for portability and simplicity of programming. For projects which do not need uncompromisingly polished interfaces, this tradeoff seems well worthwhile given the productivity benefits.

wxWindows currently maps to four native APIs: XView (Open Look), Motif, MS Windows and Windows NT. Under UNIX, wxWindows has been tested most thoroughly on Sun workstations, but users have confirmed that it compiles on other machines (including some running UNIX System V). This covers a very large proportion of machines in use today. An Apple Macintosh version is being worked on by an external contributor, and an early demonstrator is expected around September 1994.

In addition to GUI needs, wxWindows also supports a subset of DDE (Dynamic Data Exchange) on both the PC and UNIX. A simple object-oriented model of clients, servers and connections is used, making it easy to write programs which communicate synchronously. Under Windows, other non-wxWindows programs may still communicate with wxWindows programs and vice versa; under UNIX, non-wxWindows programs just have to conform to a simple protocol when

communicating via sockets with wxWindows programs.

On the PC, the tested compilers for wxWindows are Microsoft C/C++ Version 7, Visual C++ 1.5, Visual C++ 2.0, Borland C++ 4.x, Watcom C++ 10.5 (WIN32 mode). Makefiles for some of these are provided.

Under UNIX, Sun C++, GNU C++ (GCC) and AT&T C++ and others are known to work with wxWindows. See also *Requirements* (page 3).

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, it provides access to most of the functionality a GUI program normally requires, plus some extras such as form construction, interprocess communication and PostScript output, and can of course be extended as needs dictate. As a bonus, it provides an arguably cleaner interface to XView, Motif and MS Windows than the native APIs. Programmers may find it worthwhile to use wxWindows even if they are developing on only one platform.

Here is a summary of some of the advantages of wxWindows:

- Low cost (free, in fact!)
- You get the source.
- Several example programs.
- Over 200 pages of printable and on-line documentation.
- Simple-to-use, object-oriented API.
- No more messing with arcane X window calls under XView or Motif.
- Graphics calls include splines, polylines, rounded rectangles, etc.
- XView-style panel item layout, plus a constraint-based layout option.
- Print/preview and document/view architectures.
- Status line facility.
- Easy, object-oriented interprocess comms (DDE subset) under UNIX and MS Windows.
- Encapsulated PostScript generation under UNIX, normal MS Windows printing on the PC.
- Virtually automatic MDI support under Windows.
- Can be used to create DLLs under Windows, dynamic libraries on the Sun.
- Support for MS Windows printer and file common dialogs, with equivalents for UNIX.
- Under MS Windows, support for creating metafiles and copying them to the clipboard.
- Programmatic form facility for building form-like screens fast, with constraints on values.
- Hypertext help facility, with an API for invocation from applications.
- wxBuilder for building simple interfaces interactively, and generating C++ code.

And here are some of the important downsides, so you can assess wxWindows's applicability to your needs:

- No commercial support (but Internet support can be better!)
- XView-style restrictions on parent-child relationships (though these are being relaxed for non-XView platforms)
- Minimal colourmap support.
- No OLE-2 support as yet.

Figure 1 shows a demo application (described in the tutorial chapter) running under X.

Figure 2 shows the demo application running under Microsoft Windows 3.1.

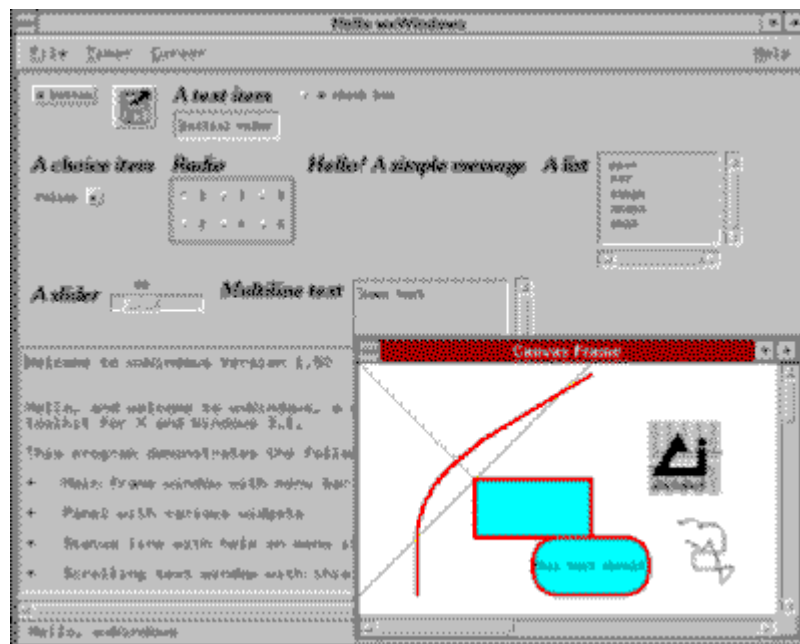


Figure 1: Demo program running under X

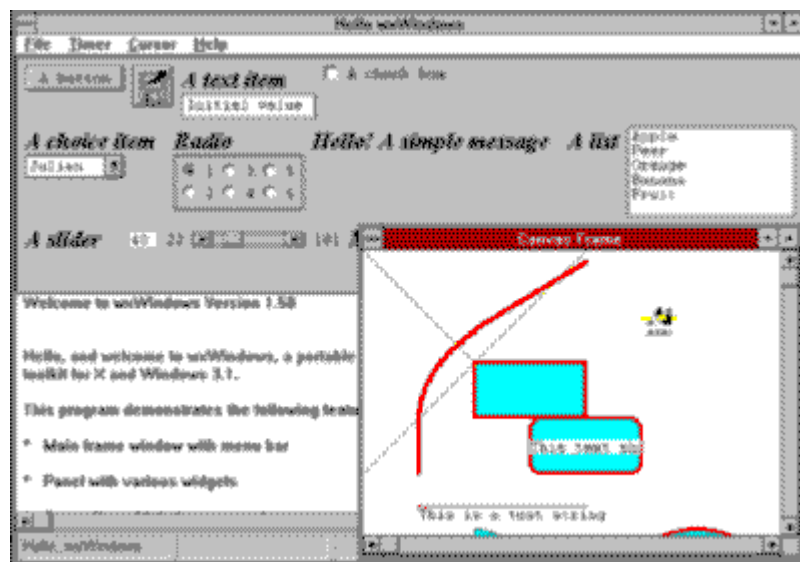


Figure 2: Demo program running under Windows 3.1

### 1.3. wxWindows requirements

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

(a) PC:

1. A 386SX or higher PC running MS Windows or Windows NT.
2. Microsoft C/C++ version 7 or above, or Borland C++ version 3.1 (other compilers may work).
3. At least 10 MB of disk space.

(b) UNIX:

1. GNU C++ version 2.1 or later, or compatible compiler (such as Sun C++ or AT&T C++)
2. A Sun or other workstation supporting GNU C++ and either XView 3.x or Motif 1.2.x
3. At least 20 MB of disk space.

## 1.4. Availability and location of wxWindows

wxWindows is currently available from the Artificial Intelligence Applications Institute by anonymous FTP. FTP to:

`ftp.aiai.ed.ac.uk/pub/packages/wxwin`

## 1.5. Acknowledgments

Thanks are due to the AIAI for being willing to release wxWindows into the public domain, and to my wife Harriet Smart for her patience while I worked on wxWindows after hours.

The Internet has been an essential prop when coming up against tricky XView, Motif and MS Windows problems. Thanks to those who answered my queries or submitted bug fixes and enhancements; wxWindows is very much a team effort.

Hermann Dunkel contributed XPM support; Arthur Seaton wrote the memory checking code; Olaf Klein and Patrick Halke wrote the ODBC classes; Harri Pasanen and Robin Dunn wrote wxPython and contributed to the wxExtend library.

Markus Holzem wrote the excellent Xt port. Jonathan Tonberg, Bill Hale, Cecil Coupe, Thomaso Paoletti, Thomas Fettig, and others slaved away writing the Mac port. Keith Gary Boyce ported wxWindows to the free GNU-WIN32 compiler, refusing to give up when I suggested taking shortcuts.

Many thanks also to: Timothy Peters, Jamshid Afshar, Patrick Albert, C. Buckley, Robin Corbet, Harco de Hilster, Josep Fortiana, Torsten Liermann, Tatu Männistö, Ian Perrigo, Giordano Pezzoli, Petr Smilauer, Neil Smith, Kari Systä, Jyrki Tuomi, Edward Zimmermann, Ian Brown, and many others.

'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.

I also acknowledge the author of XFIG, the excellent UNIX drawing tool, from the source of which I have pinched 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.*

wxCLIPS (now distributed separately) builds on NASA's CLIPS expert system shell, a paradigm of portability and a wonderful piece of (nearly) free software.

## 2. Resource guide

This is a list of other sources of information and software related to wxWindows. For information of supplementary libraries in the wxWindows distribution, see *Utilities* (page 27).

### 2.1. wxCLIPS

wxCLIPS is distributed separately from wxWindows. It is available from the /pub/wxclips directory of the AIAI ftp site; CLIPS2C, a partial CLIPS to C++ converter, will be made available at the same location.

### 2.2. WinSock for wxWindows

Giordano Pezzoli has implemented a WinSock version of the wxWindows DDE implementation. This code enables client-server applications to be written linking UNIX and PC platforms.

It is available in the /pub/wxwin/contrib directory of the AIAI ftp site. as wxsocket.tar.Z or wxsocket.zip. Note that this is not necessarily the most up-to-date version.

### 2.3. GWX: DevGuide to wxWindows

This is a Sun DevGuide to wxWindows generator. It is available in the /pub/wxwin/contrib directory of the AIAI ftp site, as gw10.zip. Note that this is not necessarily the most up-to-date version.

### 2.4. wxXPM

This package, adapted for wxWindows use by Hermann Dunkel, adds the ability for wxWindows programs to read in XPM colour bitmaps under both X and Windows. There's a neat little animation demo, and a large selection of colour pixmaps.

It is available in the /pub/packages/wxwin/tools directory of the AIAI ftp site, in wxxp1, and from 1.61 on, in the main wxWindows distribution itself.

From wxWindows 1.61, support for using wxXPM is built into the wxBitmap class (see the class reference). There is also a Windows-hosted tool, xpmshow.exe, for showing and converting between XPM and BMP files.

By default, support for wxXPM is switched off. Edit wx\_setup.h to enable wxXPM support.

### 2.5. CURSES port

Harco de Hilster is working on a CURSES port of wxWindows. A beta is available in the /pub/packages/wxwin/contrib directory of the AIAI ftp site, as wxcurs.tar.gz. Note that this is not necessarily the most up-to-date version.

### 2.6. Imakefiles

Imakefiles are highly flexible in dealing with different sites and configurations. It has been suggested that wxWindows makefiles should be Imakefiles. Perhaps one day!

Meanwhile, Patrick Albert's Imake kit is in /pub/packages/wxwin/contrib as imake.zip. Note that it is not up to date with respect to wxWindows, but should give a head start in using Imakefiles.

## 2.7. QDB database

QDB is a user-contributed database that can read and write DBASE files. As the author points out, the interface has not had much time spent on it. However, it could be a good basis for a student project.

QDB is available in the /pub/packages/wxwin/contrib directory of the AIAI ftp site as qdb.zip. The author, Dave Curtis, can be contacted at dcurtis@hgmp.mrc.ac.uk.

## 2.8. MEWEL

MEWEL is a commercial library from Magma that allows Windows source code to be compiled for DOS graphics, DOS text and OS/2 text mode. Magma use wxWindows as one of their demonstrations; a beta of version 1.50 apparently compiled with no modifications.

See below for contact information.

```
                                The MEWEL Window System
Portable User Interface Library for Text and Graphics and Microsoft
Windows

Magma Systems
15 Bodwell Terrace
Millburn, New Jersey 07041
(201) 912-0192 (voice)
(201) 912-0668 (24 hour BBS, USR HST Dual Standard, 9600-1200 baud, N-
8-1)
(201) 912-0103 (fax for orders only)

CIS : 75300,2062. To get to our conference, GO MAGMA.
BIX : magma. Our conference is the 'magma' conference.
Internet : 75300.2062@compuserve.com (preferred) or magma@bix.com.
```

```
Thank you very much for expressing your interest in Magma Systems'
MEWEL
user interface library. MEWEL 4.0 gives your application complete
portability
between Microsoft Windows, DOS text, DOS graphics, OS/2 text, UNIX
text,
VMS text, and soon, OSF Motif. Since MEWEL's API is compatible with the
standard Microsoft WIndows API, you can easily port programs between
Win-
dows and any of the environments which MEWEL supports, even if you
already
have existing Windows programs.
```

```
Programmers can develop new stand-alone applications using MEWEL, or
can port
their existing Windows applications to any of the platforms which MEWEL
supports.
```

## 2.9. wxWindows mailing list

There are discussion and announcements mailing lists for users or potential users of wxWindows. Mail one of the following addresses to subscribe to the relevant list. These lists are automated. Send a message with the message body:



```
subscribe which-list [your-mail-address]
```

or

```
unsubscribe which-list [your-mail-address]
```

where which-list is wxwin-users or wxwin-announce, and your-mail-address is your regular email address if you need to specify it explicitly. To get help on the syntax, send a message with 'help' in the message body.

If there is a problem, please mail Timothy Peters (tim.peters@nene.ac.uk).

- email **wxwin-users-request@babbage.eng.nene.ac.uk** to subscribe to wxwin-users discussion list
- email **wxwin-announce-request@babbage.eng.nene.ac.uk** to subscribe to wxwin-announce mailing list

General discussions take place on wxwin-users; wxwin-announce is for people preferring lower bandwidth, and I will always send announcements to wxwin-users as well as wxwin-announce. So there's not usually a need to subscribe to both.

## 2.10. World Wide Web

The wxWindows home page is at:

```
http://web.ukonline.co.uk/julian.smart/wxwin
```

If you wish to set up a local HTML version of the wxWindows documentation, to be viewed with Mosaic in preference to wxHelp, you can compile Tex2RTF in order to produce the HTML files from the LaTeX sources. The target 'html' exists in the makefiles for most packages in the wxWindows toolset, although they will need to be edited to copy the resulting HTML files to a sensible place.

### 3. Overview and comparison with other GUI models

wxWindows takes elements of other GUI APIs, and adds some elements of its own. Of course, it cannot hope to cover every (or even one) native API completely. The following sections discuss different GUI models and compare these with what wxWindows provides.

#### 3.1. Windows

An application presents the user primarily with a series of windows. A window can be made up of a *frame* with one or more subwindows. This is like the XView model, rather than Motif or MS Windows where child windows may be nested to any depth. The frame/subwindow method was chosen since XView required it, and because it is a useful simplifying assumption, imposing few restrictions in practise. However, from version 1.61, wxWindows relaxes some of these restrictions for platforms that allow it. Text windows and canvases may be placed on a panel under Windows and Motif, and under Windows, Motif and XView wxPanel is a subclass of wxCanvas, inheriting most of its properties including the ability to draw to a device context. Work is underway to simulate subwindow nesting under XView.

Note that the splitting of canvas views (as in the Open Look standard and some other platforms) is not allowed in wxWindows. It is envisaged that this will be eventually be introduced into wxWindows.

A frame window may be parentless or a child of another frame, and may be iconized. It may have a *menu bar*, a row of pull-down menus along the top of the frame. Subwindows within a frame come in three varieties: the *panel*, *canvas* and *text subwindow*. A panel is used for buttons, lists and other such user input items, while a canvas is used for drawing graphics such as lines and shapes. Panel items pass high-level notification of user interaction to the program, whereas any interaction with objects on a canvas must be programmed at a lower level.

In MS Windows, canvases need to be painted using a handle to a 'device context'. The purpose of this is to enable the same code to draw into a number of different devices, such as printers, windows, bitmaps and metafiles. wxWindows also has the notion of a device context, but most drawing commands may also be directly issued to a canvas for convenience. An application which implements a function to draw into the base device context, **wxDC** will be able to pass any device context object to this drawing function, including **wxPostScriptDC**, **wxCanvasDC**, **wxMetaFileDC**, and **wxMemoryDC**.

wxWindows defines several objects required for drawing on canvases: colours, fonts, pens and brushes. Neither XView nor Motif provides pens and brushes, which in MS Windows allow the selection of different predefined 'drawing tools' with certain characteristics (line thickness, colour, fill style etc.). Normally in X there is a limited range of fonts (scaleable fonts are a recent addition), while in MS Windows an infinite selection of sizes is available thanks to the scaleable TrueType font system. Under XView and Motif, wxWindows chooses the closest matching font.

Recent releases of wxWindows remove some of the limitations of the panel/canvas separation, and now wxPanel inherits from wxCanvas, and has its own panel device context for drawing graphics on. XView does not directly support this model but wxWindows uses lower-level X calls to implement the functionality.

##### 3.1.1. MDI versus SDI

In MS Windows, a popular technique is to use the MDI (Multiple Document Interface) style, where the application window has a number of iconizable document windows which fit within it. This can save clutter on the desktop since iconizing or moving the parent window iconizes or moves all the

child windows. MDI contrasts with SDI (Single Document Interface) in which windows are not constrained within one parent window, and normal practice is to run one instance of the application per document. Since wxWindows uses the large model, and large model programs may be limited to one instance at a time (unless only one data segment is used), it makes sense to offer MDI support. See *Memory Models* (page 22).

I have decided to include largely automatic, albeit relatively inflexible, support for MDI for these reasons:

- Since it is likely that wxWindows programmers are writing for multiple platforms, and only MS Windows supports MDI, they will not wish to spend much or any time implementing MDI features.
- The difference between writing MDI and SDI applications has been reduced to almost zero so even the beginner can easily produce MDI programs.
- What wxWindows's auto-MDI cannot handle is hardly worth the extra effort anyway.

In wxWindows, one argument of the frame constructor is used to indicate whether the frame is an SDI frame, an MDI parent frame or an MDI child frame. The default is to create an SDI frame. Once MDI frames are created, everything else is automatic, including appending the usual **Window** menu option, and moving a child menu to the parent frame when the child frame is activated. The choice of menu items for an MDI child frame will differ slightly, usually including menus (such as Quit) which would normally be on the main SDI window, but this only requires a small amount of extra application code.

An MDI toolbar is supported from release 1.61, by using the **wxFrame::SetToolBar** member function.

The demo program **mdi** shows how a program can easily be made switchable between SDI and MDI - use the **-mdi** command switch for MDI operation.

### 3.2. Dialog boxes

A *dialog box* is like a panel, with an implicit frame surrounding it. A dialog box may be *modal* (no other window in this application is active and the calling program flow is suspended) or *modeless* (any window may be interacted with and control returns immediately to the program). With dialog boxes, creation of separate frames and panels is not necessary, and under MS Windows, additional functionality is added 'for free', such as tabbing between items. Any panel item may be attached to a dialog box since **wxDialogBox** is derived from the **wxPanel**.

Note that under MS Windows, modal dialogs have to be emulated using modeless dialogs and a message loop. This is because MS Windows expects the contents of a modal dialog to be loaded from a resource file or created on receiving a dialog initialization message. This is too restrictive for wxWindows, where any window may be created and possibly displayed before its contents are created.

Standard dialogs are provided for printer settings, file selection, short messages, single-line text string entry and scrolling single-selection lists.

### 3.3. Menus

Menus are used in menu bars and popup menus. A menu bar is a sequence of pull-down command menus near the top of the window, usually with a **File** menu as the first menu. In MS Windows and Motif, the menu bar is a standard user interface component. Under XView,

wxWindows must simulate a menu bar with a series of menu buttons. Under wxWindows for Motif and MS Windows, but not XView, placing an ampersand before a letter in a menu name causes it to be underscored and interpreted as a keyboard shortcut. Under XView such underscores are ignored. The Motif version of wxWindows automatically right-justifies the help menu, if there is one.

Menu items are identified by integer identifiers, and for menu bars, when a menu item is selected, the parent frame is notified using the **OnMenuCommand** member. For popup menus, a callback function is executed.

### 3.4. Events

In XView and Motif, events (such as resizing, painting, mouse clicks, button presses) are handled by a rather arbitrary collection of optional XView and Xlib callbacks. In MS Windows, events are *messages* which are handled by a window procedure, or ignored.

In wxWindows, an application's GUI objects mostly receive events by wxWindows calling user-overridable handlers, such as **OnEvent**, **OnChar** and **OnSize**. Frames can receive **OnClose** events from the window manager (X) or system control menu (MS Windows). These events are handled by the application by deriving new frame classes and overriding the event-handling member functions.

Panel item notification has to be handled rather differently. In MS Windows, all panel item events (such as a button press) are sent to the parent window (requiring large case statements to differentiate the events), and all window events are sent to the relevant window. In XView and Motif, different types of callback function have to be defined, depending on the event.

In wxWindows, panel items have optional callback functions, but there is only one callback type: the function takes the object and a command event structure. The class derivation approach cannot be extended to panel items since it does not make sense to derive a new class for every individual panel item created.

From wxWindows 1.61, panel items that do not have callbacks defined for them use their parent panel (or dialog box) to notify events, via the **OnCommand** virtual member function. This is essential when loading dialogs and panels from resources, since callback functions cannot be specified in a resource file. So in such cases, the programmer must derive a new panel or dialog class in order to override this member function and intercept events. The panel item calling the function can be identified by giving each panel item a unique name, and using **wxWindow::GetName** inside the **OnCommand** function.

From wxWindows 1.62, all window classes derive from **wxEvtHandler**, and a window's event handler, which defaults to point to itself, can be changed without the need for deriving a new class. This decoupling of event handling code from the window object itself makes it possible to override behaviour, perhaps temporarily; example applications might be dialog editors and on-line tutorial systems that need to intercept user input. It also gives another method of handling panel item input, by setting an event handler object for the panel item or panel item's parent.

#### 3.4.1. The wxWindows event system

From Version 1.5, wxWindows has its own event system. Event classes are derived from the abstract base class **wxEvent** (which used to serve as a holder for canvas and panel item events). An event *class* encapsulates a range of similar event *types*. For example, the instances of the **wxMouseEvent** class may be generated for a number of event types such as **wxEVENT\_TYPE\_LEFT\_DOWN**.

There are several motivations for having an explicit representation for events.

Firstly, encapsulating events in structures makes sense compared with passing an event handler a large number of arguments.

Secondly, user input events may be simulated by sending events; this can be used for automating GUI testing, and events could (in theory) be stored for later playback.

Thirdly, event handlers can be installed by programs or 'meta-programs' which want to override or examine program behaviour. For example, special-needs access code could be written to enable GUI events to be expressed to the user in a non-graphical form. Or, a meta-program could examine the dialog structure of a running application and output a skeleton help file corresponding to the application's hierarchy of menus and dialogs, taking some of the donkey work out of preparing on-line help.

Note that this event system has nothing to do with the decoupling of windowing and event-handling classes using `wxEvtHandler`. However, both systems can allow interception of user input, albeit by different mechanisms.

The event system is meant to be user-extensible so that any GUI-related or application-specific events may be defined and used in a consistent way. However, it is not yet completed and documented, and it is expected to evolve over the next few releases. The eventual aim is an event system which is comprehensive enough to enable `wxWindows` programs to be tested automatically using scripts (possibly using NASA's CLIPS as the scripting language -- see *CLIPS* (page 28)).

### 3.5. Keyboard input

From Version 1.5, `wxWindows` defines platform-independent constants for most common keys, including function and cursor keys. The virtual keycodes for standard ASCII characters are the ASCII codes themselves, unlike under Windows. Key presses are sent to the **OnChar** member of a window with a **wxKeyEvent** argument, which can be used to determine the state of the shift and control keys as well as finding the virtual key code of the depressed key.

The **wxCanvas** class has a default **OnChar** handler which scrolls the canvas using the cursor keys. From version 1.61, `wxPanel` receives `OnChar` events, and `wxTextWindow` receives `OnChar` events under Motif and Windows.

### 3.6. Repainting of windows

All windows except for the canvas are repainted by the system. The canvas requires a paint message handler to be defined (and therefore canvas derivation is obligatory). Under XView and Motif, `wxWindows` allows a canvas to be retained if desired, which means that fewer paint messages are received and scrolling is fast. The Motif implementation has a slight overhead in that drawing must be done both to the canvas and to the backing pixmap, but this is usually made up for by the speed of repainting.

The repaint procedure will obviously be written in such a way that the minimum amount of work needs to be done (for example, positions of objects on a canvas are only recalculated when the positions change).

See *Scrolling* (page 12) for more information on repainting and scrolling.

### 3.7. Scrolling

Scrolling can be a tricky subject for a novice GUI programmer to tackle. The MS Windows API ensures that the suffering is as acute as possible by requiring the application to program the scrollbar behaviour, to check the scrollbar positions on repainting, and to reposition the scroll bars on window resizing. XView has the decency to provide a canvas event with coordinates that reflect what the scrollbars are doing. In wxWindows, the XView approach is taken, so that all the programmer has to do is to create scrollbars with a given scroll length and increment, and the repaint procedure automatically reflects scrollbar positions. Obviously this simple approach can be inefficient if the canvas doesn't know what is actually in view, so the application can get the current view in order to limit the amount of repainting required.

It is possible to tell a wxWindows canvas to be *retained* (though this only has an effect in XView and Motif). In this case no repaint events will happen when scrolling since the system remembers what was drawn and simply moves a bitmap around.

Under MS Windows, wxWindows scrolls the bits of the canvas around when the user generates a scroll event, so the canvas does not need to be cleared and only the damaged areas need be repainted---so all the application need do is redraw the whole image, and scrolling will appear to be relatively smooth. This is the case in the `hello.exe` demo.

However, there are times when we can't allow the system to scroll the bits of the image, for example when a scroll increment will result in an unpredictable actual movement of the image. This is true for wxHelp, since scrolling is in terms of lines, and lines vary in height. If we left it up to wxWindows to scroll without clearing the screen, the text would then overlay some of the previous screen since the old and new images will probably not exactly match.

In the wxHelp application (see *wxHelp* (page 27)) the horizontal direction is scrolled in terms of pixels, not character widths, and so wxWindows can be left to scroll the image smoothly, without having to clear before repainting. This kind of control is available by using **wxCanvas::EnableScrolling**.

Scrolling panels are not implemented in wxWindows, and are not in general desirable in user interfaces.

From wxWindows 1.62, an **OnScroll** member may be overridden to allow an application deal directly with scroll events. The default wxCanvas::OnScroll member implements the current scrolling behaviour.

### 3.8. Printing

Printing in MS Windows is relatively easy, since all drawing is done to a 'device context' which could equally well be associated with a printer as with a window. MS Windows handles the plethora of printer types that abound in the PC environment. In X under UNIX, the standard is PostScript; X, XView and Motif provide no help at all. The solution adopted in wxWindow is to use a device context for canvases and printers, with MS Windows printing supported on the PC and an Encapsulated PostScript driver provided under X. Thus graphic code may be extremely generic - the same piece of code can draw to MS Windows screens of all types, to X windows, and to hundreds of different printers.

### 3.9. Programmatic versus interactive GUI building

Interactive tools for rapidly building GUIs are popular, and wxWindows has a simple screen painter and code generator called wxBuilder. wxBuilder can optionally generate wxWindows resource files (suffix `.WXR`) which helps to separate out detailed GUI specification from the code.

A different solution is taken by *GWX* (page 6) which translates DevGuide files into wxWindows code, which has the advantages and disadvantages on relying on a third party GUI-building tool. It is also possible to translate some Windows .RC files into wxBuilder menus and dialog boxes, although the quality of the translation varies from good (with most menu bar definitions) to poor (for most dialogs).

It is not always possible to build GUI components interactively: the 'what you see is *all* you get' syndrome. When complex repositioning of items depending on window size is required, then GUI builders cannot be relied upon.

Using a toolkit with geometry management may be no panacea, either; for example, the Motif constraint algorithm is difficult to understand and much experimentation is necessary to make things work. The approach taken by wxWindows is in keeping with the main goal of simplicity: wxWindows has the ability to create panel items from left to right, top to bottom with appropriate horizontal and vertical spacing; or the programmer may position the panel items explicitly. The first method gives resolution and font independence, and is less fiddly, and the second method may be used for tidying up a display for a specific platform.

From wxWindows 1.62, there is a facility for simple constraint-based window layout, using the **wxLayoutConstraints** class. This can be a more powerful alternative to left-to-right panel item positioning, and is a good method for laying out subwindows without the need to right a complex **OnSize** member function.

Recent improvements to wxWindows opens the prospect of wxWindows-based 'visual' editing tools: panels and dialog boxes can be switched into user-interface edit mode, enabling panel items to be dragged and sized, and panel mouse clicks intercepted. Since this functionality is built in, it becomes relatively straightforward to construct dialog editors in any language linked to wxWindows.

### 3.10. Dimensions

The graphics origin is always the top left hand corner of a window. Dimensions are a problem in a multi-platform application, since display and character widths will change from machine to machine, even more so than for different PC display boards. At the moment wxWindows uses pixels; MS Windows tackles the problem by using 'dialog units' based on the size of the standard system font. To avoid this problem when creating panel items, wxWindows provides automatic left to right, top to bottom item layout (similar to XView), in addition to absolute positioning, in which case, portability is up to the discretion of the programmer.

A canvas has a *mapping mode* associated with it, which determines the meaning of dimensions in subsequent graphics operations. Drawing may be done using various units including mm, 1/10 mm, pixels and points. Mapping modes other than pixels cannot be relied upon, however; future versions of wxWindows may support mapping modes better, and allow a change of graphics origin.

### 3.11. Colour

The presence of a monochrome screen can be detected so the application can change its use of colour accordingly. Currently, wxWindows tries to choose appropriate pen and brush colours for a monochrome display. To override this behaviour, set the device context **Colour** member to TRUE and choose custom colours for drawing graphics.

The *Colours* (page 30) utility can be useful for displaying colours and their names.

Colourmaps (or palettes) are supported only for colourmaps that have been created by platform-

specific utilities such as the DIB and wxImage utilities. So you can associate a colourmap with a window or device context, but functions to create and manipulate colourmaps are lacking. Future versions of wxWindows should improve upon colourmap handling.

### 3.12. On-line help

Most modern GUI applications have on-line hypertext help. In MS Windows, help is normally supplied in binary files which are read by an external program, which is itself accessed from within the application using Microsoft-supplied function calls. XView has provisions for simple context sensitive help, but no equivalent of the MS Windows browseable help.

From version 1.30, wxWindows comes with wxHelp, a hypertext help system which may be invoked from wxWindows applications. However, a wxWindows program may specify that under MS Windows, WinHelp should be used instead of wxHelp. wxHelp may be seen as a 'last resort' if the GUI system does not have a native help facility.

Please see *wxHelp* (page 27) and the separate wxHelp manual for further details.

wxWindows programmers may wish to follow the example of the wxWindows documentation, and prepare their manuals in a LaTeX form. The supplied *Tex2RTF* (page 29) utility may then be used to generate online help in a variety of paper and on-line hypertext formats, including wxHelp and Windows Help.

### 3.13. User preferences

In both X and Windows, there are mechanisms for handling user preferences, or resources. Under X, there is a global .Xdefaults file plus individual application defaults files. Under Windows, either win.ini or application specific .ini files are used.

wxWindows unifies these with **wxGetResource** and **wxWriteResource** functions: section, entry, value and file arguments may be specified.

In X, a section is the first word in a resource specification; in Windows, this is enclosed in square brackets on a line of its own. In X, an entry is taken to mean the rest of a resource specification, and in Windows this is a name followed by an equals character. The value is an arbitrary string in both cases, although wxWindows overloads the resource functions for commonly used types.

If the resource filename is omitted, the main resource file is assumed (.Xdefaults or win.ini). Under X, if an application class (wxApp::wx\_class) 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.

## 3.14. Interprocess Communication

### 3.14.1. What wxWindows has

Interprocess communication (IPC) has always been a tricky area, and the plethora of techniques on different platforms has not helped. Microsoft has laid down several standards for IPC under MS Windows, the most established (if long in the tooth) being Dynamic Data Exchange (DDE). DDE is the basis for wxWindows's IPC capability: the same, simple, object-oriented interface is provided for a subset of DDE under both Windows on the PC, and under XView and Motif on UNIX. The UNIX version is implemented using sockets, and allows processes on the same or different machines to talk to each other.



The benefits of wxWindows's DDE package are twofold: much greater simplicity compared with raw DDE and UNIX sockets; and the considerable advantage of keeping to platform-independence even in this notoriously platform-dependent area. Currently only synchronous transactions are handled; a later version of wxWindows may support asynchronous transactions also.

A user-contributed package is available that replaces the wxWindows implementation of DDE with a WinSock implementation, to allow communication between Windows and UNIX. The API of that package is identical to the one described in this manual. The WinSock implementation will probably soon be incorporated into wxWindows.

See the *overview of DDE* (page 378) for more detailed information.

## 4. 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.

### 4.1. Code structure

Since version 1.50, the files have been split as much as possible into command and platform-specific code. The `include` and `src` directories each have a `base` directory for the common code, plus a directory for each platform, currently `msw` and `x`. Because much X functionality is shared, the Motif and XView versions are contained in the same files, separated by conditional compilation statements. This structure should make it easy both to implement support for new platforms, and to edit the existing code.

In order to provide separate out the base functionality from the platform-specific functionality, each wxWindows class is composed of two classes: a base for the common code with prefix **wxb**, and a derived class for the platform with prefix **wx**. Base files have a **wb\_** prefix whilst platform-specific files (or files which *never* have corresponding platform-specific code) have a **wx\_** prefix.

For example, the **wxCanvas** class is declared for each platform, and derives from **wxbCanvas**. The relevant files for this class may therefore be found as follows:

```
include\base\wb_canvas.h
include\msw\wx_canvas.h
include\x\wx_canvas.h
src\base\wb_canvas.cpp
src\msw\wx_canvas.cpp
src\x\wx_canvas.cpp
```

Each **wx\_** header file includes the corresponding **wb\_** header file.

### 4.2. Include files

The main include file is `"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_prec.h"

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP
... include minimum set of files necessary here ...
#endif

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

The file `"wx_prec.h"` includes `"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\_prec.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.

### 4.3. Libraries

Under UNIX, use the library libwx\_ol.a (XView) or libwx\_motif.a (Motif). Under Windows, use the library wx.lib for stand-alone Windows applications, or wxdll.lib for creating DLLs.

### 4.4. Configuration

The following lists the options configurable in the file `include/base/wx_setup.h`. 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.

#### 4.4.1. General features

**ENHANCED\_FONTS** Define to be 1 to have pre-defined fonts in wxEnhDialogBox.  
**USE\_BUTTONBAR** If 1, the wxButtonBar class is compiled.  
**USE\_CLIPBOARD** If 1, clipboard code is compiled (Windows only).  
**USE\_CONSTRAINTS** If 1, the constraint-based window layout system is compiled.  
**USE\_DOC\_VIEW\_ARCHITECTURE** If 1, wxDocument, wxView and related classes are compiled.  
**USE\_DRAG\_AND\_DROP** If 1, drag and drop code is compiled (Windows only).  
**USE\_DYNAMIC\_CLASSES** If 1, the run-time class macros and classes are compiled. Recommended, and necessary for the document/view framework.  
**USE\_ENHANCED\_DIALOG** If 1, wxEnhDialogBox code is compiled.  
**USE\_EXTENDED\_STATICS** If 1, wxStaticItem code is compiled for enhanced panel decorative items. Not rigorously tested, and not documented.  
**USE\_HELP** If 1, interface to help system is compiled.  
**USE\_FORM** If 1, wxForm code is compiled.  
**USE\_GAUGE** If 1, the wxGauge class compiled.  
**USE\_GLOBAL\_MEMORY\_OPERATORS** If 1, redefines global new and delete operators to be compatible with the extended arguments of the debugging wxObject new and delete operators. If this causes problems for your compiler, set to 0.  
**USE\_GNU\_WXSTRING** If 1, the enhanced GNU wxString and regular expression class are compiled in place of the normal wxString class. See contrib/wxstring for details.  
**USE\_IMAGE\_LOADING\_IN\_MSW** Use code in utils/dib to allow dynamic .BMP loading under MS Windows.  
**USE\_IMAGE\_LOADING\_IN\_X** Use code in utils/image to allow dynamic .BMP/.GIF loading under X.  
**USE\_RESOURCE\_LOADING\_IN\_MSW** Use code in utils/rcparser to allow dynamic .ICO/.CUR loading under MS Windows.  
**USE\_IPC** If 1, interprocess communication code is compiled.  
**USE\_MEMORY\_TRACING** If 1, enables debugging versions of wxObject::new and wxObject::delete if the value of DEBUG is defined to more than 0.  
**USE\_METAFILE** If 1, Windows Metafile code is compiled.  
**USE\_PANEL\_IN\_PANEL** If 1, experimental panel-in-panel code is used for common dialog boxes. Not recommended, since tab traversal can suffer.

**USE\_POSTSCRIPT** If 1, PostScript code is compiled.

**USE\_POSTSCRIPT\_ARCHITECTURE\_IN\_MSW** Set to 1 to enable the printing architecture to make use of either native Windows printing facilities, or the wxPostScriptDC class depending on the wxApp::SetPrintMode setting.

**USE\_PRINTING\_ARCHITECTURE** If 1, wxPrinter, wxPrintout and related classes are compiled for the print/preview framework.

**USE\_RESOURCES** If 1, win.ini or .Xdefaults-style resource read/write code is compiled.

**USE\_SCROLLBAR** If 1, wxScrollBar class is compiled. Not rigorously tested, and not documented.

**USE\_SPLINES** If 1, spline code is compiled.

**USE\_TOOLBAR** If 1, the wxToolBar class is compiled.

**USE\_TYPEDEFS** If 1, a typedef will be used for wxPoint instead of a class declaration, to reduce overhead and avoid a Microsoft C++ memory bug.

**USE\_VLBOX** If 1, wxVirtListBox code is compiled for a virtual listbox item. Not rigorously tested, and not documented.

**USE\_WX\_RESOURCES** If 1, wxWindows resource file (.WXR) code is compiled.

**USE\_XFIG\_SPLINE\_CODE** If 1, XFig-derived code is used for spline drawing. If 0, AIAl code is used, which is slower.

**USE\_XPM\_IN\_X** If 1, XPM (colour pixmap) facilities will be compiled and used in wxBitmap under X.

**USE\_XPM\_IN\_MSW** If 1, XPM (colour pixmap) facilities will be compiled and used in wxBitmap under MS Windows.

**WXGARBAGE\_COLLECTION\_ON** If 1, wxWindows is made compatible with a experimental garbage collector (needs MrEd distribution). Not recommended for general use.

#### 4.4.2. X features

**DEFAULT\_FILE\_SELECTOR\_SIZE** Let Motif choose the size of XmFileSelectionBox. Otherwise, size is 500x600.

**PIXELO\_DISABLE** Define to disallow allocation of pixel 0 (wxXOR problem).

**USE\_GADGETS** Use gadgets where possible rather than Widgets for items. Default is to use Gadgets.

**USE\_BUTTON\_GADGET** Use gadgets for buttons. This can interfere with default button selection, so the default is zero.

**USE\_NOTICES** Under XView, use Notice package where possible instead of normal dialog boxes.

**wxFSB\_WIDTH** Width of file selector box, if fixed.

**wxFSB\_HEIGHT** Height of file selector box, if fixed.

#### 4.4.3. Windows and NT features

**CTL3D** It is recommended that CTL3D is used under Windows, since the 3D effects are good-looking and will be standard with Windows 4.0. If you want to use it and don't already have CTL3D installed, copy the files in contrib/ctl3d to appropriate places (ctl3dv2.lib/ctl3d32.lib into your compiler lib directory, ctl3d.h into an include directory, and ctl3dv2.dll into windows/system). You may need to find a compiler-specific version of ctl3dv2.lib or ctl3d32.lib. Define CTL3D to be 1 in wx\_setup.h and link your executables with ctl3dv2.lib or ctl3d32.lib.

If both CTL3D and FAFA are set to 1, then all controls except wxButton will use CTL3D and have 3D appearances. wxButton will have the ability to use bitmaps.

**FAFA\_LIB** This is the recommended configuration.  
Define this to be 1 if you wish to use the Fafa enhanced control library (in the contrib directory). The Fafa library is mandatory for use of bitmap buttons.

An application using the Fafa library must include `fafa.rc` in the its RC file. Otherwise, some controls will not show up.

Windows 95 update: dialogs can be marked with the Win95 3D look by specifying the `DS_3DLOOK`. But this doesn't apply to panels. Although `WS_EX_CLIENTEDGE` could be used to provide 3D controls without the need for CTL3D, the required changes have so far only been applied to wxWindows 2.0. So for now, CTL3D is still required for Windows 95 applications.

**EDITABLE\_TEXT\_WINDOW** If 1, allow ONLY an editable wxTextWindow and compile out the large-file support. That is, always use the standard EDIT control. Defaults to 0.

**USE\_COMMON\_DIALOGS** If 0, disables common dialogs. Defaults to 1; would rarely be changed.

**USE\_GREY\_BACKGROUND** If 1, will use grey for panel and dialog backgrounds (Julian Smart's preferred setting). Grey is used by CTL3D anyway. If this is set to 0 and `USE_FAFA` is set to 1, you may see unsatisfactory display of some control backgrounds.

If both CTL3D and FAFA are set to 1, then all controls except wxButton will use CTL3D and have 3D appearances. wxButton will have the ability to use bitmaps. This is the recommended configuration.

**USE\_KEYBOARD\_HOOK** If 1, sends OnCharHook message to wxApp and active wxWindow classes.

**USE\_ITSY\_BITS** If 1, compiles in code to support tiny window titlebars.

**USE\_BITMAP\_MESSAGE** If 1, compiles bitmap support for wxMessage using the FAFA library.

**USE\_ODBC** If 1, compiles wxDatabase and wxRecordSet classes for ODBC access. Requires `sql.h`, `sqlext.h` files if set to 1 (see topic on database support).

## 4.5. 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.

Sample makefiles for UNIX (suffix `.UNX`), MS C++ (suffix `.DOS` and `.NT`), Borland C++ (`.BCC`) and Symantec C++ (`.SC`) are included for the library, demos and utilities. The NT, Borland and Symantec makefiles cannot be guaranteed to be up-to-date since the author does not have these compilers.

The controlling makefile for wxWindows is in the platform-specific directory, such as `src/msw` or `src/x`. This makefile will recursively execute the makefile in `src/base`.

### 4.5.1. Windows makefiles

For Microsoft C++, normally it is only necessary to type `nmake -f makefile.dos` (or an alias or batch file which does this). By default, binaries are made with debugging information, and no optimization. Use `FINAL=1` on the command line to remove debugging information (this only really necessary at the link stage), and `DLL=1` to make a DLL version of the library, if building a library.

### 4.5.2. UNIX makefiles

All makefiles have the targets *xview*, *motif* and *hp*.

*xview* builds an Sun Open Look version of the library, *motif* builds a Sun Motif version, and *hp* builds a Motif version for HP workstations.

Remove object files, executables and libraries for the current module with the *clean\_ol* and *clean\_motif* and *clean\_hp* targets.

It is possible to maintain simultaneous Motif and Open Look versions of an application, since object files are kept in separate directories (*objects\_ol* and *objects\_motif*).

Debugging information is included by default; you may add `DEBUG=` as an argument to make to compile without it, or use the UNIX **strip** command to remove debugging information from an executable.

*Important note:* Most compiler flags are kept centrally in `src/make.env`, which is included by all other makefiles. This is the file to edit to tailor wxWindows compilation to your environment.

## 4.6. Windows-specific files

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

### 4.6.1. Resource file

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

```
rcinclude 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.

### 4.6.2. Module definition file

A module definition file (extension DEF) 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.

## 4.7. Memory models and memory allocation

Under UNIX, memory allocation isn't a problem. Under Windows, the only really viable way to go is to use the large model, which uses the global heap instead of the local heap for memory allocation. Unless more than one read-write data segment is used (see *large data* (page 25) below), large model programs may still have multiple instances under MS C/C++ 7. Microsoft give the following guidelines for producing multiple-instance large model programs:

- Do not use /ND to name extra data segments unless the segment is READONLY.
- Use the .DEF file to mark extra data segments READONLY.
- Do not use \_\_far or FAR to mark data items.
- Use /PACKDATA to combine data segments.
- Use /Gt65500 /Gx to force all data into the default data segment.

Even with the single-instance limitation, the productivity benefit is worth it in the majority of cases. Note that some other multi-platform class libraries also have this restriction. (If more than one instance really is required, create several copies of the program with different names.)

Having chosen the large model, just use C++ 'new', 'delete' (and if necessary 'malloc' and 'free') in the normal way. The only restrictions now encountered are a maximum of 64 KB for a single program segment and for a single data item, unless huge model is selected.

For Borland users, use the data threshold switch, and the following is also recommended:

- Check "Automatic Far Data Segments"
- Check "Put Constant Strings into Code Segment"

See also the Frequently Asked Questions document for further details on using Borland with wxWindows.

### 4.7.1. 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.

Don't statically create a window unless you know that the window cannot be deleted dynamically. Modal dialogs, such as those used in the `dialogs` sample, can usually be created statically, if you know that the OK or Cancel button does not destroy the dialog.

Most drawing objects, such as wxPen, wxBrush, wxFont, and wxBitmap, should be created dynamically. They are cleaned up automatically on program exit. wxColourMap is an exception to this rule (currently). In particular, do not attempt to create these objects globally before OnInit() has a chance to be called, because wxWindows might not have done essential internal initialisation (including creation of lists containing all instances of wxPen, wxBrush etc.)

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(NULL) or wxDC::SelectObject(NULL) 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.

## 4.8. Dynamic Link Libraries

wxWindows may be used to produce DLLs which run under MS Windows. Note that this is not the same thing as having wxWindows as a DLL, which is not currently possible. For Microsoft C++, use the makefile with the argument DLL=1 to produce a version of the wxWindows library which may be used in a DLL application. There is a bug in Microsoft C++ which makes the compiler complain about returned floats, which goes away when the /Os option is used, which is why that flag is set in the makefile.

For making wxWindows as a Sun dynamic library, there are comments in the UNIX makefile for the appropriate flags for AT&T C++. Sorry, I haven't investigated the flags needed for other compilers.

## 4.9. 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 following identifiers may be used for this purpose, along with any user-supplied ones:

- wx\_x - for code which should work under any X toolkit
- wx\_xview - for code which should work under XView only
- wx\_motif - for code which should work under Motif only
- wx\_msw - for code which should work under Microsoft Windows only

For example:

```
...
#ifdef wx_x
    (void)wxMessageBox("Sorry, metafiles not available under X.");
#endif
#ifdef wx_msw
    wxMetaFileDC dc;
    DrawIt(dc);
    wxMetaFile *mf = dc.Close();
    mf->SetClipboard();
    delete mf;
#endif
...
```

## 4.10. Building on-line help



wxWindows has its own help system from version 1.30: wxHelp. It can be used to view the wxWindows class library reference, and also to provide on-line help for your wxWindows applications. The API, made accessible by including `wx_help.h`, allows you to load files and display specific sections, using DDE to communicate between the application and wxHelp.

wxHelp files can be marked up by hand from ASCII files within wxHelp, or may be generated from other files, as is the case with the wxWindows documentation.

From version 1.50, it is possible to use the platform-specific help system (e.g. WinHelp) instead of wxHelp.

See `install.txt`, the wxHelp documentation (in `utils/wxhelp/docs`) and *wxHelp* (page 27) for further details.

## 4.11. C++ issues

There are cases where a C++ program will compile and run fine under one environment, and then fail to compile using a different compiler. Some caveats are given below, from experience with the GNU C++ compiler (GCC) and MS C/C++ compiler version 7.

### 4.11.1. Templates

wxWindows does not use templates for two main reasons: one, it is a notoriously unportable feature, and two, the author is irrationally suspicious of them and prefers to use casts. More compilers are now implementing templates, and so it will probably be safe to use them soon without fear of portability problems.

### 4.11.2. Definition of constructors

Some compilers allows the user to omit constructor definitions where a parent class provides a constructor with parameters. In Microsoft C++, all constructors with parameters must be defined in the derived class, or the compiler cannot find the required constructor. This may mean defining dummy constructors which call parent constructors, for example:

```
MyClass::MyClass(int x, int y):ParentClass(x, y)
{
}
```

This is not a problem where the constructor has no parameters.

### 4.11.3. Pointers to functions

Some compilers are clever in their matching of function pointer arguments to the declaration of the function, and will not complain in the following case:

```
typedef void (*wxFunction) (wxObject&, wxCommandEvent&);
...
void mycallback(wxButton& button, wxCommandEvent& event);
...
wxButton button(parent, &mycallback, label, 100, 200);
```

Since `wxButton` is derived from `wxObject`, the function *mycallback* is a subtype of **`wxFunction`**. In Microsoft C++, and most compilers with a high warning level set, the function is not an exact match, and the compiler complains. The solution is to place a cast in front of the function address, thus:

```
wxButton button(parent, (wxFunction)&mycallback, label, 100, 200);
```

#### 4.11.4. 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++.

#### 4.12. 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 **`FileNameFromPath`** 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.

#### 4.13. Large amounts of global data

Under Windows, it is possible that the default data segment becomes too large (for example, a

large number of small, global data items have been declared). This may be cured by using more than one data segment. In Microsoft C++, specify the `/Gt` compiler option with a number representing the data size threshold for putting data items in a separate segment. For example, `/Gt8`.

The tradeoff is that using more than one data segment prevents you from having more than one instance of the program running at a time (see *Memory Models* (page 22)). Large model programs with one data segment may still have multiple instances.

A separate problem sometimes occurs when the linker complains about too many segments. This can be cured by using the `/SEG` linker switch, for example `/SEG:256`.

For optimum Borland compilation, please the Frequently Asked Questions guide ([docs/faq.txt](#), [docs/faq.ps](#) and via the wxWindows WWW home page).

## 5. Utilities supplied with wxWindows

A number of 'extras' are supplied with wxWindows, to complement the GUI functionality in the main class library. These are found below the `utils` directory and usually have their own source, library and documentation directories. For larger user-contributed packages, see the directory `/pub/wxwin/contrib` and *Resource Guide* (page 6).

### 5.1. wxBuilder

wxBuilder is a start at an interactive GUI builder for wxWindows applications. It allows the developer to quickly construct a fair amount of the skeleton of a GUI program, before filling in the details by hand. Presently wxBuilder runs best under Windows, but it will run under Motif as well. The XView version is pending but there are problems with modal dialogs that need to be ironed out.

wxBuilder is so far the largest freely-available program written in wxWindows, totalling about 16,000 lines, and so may be of interest as a sample application of reasonable complexity. However, I don't make any claims about the C++ style or user interface aesthetics.

Please see the printed and on-line documentation for wxBuilder.

### 5.2. wxToolBar

wxToolBar implements a simple toolbar class, to help give wxWindows applications a more graphical and intuitive look and feel. A wxToolBar is a canvas with bitmaps either arranged automatically in rows and columns, or spaced explicitly by the application. Individual tools can be toggle or non-toggle, and toggle tools may have a second bitmap to denote the on-state. Without a second bitmap, a distinguishable on-state is supplied by wxToolBar. To supply further feedback for the user, the application may override a member to intercept mouse movement over the tools, typically using this to supply an explanatory string on the status line.

A toolbar can be made to exist in its own frame, or (for example) below the menu bar of the main application window. Both these types are shown in the demo toolbar application `samples/toolbar/test.cpp`. wxToolBar works under Motif, XView, Windows and NT.

New in wxWindows 1.61 is an adjunct to wxToolBar called **wxButtonBar**. This is almost identical in use to wxToolBar, but is optimized to look and behave in a more sensible manner under Windows, with 3D buttons that depress properly. Under X, behaviour is the same as for wxToolBar.

Please refer to the class reference for further details.

### 5.3. wxHelp

wxHelp is a stand-alone program, written using wxWindows and *hyText* (page 28), for displaying hypertext help. It is necessary since not all target systems (notably X in the guise of XView and Motif) supply an adequate standard for on-line help. wxHelp is modelled on the MS Windows help system, with contents, search and browse buttons, but does not reformat text to suit the size of window, as WinHelp does, and its input files are uncompressed ASCII with some embedded font commands and an `.xlp` extension. Most wxWindows documentation (user manuals and class references) is supplied in wxHelp format, and also in Windows Help format.

Note that an application can be programmed to use Windows Help under MS Windows, and wxHelp under X. An alternative help viewer under X is Mosaic, a World Wide Web viewer that uses HTML as its native hypertext format. However, this is not currently integrated with

wxWindows applications.

wxHelp works in two modes---edit and end-user. In edit mode, an ASCII file may be marked up with different fonts and colours, and divided into sections. In end-user mode, no editing is possible, and the user browses principally by clicking on highlighted blocks.

When an application invokes wxHelp, subsequent sections, blocks or files may be viewed using the same instance of wxHelp since the two programs are linked using wxWindows interprocess communication facilities. When the application exits, that application's instance of wxHelp may be made to exit also. See the **wxHelpInstance** entry in the reference section for how an application controls wxHelp.

## 5.4. hyText

This is the hypertext library used by wxHelp to show text with mixed fonts and colour, and manipulate blocks of text. It is supplied in library form so it can be used by other applications (it is in fact used by another application at AIAI).

See the separate manual and class reference for hyText.

## 5.5. wxCLIPS

C++ is all very well for applications that have to be fast, deliverable and, above all, written in C++. But what about casual programmers, prototypers, and those who want something a little more high-level?

Naturally, wxWindows, in collaboration with NASA, can supply the answer---wxCLIPS. CLIPS is NASA's expert system shell consisting of a LISP-like functional language, a rule interpreter, an object system, and the crucial characteristic of portability. It's implemented as a C library, can be embedded in any C or C++ application with no royalty payments, and is free within the United States, available at low cost elsewhere.

As a language, CLIPS has several advantages over C++. It requires no explicit memory management, since it has garbage collection; it's interpreted, so no long compilations; and it has a rule interpreter, useful for knowledge based system projects, or just occasional pattern matching and searching. So it's useful for prototyping applications, or for non-C++ programmers. From the C++ developer's point of view, it makes a great embedded language for user-extensible applications.

wxCLIPS adds a library of GUI CLIPS functions to CLIPS; the intention is to cover all of wxWindows functionality eventually, though presently, only the most important functionality is covered. wxCLIPS comes as both a library and an executable, so it can be used straightaway to develop GUI programs, and it can also be linked into applications to provide a built-in language. Within AIAI, wxCLIPS has proven popular amongst students and professionals for rapid-prototyping and customization.

A program called CLIPS2C has been written to translate a subset of CLIPS into C++, to combine the advantages of interactive development with efficient delivery. CLIPS2C knows about the wxCLIPS extensions.

See the wxCLIPS manual and NASA's CLIPS manuals for further details.

wxCLIPS and CLIPS2C are now distributed separately, to save space in the wxWindows distribution. They are now available from the /pub/wxclips directory of AIAI's ftp site.

## 5.6. PrologIO

Much of a programmer's time can be spent in writing and modifying code to load and save data files. PrologIO is a utility that makes life easier, if your application's data needs look something like the following:

- The data are complex, possibly involving linked records and nested lists.
- The application is under development, so the size and contents of each record are liable to change.
- Backward compatibility of data files is required.
- Ability to read and edit data file is required.
- Speed of application development is important, and speed of data I/O is a lesser priority.

A PrologIO data file is an ASCII series of 'objects'. In fact, each object is a subset of the syntax for a term in the Prolog language, and a PrologIO file can be read into Prolog with a single command, but this is irrelevant for most purposes. Because these objects consist of a list of attribute-value pairs, parsed by a YACC/LEX grammar, attributes can be removed from or added to an object without 'breaking' the data format. A typical application will be written to deal with missing attribute values, supplying defaults instead, and so data I/O can be very robust, even though the application changes substantially over the months and years. Very old data files are likely to be useable, even though many bells and whistles have been added to the application.

Since the file is ASCII, and readable, it is possible to edit the data file directly if something goes wrong---a very useful fallback position. Also, one application's data file can be easily read by another's, encouraging the separation of a complex application into a suite of smaller tools.

PrologIO supplies a number of classes to manipulate these objects and whole databases of objects. A PrologIO database can be built up in memory and then dumped to a file with a single statement, and conversely, a data file can be loaded into memory with a single statement, and then picked apart.

Please refer to the separate PrologIO manual. *wxBUILDER* (page 27) is a good example of a program that makes extensive use of PrologIO, and the wxWindows resource system also relies on it.

## 5.7. Tex2RTF

Supplied with wxWindows is a utility called Tex2RTF for converting LaTeX manuals to the following formats:

**wxHelp** wxWindows help system format (XLP).

**Linear RTF** Rich Text Format suitable for importing into a word processor.

**Windows Help RTF** Rich Text Format suitable for compiling into a WinHelp HLP file with the help compiler.

**HTML** HTML is the native format for Mosaic, the main hypertext viewer for the World Wide Web. Since it is freely available it is a good candidate for being the wxWindows help system under X, as an alternative to wxHelp.

Tex2RTF is used for the wxWindows manuals and can be used independently by authors wishing to create on-line and printed manuals from the same LaTeX source. Please see the separate documentation for Tex2RTF.

### **5.8. wxTreeLayout**

This is a simple class library for drawing trees in a reasonably pretty fashion. It provides only minimal default drawing capabilities, since the algorithm is meant to be used for implementing custom tree-based tools.

Directed graphs may also be drawn using this library, if cycles are removed before the nodes and arcs are passed to the algorithm.

Tree displays are used in many applications: directory browsers, hypertext systems, class browsers, and decision trees are a few possibilities.

See the separate manual and the directory `utils/wxtree`.

### **5.9. wxGraphLayout**

The `wxGraphLayout` class is based on a tool called 'graphplace' by Dr. Jos T.J. van Eijndhoven of Eindhoven University of Technology. Given a (possibly cyclic) directed graph, it does its best to lay out the nodes in a sensible manner. There are many applications (such as diagramming) where it is required to display a graph with no human intervention. Even if manual repositioning is later required, this algorithm can make a good first attempt.

See the separate manual and the directory `utils/wxgraph`.

### **5.10. wxImage**

This is a collection of GIF/BMP/XBM bitmap loading and displaying routines for X, which may be used in conjunction with the Windows-only *DIB* (page 30) library for multi-platform bitmap display.

### **5.11. DIB**

A Windows-only BMP loading and displaying library (see also *wxImage* (page 30) for an equivalent for the X platform).

### **5.12. rcParser**

This library, written by Petr Smilauer, parses Windows resource files, bitmaps, icons and cursors. The library is used in `wxBuilder`, but other applications may find uses for it. Sorry, there isn't much documentation, but the header files and `wxBuilder` sources should supply a few clues.

### **5.13. MFUTILS**

A very modest step towards reading Windows metafiles on the any platform. The `ClockWorks` program, available from AIAI, demonstrates how extremely simple metafiles may be read and displayed (in this case, to be used as clock hands).

### **5.14. Colours**

A colour sampler for viewing colours and their names on each platform.

## 6. Bugs and future directions

### 6.1. Bugs

These are the known bugs (the contents of bugs.txt) plus a to-do list.

wxWindows Bug and To Do List

##### BUGS #####

19/4/93 Version 1.40 (First version for Motif)

- Haven't sorted out how to set default buttons without messing up tab traversal.
- Probably need to set scrollbars to force an initial paint, or call wxCanvas::SetSize. If you set the scrollbars, it works fine.
- No default colours. Works ok on monochrome though...
- Too many repaint messages sent sometimes.

7/9/93 Version 1.50

- Programmatic setting of multiple selections listboxes doesn't work (Motif 1.1).
- Canvas doesn't work under Motif 1.2.
- Colourmap still only producing a small number of colours - FIXED (was using too small values for RGB values)
- PostScript driver leaves something to be desired (espec. for Landscape mode).
- Windows 3.1 canvas scrolling problem with large scale factor or some mapping modes: leaves streaks behind. Could be a rounding error with SetViewport...?
- XView: closing of dialog boxes from the Window Manager not handled properly

16/11/93 Version 1.50 beta (h)

- Motif 1.2.1 support added, but this bug (at least) remains: setting panel size doesn't seem to return correct size if there are panel items. So a Fit on a panel then a surrounding frame doesn't work properly. WORKAROUND: make frame and panel very big before placing items.



-- Status line in Motif doesn't show separate regions for > 1 region.

-- No PostScript DrawArc, or documentation for it.

16/11/93 Version 1.50 beta (i)  
-----

-- Motif: listbox gets smaller when you add 1 or more items to it.

WORKAROUND: Set the listbox size after appending, e.g.

SetSize(-1, -1, 100 100)

-- Motif: Technicolour appears after a while on a canvas (HP only?)

-- Motif: XORing doesn't seem to work in colour

-- Panel-in-panel doesn't work for wxABSOLUTE\_POSITIONING panels  
(Motif) or XView (at all)

-- In Windows, CTL3D and Fafa library seem to conflict (some items  
don't

appear on a panel if both are in operation). SOLUTION: I probably  
didn't include the fafa.rc file. This should make it work...

17/1/94 Version 1.50 beta (j)  
-----

-- In beta (i), users found that buttons in Motif were random  
sizes. Something to do with using gadgets??

-- Cured bad MDI bug that crashed application (and Windows) on exit.

-- Cured XView wxPanel::Fit bug.

-- wxFrame::Fit() doesn't seem to work for an MDI frame that is  
iconized. It has to be displayed first, which leads to messy  
screen redraws. Ugh. Is this a Windows problem, or wxWindows?  
I suspect the former.

17/10/94 Version 1.60  
-----

-- PopupMenu doesn't always work under Motif, for some reason.  
The workaround is to use FakePopupMenu.

-- Dialog box destruction isn't always vetoed when it should be  
under XView (Motif?)

-- Frames don't always position properly (some interaction with  
window managers?)

-- wxGetFirst/NextFile not implemented for Borland or NT

-- wxHelp and hyText refreshing is BAD, especially under Motif  
when sometimes the text doesn't appear at all.

-- Under Motif, listboxes sometimes change size mysteriously.  
Cured (I think) in 1.62.

June 1995 version 1.62  
-----

-- OnSize events not sent in a consistent order across platforms.

-- When loading BMP or GIF into wxBitmap in X, depth not set.

-- wxbWindow::GetConstraints() (called by DeleteRelatedConstraints()  
and wxIndividualLayoutConstraint::ResetIfWin()) refer to windows  
that have been deleted. Perhaps windows are not always  
removed from the constraintsInvolvedIn list when deleted?  
CURED IN 1.65.

-- There still seems to be an interaction between GCC, wxWindows  
and Motif under HPUX, where the file selector causes a core  
dump. SEE FAQ FOR FIX.

-- wxCheckBox (and probably wxRadioBox) behave wrong under Motif:  
setting the value programmatically causes the callback to  
be invoked. This should be suppressed.

-- Under Windows, wxWin crashes when GDI resources run out  
(insufficient  
checks for GDI function failures).

-- wxDebugMsg doesn't seem to work under WIN32s if the format string is  
complex (gives bizarre characters). Workaround: pass pure strings  
with no formatting (use e.g. sprintf instead).

December 1995 1.65  
-----

-- Cannot persuade XView to refresh dialog boxes properly: in the  
Dialog Editor, handles are not redrawn when the dialog is refreshed,  
although they *are* drawn when not within a window-manager-  
originated  
OnPaint call.

-- For the above reason and others, Dialog Editor does not work for  
XView.

-- OnSet/KillFocus may not work for wxText in Motif.

-- XView (and possibly Motif) version of the wxChart library crashes the  
X server. Probably some incorrect values are being passed to drawing  
primitives, which work OK under Windows but not under X.

-- Panel item text and background colouring is better under Windows  
than it was, but  
not recently tested under Motif. Colouring doesn't work for wxButton  
under  
Windows for some reason, and I don't know how to alter the

foreground  
colour for panel items under Windows.  
There's confusion over what `wxPanel::SetBackgroundColour` and  
`wxItem::SetBackgroundColour` should do (should the former set the  
default for the latter? Not according to e.g.  
`wxButton::ChangeColor`).  
Similarly for `SetButtonColour`: button or font colour?

-- Programmatic scrollbars in Windows don't quite behave  
right: they don't always scroll to full extent.

-- Modal dialogs don't behave themselves well in XView, still  
(do XView bug fixes help?) E.g. `samples/dialogs.cc` shows  
choice items not working. So `wxColourDialog` and `wxFontDialog`  
don't work in XView.

-- `wxItem::OnEvent` returns slightly wrong mouse coordinates for  
Motif: must be normalized to take into account position of item  
widget on its form widget.

-- `wxResourceTable`, `wxItemResource`, `wxStaticItem`, `wxSizer` and  
derivatives not documented.

-- I recently realized that there was a bug in the way I calculate  
font point sizes under MS Windows. This results in fonts that  
are the wrong size, compared with point sizes in other applications  
and most annoyingly, the Windows font selector common dialog.  
Ironically, the bug allowed good font matching between X and  
Windows, but when the bug is corrected, fonts of a given size show  
up differently between X and Windows. So at present you can either  
have fonts that are consistent between platforms, or ones that  
are consistent with other Windows applications, but not both.  
Set `FONT_SIZE_COMPATIBILITY` to 1 for compatibility with previous  
version of `wxWindows`, to 0 for compatibility with other Windows apps  
(especially if you will be using the Windows font selector dialog).

-- `wxTextWindow::GetLastPosition` appears not to work in WIN32 (Watcom),  
although it does for WIN16.

January 1997 1.66F  
-----

-- Popup menu bug in Motif now fixed by Torsten Schmale.

## 6.2. Future directions

These are addressed in a separate `wxWindows` planning document, available from AIAI and on the `wxWindows` World Wide Web pages.

The plans include:

- document/view library;
- OLE-2 wrapper;
- Database connectivity;
- improved `wxBUILDER`;
- integration with an existing networking toolkit;

- multimedia widget (under construction);
- better makefiles and installation procedures;
- ports: those in progress include Mac, NeXT and OS/2.

As ever, the future of wxWindows is largely in the hands of its users, since no one person could cope with supporting all platforms simultaneously. So, please don't be backward in coming forward with project ideas and code contributions!

## 7. Tutorial

This short tutorial takes a look at some of the supplied demonstration programs. The tutorial is incomplete and may be expanded in later releases.

### 7.1. The demo programs

#### 7.1.1. A minimal wxWindows program

The best way to get a feel for how to use a tool is to see a small example. The supplied demo 'minimal' (source file `minimal.cpp`) shows a rudimentary wxWindows program. It has a main window with a panel inside it, displaying a message. There is a menu bar with a **File** menu which in turn has a **Quit** option, and the program has its own icon. Under MS Windows, the system menu shows the usual options including Minimize, Maximize and Close, and under X, there is a similar pull-down system menu provided by the current window manager. The window is resizable, and the panel automatically resizes to fit its parent.

Look at `minimal.cpp`.

```
/*
 * File:      minimal.cpp
 * Purpose:   Minimal wxWindows app
 * Author:    Julian Smart
 * Created:   1993
 * Updated:
 * Copyright:  (c) 1993, AIAI, University of Edinburgh
 */

/* static const char sccsid[] = "%W% %G%"; */

#ifdef __GNUG__
#pragma implementation
#pragma interface
#endif

// For compilers that support precompilation, includes "wx.h".
#include "wx_prec.h"

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP
#include "wx.h"
#endif

// Define a new application type
class MyApp: public wxApp
{ public:
    wxFrame *OnInit(void);
};

// Define a new frame type
class MyFrame: public wxFrame
{ public:
```

```
    MyFrame(wxFrame *frame, char *title, int x, int y, int w, int h);
    void OnMenuCommand(int id);
};

// ID for the menu quit command
#define MINIMAL_QUIT 1

// This statement initializes the whole application and calls OnInit
MyApp MyApp;

// A macro needed for some compilers (AIX) that need 'main' to be
defined
// in the application itself.
IMPLEMENT_WXWIN_MAIN

// `Main program' equivalent, creating windows and returning main app
frame
wxFrame *MyApp::OnInit(void)
{
    // Create the main frame window
    MyFrame *frame = new MyFrame(NULL, "Minimal wxWindows App", 50, 50,
400, 300);

    // Give it an icon
#ifdef wx_msw
    frame->SetIcon(new wxIcon("mondrian"));
#endif
#ifdef wx_x
    frame->SetIcon(new wxIcon("ai.ai.xbm"));
#endif

    // Make a menubar
    wxMenu *file_menu = new wxMenu;

    file_menu->Append(MINIMAL_QUIT, "Quit");
    wxMenuBar *menu_bar = new wxMenuBar;
    menu_bar->Append(file_menu, "File");
    frame->SetMenuBar(menu_bar);

    // Make a panel with a message
    wxPanel *panel = new wxPanel(frame, 0, 0, 400, 400);

    panel->SetLabelPosition(wxHORIZONTAL) ;
    wxMessage *msg = new wxMessage(panel, "Hello, this is a minimal
wxWindows program!", 5, 5);

    // Show the frame
    frame->Show(TRUE);

    // Return the main frame window
    return frame;
}

// My frame constructor
MyFrame::MyFrame(wxFrame *frame, char *title, int x, int y, int w, int
h):
    wxFrame(frame, title, x, y, w, h)
```

```
{}  
  
// Intercept menu commands  
void MyFrame::OnMenuCommand(int id)  
{  
    switch (id) {  
        case MINIMAL_QUIT:  
            delete this;  
            break;  
    }  
}
```

The statement `#include "wx.h"` provides the program with access to all the wxWindows classes and functions.

The first class declaration, **MyApp**, declares a new application, overriding one member function **OnInit**. This is an essential part of writing a wxWindows program, since **OnInit** is the equivalent of **main** in a normal C++ program.

The class **MyFrame** declares a constructor, and a message handler for intercepting menu commands.

The definition of the global variable **myApp** looks innocuous enough but this starts the whole application going simply by being defined.

The **OnInit** MyApp member function does the initialization of the program. It creates a main frame, sets the icon, creates a menu bar, a panel, and a panel item.

The **MyFrame** constructor may seem a little pointless, but it fulfils the requirements of C++ syntax in defining the constructor in terms of its parent's constructor.

The **OnMenuCommand** definition intercepts menu commands for the main frame. If the option chosen is **Quit**, the application terminates by deleting the main frame. Normally any other existing frames should be deleted (subframes are deleted automatically); these calls are usually put in the frame's **OnClose** handler so that a system-generated **OnClose** event will enable the program to clean itself up first. System-generated **OnClose** events delete the main frame after calling **OnClose**, so this should not be done from within **OnClose**.

### 7.1.2. More advanced features: the *hello* demo

The 'Hello wxWindows' demo (source files, `hello.cpp` and `hello.h`) shows off some more wxWindows features. When run, two windows pop up. One is the 'main window', with two subwindows - a panel containing various 'widgets', and a text window. The other contains a canvas, drawing some simple shapes, and allowing the user to doodle on it by dragging with the left mouse button. The canvas contents can be scaled and printed out, either to a printer supported by Windows or to PostScript, writing to a file or invoking the printer directly. Under Windows, the graphic may be copied to the clipboard as a metafile.

Both frames can be resized, and the subwindows will be resized in an appropriate manner. The text subwindow can be scrolled; on the panel, a button can be pressed for the program to prompt the user with text with which to set the status bar. Clicking on the list box writes a line of text into the text window.

The **File** menu has options for selecting the 'mapping mode' (logical dimensions) used in drawing graphics, a zoom option, and an option for loading a file into the text subwindow using a file

selector tool.

The **Timer** menu allows the user to switch a timer on and off; when on, some text gets written to the text subwindow every five seconds.

The **Cursor** menu enables the canvas cursor to be changed, and lets the potential wxWindows programmer view the available standard cursors.

The **About** option of the Help menu pops up a dialog box with some information.

This represents a fair amount of GUI functionality for a relatively small program. This is because wxWindows calls are high level (creating a working text window is a single call) and because an object-oriented approach is taken, where much default functionality is provided.

Note also the lack of explicit coordinates or sizes in the panel item creation calls. This is the preferred approach, leaving wxWindows to lay out the items from left to right and top to bottom, with the user interjecting the occasional **NewLine** call. Explicit positioning is not recommended since it is less device independent, but can be achieved by using more parameters to the creation calls, or by using **SetSize** after an item has been created. Coordinates and sizes default to -1, which tells wxWindows to choose appropriate positioning and sizing. In this example, the windows are explicitly sized, but you may size a frame or panel to fit around its contents by calling **Fit**.

**MyFrame's OnSize** member sizes each subwindow in proportion to the new size of the frame. **MyCanvas's OnPaint** draws a couple of lines, a rectangle and a spline whenever the canvas requires repainting (e.g. on creation, and when exposed). The **OnEvent** member checks for mouse dragging, and draws a line from the last point to the current position. Scrolling the canvas and subsequent repainting is handled automatically by wxWindows.

Finally, two callback functions demonstrate popping up dialog boxes, setting the status line, and inserting text into a text window.

This demo can provide a template for your own application. Gradually modify it for your own needs, and you will rapidly be writing portable X and MS Windows programs!

### 7.1.3. The MDI demo

As explained in the manual (see *MDI versus SDI* (page 9)), wxWindows takes an automated view of Microsoft Windows's MDI (Multiple Document Interface) since it is a very platform-specific feature. The special MDI 'Window' menu, allowing the user to switch between child windows, is provided automatically, though without any accelerators and without a choice of position. Also, instead of deriving frames from distinct classes for MDI versus SDI, the approach taken in Microsoft's class library, wxWindows uses an option in the frame constructor to switch between styles. This allows the programmer to delay the MDI/SDI decision, perhaps even providing a command-line switch to let the user decide.

The **mdi** example program shows this run-time switching. Invoked without a command line switch, it defaults to SDI. Invoked with the switch `-mdi` it runs as an MDI program, but only under MS Windows.

There are a few extra considerations when programming an MDI applications. One is the choice of menu items. An SDI program might have a main window and several child windows, where the main window menu has options for quitting the program and other global matters, while child window menus have child-specific options. In MDI, the child window menu visually replaces the main window menu when activated, and so it must duplicate some main window menu options.



One solution is to include logic to add extra menu items depending on whether MDI or SDI is specified, as the **mdi** example does.

Also, for programs which must be both SDI and MDI (on non-MS Windows platforms SDI mode is mandatory), the main window must not have subwindows (i.e. panels, canvases or text subwindows) since the client area may be occupied with child MDI windows under MS Windows.

#### 7.1.4. The IPC demo

The demo in the `directory samples/ipc` shows how processes may easily talk to each other synchronously (i.e. when A sends a message to B, A waits for an answer). If you start `server`, then `client`, a new window should appear on top of the server window, which represents the connection between server and client. Quitting the client causes the connection to be broken and this window to disappear.

To illustrate 'hot linking', click on the server's listbox. This sends an *advise* message to the client, telling it to update its own listbox. The reverse is not true, however.

A client may request information from the server. Select the **Request** menu item from the client's **File** menu. The window which pops up is created by the client and contains a message that the server sent back.

Selecting **Execute** from the client's menu makes the server pop up a window. Normally this would execute some command that the client wishes the server to run.

The **Poke** menu item sends a *poke* message to the server; normally this would insert some data into the server's memory.

Interprocess communication in wxWindows uses a subset of DDE (Dynamic Data Exchange) which is Microsoft's standard for low-level IPC under Windows. wxWindows gives you DDE under UNIX as well as Windows, and makes it easier to program into the bargain by using an intuitive object-oriented model of communication.

## **8. 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.

### **8.1. Strategies for reducing programming errors**

#### **8.1.1. 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.

#### **8.1.2. 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.

### **8.2. Strategies for portability**

#### **8.2.1. 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. If your needs are simple, the default relative positioning behaviour may be adequate (using default position values and wxPanel::NewLine).

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.

#### **8.2.2. 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).

### **8.3. Strategies for debugging**

#### **8.3.1. 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.

### **8.3.2. 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!

### **8.3.3. 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.

### **8.3.4. 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. Some platforms don't allow for debugging, such as WIN32s under Windows 3.x. In this case, you might be advised to debug under 16-bit Windows and when you're confident, compile for WIN32s. In fact WIN32s can be very strict about bad memory handling, so testing out under WIN32s is a good thing to do even if you're not going to distribute this version. (Unless you've got a good memory checking, utility, of course!) Tracking bugs under WIN32s can involve a lot of debug message insertion and relinking, so make sure your compiler has a fast linker (e.g. Watcom, Symantec).

### **8.3.5. Use tracing code**

You can use `wxDebugMsg` statements (or the `wxDebugStreamBuf` class) to output to a debugging window such as DBWIN under Windows, or standard error under X. If compiling in DEBUG mode, you can use TRACE statements that will be compiled out of the final build of your application.

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).

### 8.3.6. Use `wxObject::Dump` and the `wxDebugContext` class

It's good practice to implement the `Dump` member function for all classes derived from `wxObject`. You can then make use of `wxDebugContext` to dump out information on all objects in the program, if `DEBUG` is defined to be more than zero. You can use `wxDebugContext` to check for memory leaks and corrupt memory. See the debugging topic in the reference manual for more information.

### 8.3.7. 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.

## 9. Alphabetical class reference

See also *Writing a wxWindows application: a rough guide* (page 413)

*Notes on using the reference* (page 419)

*Guide to functions* (page 328)

### 9.1. wxApp: wxObject

See also *wxApp overview* (page 383)

The **wxApp** class represents the application itself.

#### **wxApp::wxApp**

**void wxApp(int language = wxLANGUAGE\_ENGLISH)**

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.

#### **wxApp::~~wxApp**

**void ~wxApp(void)**

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

#### **wxApp::argc**

**int argc**

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

#### **wxApp::argv**

**char \*\* argv**

Command line arguments (after environment-specific processing).

#### **wxApp::wx\_class**

**char \* wx\_class**

Currently used under Motif only, where the value is passed to *XtOpenDisplay* on initialization. Set this member in the constructor of your derived **wxApp** class to give the application a name other than the default "wxApp".

**wxApp::work\_proc****void \* work\_proc**

Set this member to the address of a function that takes a pointer to **wxApp**. It will be called whenever the system is idle and can be used to schedule background tasks.

**wxApp::Dispatch****void Dispatch(void)**

Dispatches the next event in the windowing system event queue. (MS Windows and Motif). See also *wxApp::Pending* (page 47).

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

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

**wxApp::GetAppName****char \* GetAppName(void)**

Returns the application name. *wxWindows* sets this to a reasonable default before calling *wxApp::OnInit*, but the application can reset it at will.

**wxApp::GetClassName****char \* GetClassName(void)**

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

**wxApp::GetExitOnDelete****Bool GetExitOnDelete(void)**

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

**wxApp::GetPrintMode****Bool GetPrintMode(void)**

Returns the print mode: see *SetPrintMode* (page 48).

**wxApp::GetTopWindow****wxWindow \* GetTopWindow(void)**

Returns a pointer to the top window (as returned by `wxApp::OnInit`). This may return `NULL` if called before the end of `wxApp::OnInit`.

### **wxApp::ExitMainLoop**

**void ExitMainLoop(void)**

Call this to explicitly exit the main message (event) loop. You should normally exit the main loop (and the application) by deleting the frame returned from `wxApp::OnInit`.

### **wxApp::Initialized**

**Bool Initialized(void)**

Returns `TRUE` if the application has been initialized (i.e. if *OnInit* (page 46) 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).

### **wxApp::MainLoop**

**int MainLoop(void)**

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

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

### **wxApp::OnExit**

**int OnExit(void)**

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

### **wxApp::OnCharHook**

**Bool OnCharHook(wxKeyEvent& ch)**

This member is called (under Windows only) to allow the window to intercept keyboard events before they are processed by child windows. The default implementation forwards the message to the currently active window. The function should return `TRUE` to indicate the character has been processed, or `FALSE` to allow default processing.

See also *wxKeyEvent* (page 193), *wxEvtHandler::OnChar* (page 151), *wxEvtHandler::OnCharHook* (page 152), *wxDialogBox::OnCharHook* (page 125).

### **wxApp::OnInit**

**wxFrame \* OnInit(void)**

This must be provided by the application, and must create and return the application's main window.

**wxApp::Pending****Bool Pending(void)**

Returns TRUE if unprocessed events are in the window system event queue (MS Windows and Motif). See also *wxApp::Dispatch* (page 45).

**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);
}
```

**wxApp::SetAppName****void SetAppName(char \*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.

**wxApp::SetClassName****void SetClassName(char \*name)**

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

**wxApp::SetExitOnDelete****void SetExitOnDelete(Bool flag)**



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

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

### **wxApp::SetPrintMode**

**void SetPrintMode(int mode)**

Sets the print mode determining what printing facilities will be used by the printing framework.

- wxPRINT\_WINDOWS: under Windows, use Windows printing (wxPrinterDC). This is the default under Windows.
- wxPRINT\_POSTSCRIPT: use PostScript printing (wxPostScriptDC). This is the default for non-Windows platforms.

## **9.2. wxBitmap: wxObject**

See also *Overview* (page 384)

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

### **wxBitmap::wxBitmap**

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

Constructs a (usually monochrome) bitmap from an array of pixel values, under both X and Windows.

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

Constructs a new bitmap. If the final argument is omitted, the display depth of the screen is used.

**void wxBitmap(char \*\*bits)**

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);
```

**void wxBitmap(char \*name, long flags)**

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, *flags* defaults to `wxBITMAP_TYPE_BMP_RESOURCE | wxBITMAP_DISCARD_COLOURMAP`. Under X, *flags* defaults to `wxBITMAP_TYPE_XBM | wxBITMAP_DISCARD_COLOURMAP`.

The meaning of *name* is determined by the *flags* parameter which may be a bit list of **`wxBITMAP_DISCARD_COLOURMAP`** (meaning the colourmap read, if any, should be thrown away) and one of:

<code>wxBITMAP_TYPE_BMP</code>	Load a Windows bitmap file.
<code>wxBITMAP_TYPE_BMP_RESOURCE</code>	Load a Windows bitmap from the resource database.
<code>wxBITMAP_TYPE_GIF</code>	Load a GIF bitmap file.
<code>wxBITMAP_TYPE_XBM</code>	Load an X bitmap file.
<code>wxBITMAP_TYPE_XPM</code>	Load an XPM bitmap file.
<code>wxBITMAP_TYPE_RESOURCE</code>	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, BMP\_RESOURCE, XPM\_DATA, and XPM. Under X, the available formats are BMP, GIF, XBM, and XPM.

## **`wxBitmap::~wxBitmap`**

**`void ~wxBitmap(void)`**

Destroys the bitmap. The bitmap will be destroyed automatically by `wxWindows` when the application exits.

## **`wxBitmap::Create`**

**`void Create(int width, int height int depth = -1)`**

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

## **`wxBitmap::GetColourMap`**

**`wxColourMap * GetColourMap(void)`**

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

## **`wxBitmap::GetDepth`**

**`int GetDepth(void)`**

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

## **`wxBitmap::GetHeight`**

**int GetHeight(void)**

Gets the height of the bitmap in pixels.

**wxBitmap::GetWidth****int GetWidth(void)**

Gets the width of the bitmap in pixels.

**wxBitmap::LoadFile****Bool LoadFile(char \*name, long flags)**

Loads 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.

The meaning of *name* is determined by the *flags* parameter which may be a bit list of **wxBITMAP\_DISCARD\_COLOURMAP** (meaning the colourmap read, if any, should be thrown away) and one of:

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.

A colourmap 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 *GetColourMap* (page 49) member.

**wxBitmap::Ok****Bool Ok(void)**

Returns TRUE if the bitmap was successfully created.

**wxBitmap::SaveFile****Bool SaveFile(char \*name, int type, wxColourMap \*cmap)**

Saves a bitmap in the named file.

The type of saved is determined by the *type* parameter which may be one of:

wxBITMAP_TYPE_BMP	Save a Windows bitmap file.
wxBITMAP_TYPE_GIF	Save a GIF bitmap file.

<code>wxBITMAP_TYPE_XBM</code>	Save an X bitmap file.
<code>wxBITMAP_TYPE_XPM</code>	Save an XPM bitmap file.

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

If a colourmap is supplied, it may be used when saving the bitmap. If this parameter is NULL and there is a colourmap associated with the bitmap, this internal colourmap may be used instead.

## **wxBitmap::SetColourMap**

**void SetColourMap(wxColourMap \*cmap)**

Sets the associated colourmap: it will be deleted in the wxBitmap destructor, so if you do not wish it to be deleted automatically, reset the colourmap to NULL before the bitmap is deleted.

## **9.3. wxBrush: wxObject**

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.

The style may be one of:

- `wxTRANSPARENT`
- `wxSOLID`
- `wxBDIAGONAL_HATCH`
- `wxCROSSDIAG_HATCH`
- `wxFDIAGONAL_HATCH`
- `wxCROSS_HATCH`
- `wxHORIZONTAL_HATCH`
- `wxVERTICAL_HATCH`

On a monochrome display, the default behaviour is to show all brushes as white unless the colour is really black. If you wish the policy to be 'all non-white colours are black', as with pens, uncomment the piece of code documented in `wxDC::SetBrush` (page 117) in `wx_dc.cpp`. Alternatively, set the **Colour** member of the device context to TRUE, and select appropriate colours.

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` (page 46) 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**. See `wxBrushList` (page 53) and `wxDC` (page 108).

## **wxBrush::wxBrush**

**void wxBrush(void)**

**void wxBrush(wxColour &colour, int style)**

**void wxBrush(char \*colour\_name, int style)**

Constructs a brush: uninitialized, initialized with an RGB colour and a style, or initialized using a colour name and a style (see *wxBrush::SetStyle* (page 53)). If the named colour form is used, an appropriate *wxColour* (page 76) structure is found in the colour database.

**wxBrush::~~wxBrush**

**void ~wxBrush(void)**

Destructor, destroying the brush. Note that brushes should very rarely be deleted since windows may contain pointers to them. All brushes will be deleted when the application terminates.

If you have to delete a brush, then call *wxDC::SetBrush* (page 117) with a NULL argument to ensure that the old brush is restored, and the current brush is selected out of the device context.

**wxBrush::GetColour**

**wxColour& GetColour(void)**

Returns a reference to the brush colour.

**wxBrush::GetStipple**

**wxBitmap \* GetStipple(void)**

Gets the stipple bitmap.

**wxBrush::GetStyle**

**int GetStyle(void)**

Returns the brush style, one of:

- wxTRANSPARENT
- wxSOLID
- wxBDIAGONAL\_HATCH
- wxCROSSDIAG\_HATCH
- wxFDIAGONAL\_HATCH
- wxCROSS\_HATCH
- wxHORIZONTAL\_HATCH
- wxVERTICAL\_HATCH

**wxBrush::SetColour**

**void SetColour(wxColour &colour)**

**void SetColour(char \*colour\_name)**

**void SetColour(int red, int green, int blue)**

The brush's colour is changed to the given colour.

### **wxBrush::SetStipple**

**void SetStipple(wxBitmap \*bitmap)**

Sets the stipple bitmap.

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.

### **wxBrush::SetStyle**

**void SetStyle(int style)**

Sets the brush style, one of:

- wxTRANSPARENT
- wxSOLID
- wxBDIAGONAL\_HATCH
- wxCROSSDIAG\_HATCH
- wxFDIAGONAL\_HATCH
- wxCROSS\_HATCH
- wxHORIZONTAL\_HATCH
- wxVERTICAL\_HATCH

## **9.4. wxBrushList: wxList**

A brush list is a list containing all brushes which have been created. 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 is best 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'. See *wxBrush* (page 51).

### **wxBrushList::wxBrushList**

**void wxBrushList(void)**

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

### **wxBrushList::AddBrush**

**void AddBrush(wxBrush \*brush)**

Used by `wxWindows` to add a brush to the list, called in the brush constructor.

### **wxBrushList::FindOrCreateBrush**

```
wxBrush * FindOrCreateBrush(wxColour *colour, int style)
```

```
wxBrush * FindOrCreateBrush(char *colour_name, int style)
```

Finds a brush of the given specification, or creates one and adds it to the list. See `wxBrush::SetStyle` (page 53) for a list of styles.

### **wxBrushList::RemoveBrush**

```
void RemoveBrush(wxBrush *brush)
```

Used by `wxWindows` to remove a brush from the list.

## **9.5. wxButton: wxItem**

A button is a *panel item* that contains a text string or bitmap, and is one of the commonest elements of a GUI. It may be placed on a *dialog box* (page 123) or *panel* (page 228).

### **wxButton::wxButton**

```
void wxButton(wxPanel *parent, wxFunction func, char *label,  
int x = -1, int y = -1, int width = -1, int height = -1,  
long style = 0, char *name = "button")
```

```
void wxButton(wxPanel *parent, wxFunction func, wxBitmap *wxBitmap,  
int x = -1, int y = -1, int width = -1, int height = -1,  
long style = 0, char *name = "button")
```

Constructor, creating and showing a button. The parent must be a valid panel or dialog box pointer.

*func* may be `NULL`; otherwise it is used as the callback for the button. Note that the cast (`wxFunction`) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration. See `wxFunction` (page 180).

The parameters *x* and *y* are used to specify an absolute position, or a position after the previous panel item if omitted or default.

If *width* or *height* are omitted (or are less than zero), an appropriate size will be used for the item. The *style* parameter is reserved for future use. The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual buttons.

If the first form is used, the *label* will be shown on the button. If the second form is used, the *bitmap* will be used.

**wxButton::~~wxButton****void ~wxButton(void)**

Destructor, destroying the button.

**wxButton::Create****void Create(wxPanel \*parent, wxFunction func, char \*label,**  
**int x = -1, int y = -1, int width = -1, int height = -1,**  
**long style = 0, char \*name = "button")****void Create(wxPanel \*parent, wxFunction func, wxBitmap \*wxBitmap,**  
**int x = -1, int y = -1, int width = -1, int height = -1,**  
**long style = 0, char \*name = "button")**

Button creation functions called by the button constructors. Call these when a derived button class uses the zero-argument **wxButton** constructor, but can reuse the existing button creation code. See *wxButton::wxButton* (page 54) for details.

**wxButton::SetDefault****void SetDefault(void)**

This sets the button to be the default item for the panel or dialog box. Under XView, the default item is highlighted, and pressing the return key executes the callback for the item (but with no visual feedback, and only if a text item does not have the focus).

Under MS Windows, only dialog box buttons respond to this function. As normal under MS Windows and Motif, pressing return causes the default button to be depressed when the return key is pressed. See also *wxWindow::SetFocus* (page 325) which sets the keyboard focus for windows and text panel items, *wxPanel::OnDefaultAction* (page 231) and *wxPanel::GetDefaultItem* (page 229).

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.

**wxButton::SetLabel****void SetLabel(wxBitmap \*label)****void SetLabel(char \*label)**

Sets the string or bitmap label for a button.

**9.6. wxButtonBar: wxToolBar**

See also *Overview* (page 392)

A *wxButtonBar* very similar to the *wxToolBar* (page 308), but is optimized for use under MS



Windows, giving a more attractive appearance and better feedback. Include the file `wx_bbar.h` to use this class.

See the comments in documentation for `wxToolBar` for functions such as `CreateTools` that have are harmless when called for `wxToolBar` but have specific meaning for `wxButtonBar` under Windows 95. `CreateTools` *must* be called under Windows for `wxButtonBar`.

*Note:* under Windows 95, a `wxButtonBar` cannot be moved to any position other than the top-left of the frame. If this is a problem, you may wish to alter `wx_bbar.h` and `wx_bbar.cpp` to compile the non-Windows 95 code instead.

### **wxButtonBar::wxButtonBar**

```
void wxButtonBar(wxWindow *parent, int x = 0, int y = 0,  
    int width = -1, int height = -1, long style = 0,  
    int orientation = wxVERTICAL, int nRowsOrColumns = 1, char *name = "buttonBar")
```

Constructs a buttonbar panel (canvas under XView).

*parent* is a parent window, usually a `wxFrame`.

*x*, *y* set the position of the window.

*width*, *height* set the size of the window.

*style* is a bitlist, with no buttonbar specific flags at present.

*orientation* specifies a `wxVERTICAL` or `wxHORIZONTAL` orientation for laying out the buttonbar. Must always be `wxVERTICAL` under Windows 95.

*nRowsOrColumns* specifies the number of rows or columns, whose meaning depends on *orientation*. If laid out vertically, *nRowsOrColumns* specifies the number of rows to draw before the next column is started; if horizontal, it refers to the number of columns to draw before the next row is started. Under Windows 95, this value refers to the number of rows only.

*name* specifies a window name for the buttonbar.

### **wxButtonBar::GetDefaultButtonHeight**

```
float GetDefaultButtonHeight(void)
```

Returns the real height of the button (bitmap height plus the extra for 3D effects).

### **wxButtonBar::GetDefaultButtonWidth**

```
float GetDefaultButtonWidth(void)
```

Returns the real width of the button (bitmap width plus the extra for 3D effects).

### **wxButtonBar::SetDefaultSize**

**void SetDefaultSize(float width, float height)**

Sets the default size of the button bitmap. The default is 16x15 pixels.

## 9.7. wxCanvas: wxWindow

A canvas is a subwindow onto which graphics and text can be drawn, and mouse and keyboard input can be intercepted. At present, panel items cannot be placed on a canvas.

To determine whether a canvas is colour or monochrome, test the canvas's device context **Colour** boolean member variable.

When you draw onto a canvas, you are really drawing onto a *device context* (see *wxDC* (page 108), *wxCanvasDC* (page 68)). Although you can use the members of *wxCanvas* for drawing, it is much better to get the device context from the canvas (see *GetDC* (page 61)) and draw into that. Then, code which can draw into one device context can be reused for others, such as *PostScript* or memory device contexts (see *wxPostScriptDC* (page 240) and *wxMemoryDC* (page 205)).

### wxCanvas::wxCanvas

```
void wxCanvas(wxWindow *parent, int x = -1, int y = -1,  
    int width = -1, int height = -1,  
    long style = wxRETAINED, char *name = "canvas")
```

Constructor.

Under Windows and Motif, the parent can be either a frame or panel. Under XView, the parent must be a frame.

The parameters *x*, *y*, *width* and *height* can be omitted on construction if the position and size will later be set (for example by a application frame's **OnSize** callback, or if there is only one subwindow for the frame, in which case the subwindow fills the frame).

The style parameter may be a combination (using the C++ bitwise 'or' operator) of the following flags:

<b>wxBORDER</b>	Gives the canvas a thin border (MS Windows and Motif only).
<b>wxRETAINED</b>	Gives the canvas a wxWindows-implemented backing store, making repainting much faster but at a potentially costly memory premium (XView and Motif only).

The *name* parameter is used to associate a name with the canvas, allowing the application user to set Motif resources for individual canvases.

### wxCanvas::~~wxCanvas

```
void ~wxCanvas(void)
```

Destructor.

### wxCanvas::AllowDoubleClick

**void AllowDoubleClick(int interval)**

Allows or disables double click handling on a canvas (Motif, MS Windows). Specify a double-click interval in milliseconds.

See also *wxMouseEvent::ButtonDClick* (page 216).

**wxCanvas::BeginDrawing****void BeginDrawing(void)**

Allows optimization of drawing code under MS Windows. Enclose drawing primitives between **BeginDrawing** and **EndDrawing** calls.

**wxCanvas::Clear****void Clear(void)**

Clears the canvas (fills it with the current background brush).

**wxCanvas::Create**

```
void Create(wxWindow *parent, int x = -1, int y = -1,  
int width = -1, int height = -1,  
long style = wxRETAINED, char *name)
```

Creates the canvas for two-step construction. Derived classes should call or replace this function. See *wxCanvas::wxCanvas* (page 57) for details.

**wxCanvas::CrossHair****void CrossHair(float x, float y)**

Displays a cross hair using the current pen. This is a vertical and horizontal line the height and width of the canvas, centred on the given point.

**wxCanvas::DestroyClippingRegion****void DestroyClippingRegion(void)**

Destroys the current clipping region so that none of the canvas is clipped.

**wxCanvas::DrawArc****void DrawArc(float x1, float y1, float x2, float y2, float xc, float yc)**

Draws an arc, 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.

**wxCanvas::DrawEllipse****void DrawEllipse(float x, float y, float width, float 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.

**wxCanvas::DrawLine****void DrawLine(float x1, float y1, float x2, float y2)**

Draws a line from the first point to the second. The current pen is used for drawing the line.

**wxCanvas::DrawLines****void DrawLines(int n, wxPoint points[], float xoffset = 0, float yoffset = 0)****void DrawLines(wxList \*points, float xoffset = 0, float yoffset = 0)**

Draw 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.

**wxCanvas::DrawPolygon****void DrawPolygon(int n, wxPoint points[], float xoffset = 0, float yoffset = 0,  
int fill\_style = wxODDEVEN\_RULE)****void DrawPolygon(wxList \*points, float xoffset = 0, float yoffset = 0, int fill\_style =  
wxODDEVEN\_RULE)**

Draw 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.

**wxCanvas::DrawPoint****void DrawPoint(float x, float y)**

Draws a point using the current pen.

**wxCanvas::DrawRectangle****void DrawRectangle(float x, float y, float width, float 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.

**wxCanvas::DrawRoundedRectangle****void DrawRoundedRectangle(float x, float y, float width, float height, float 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.

**wxCanvas::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(float x1, float y1, float x2, float y2, float x3, float y3)**

Draws a three-point spline using the current pen.

**wxCanvas::DrawText****void DrawText(char \*text, float x, float y)**

Draws a text string at the specified point, using the current text font, and the current text foreground and background colours.

**wxCanvas::EnableScrolling****void EnableScrolling(Bool xScrolling, Bool yScrolling)**

Enable or disable Windows scrolling in the given direction, where in this context 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 messes up the display.

**wxCanvas::EndDrawing**

**void EndDrawing(void)**

Allows optimization of drawing code under MS Windows. Enclose drawing primitives between **BeginDrawing** and **EndDrawing** calls.

**wxCanvas::FloodFill****void FloodFill(float x, float y, wxColour \*colour, int style=wxFLOOD\_SURFACE)**

Flood fills the canvas 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.

**wxCanvas::GetDC****wxCanvasDC \* GetDC(void)**

Get a pointer to the canvas's device context. See `wxDC` (page 108) and `wxCanvasDC` (page 68).

**wxCanvas::GetScrollPage****int GetScrollPage(int orient)**

Returns the lines per page of the scrollbar. Pass `wxHORIZONTAL` or `wxVERTICAL` to indicate the scrollbar whose lines per page value is to be returned.

**wxCanvas::GetScrollPixelsPerUnit****void GetScrollPixelsPerUnit(int \*x\_unit, int \*y\_unit)**

Get the number of pixels per scroll unit (line), in each direction, as set by `wxCanvas::SetScrollbars` (page 66). A value of zero indicates no scrolling in that direction.

**wxCanvas::GetScrollPos****int GetScrollPos(int orient)**

Returns position (in scroll units) of a scrollbar. Pass `wxHORIZONTAL` or `wxVERTICAL` to indicate the scrollbar whose position is to be returned.

**wxCanvas::GetScrollRange****int GetScrollRange(int orient)**

Returns the maximum position of the scrollbar, in scroll units. Pass `wxHORIZONTAL` or

wxVERTICAL to indicate the scrollbar whose range is to be returned.

### **wxCanvas::GetScrollUnitsPerPage**

**void GetScrollUnitsPerPage(int \*x\_page, int \*y\_page)**

Get the number of units per page, in each direction, as set by *wxCanvas::SetScrollbars* (page 66). A value of zero indicates no scrolling in that direction.

### **wxCanvas::GetVirtualSize**

w

**void GetVirtualSize(int \*x, int \*y)**

Gets the size in device units of the scrollable canvas area (as opposed to the client size, which is the area of the canvas currently visible).

Use *wxDC::DeviceToLogicalX* (page 109) and *wxDC::DeviceToLogicalY* (page 109) to translate these units to logical units.

### **wxCanvas::IntDrawLine**

**void IntDrawLine(int x1, int y1, int x2, int y2)**

Draws a line from the first point to the second. The current pen is used for drawing the line.

### **wxCanvas::IntDrawLines**

**void IntDrawLines(int n, wxIntPoint points[], int xoffset = 0, int yoffset = 0)**

Draw lines using an array of *points* of size *n*. The current pen is used for drawing the lines. The programmer is responsible for deleting the list of points.

### **wxCanvas::IsRetained**

**Bool IsRetained(void)**

TRUE if the canvas has a backing bitmap.

### **wxCanvas::OnChar**

**void OnChar(wxKeyEvent& event)**

This default handler interprets cursor key movement and scrolls the canvas accordingly. Override the function to change this behaviour.

The member *keyCode* contains the key pressed.

See `wxEvtHandler::OnChar` (page 151) for more details.

## **wxCanvas::OnEvent**

**void OnEvent(wxMouseEvent& event)**

Sent to the canvas when the user has initiated an event with the mouse. Derive your own class to handle this message. See `wxCanvas::OnChar` (page 62) for character events, and also `wxMouseEvent` (page 214) for how to access event information.

## **wxCanvas::OnPaint**

**void OnPaint(void)**

Sent to the canvas when the canvas must be refreshed. Derive your own class to handle this message.

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 `wxUpdateIterator` (page 314) class:

```
// Called when canvas needs to be repainted.
void MyCanvas::OnPaint(void)
{
    // Speeds up drawing under Windows.
    GetDC()->BeginDrawing();
    wxCanvasDC *canvdc = GetDC();

    // 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
    wxUpdateIterator            upd(this); // get the update rect list

    while (upd)
    {
        vX = upd.GetX();
        vY = upd.GetY();
        vW = upd.GetW();
        vH = upd.GetH();

        // Alternatively we can do this:
        // wxRectangle rect;
        // upd.GetRect(&rect);

        // Repaint this rectangle
        <some code>

        upd ++ ;
    }
}
```



```
    GetDC()->EndDrawing();  
}
```

## **wxCanvas::OnScroll**

**void OnScroll(wxCommandEvent& 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 canvas contents at an appropriate position relative to the scrollbar.

The *commandInt* member of *wxCommandEvent* is the position of the scrollbar. The macro *WXSCROLLPOS(event)* may be used to access or set this member.

The *extraLong* member of *wxCommandEvent* is *wxHORIZONTAL* or *wxVERTICAL*. The macro *WXSCROLLORIENT(event)* may be used to access or set this member.

The *eventType* member of *wxCommandEvent* will be one of:

<i>wxEVENT_TYPE_SCROLL_LINEDOWN</i>	Called when the scrollbar is incremented by a line, by clicking on the bottom arrow of a vertical scrollbar or right-hand arrow of a horizontal scrollbar.
<i>wxEVENT_TYPE_SCROLL_LINEUP</i>	Called when the scrollbar is decremented by a line, by clicking on the top arrow of a vertical scrollbar or left-hand arrow of a horizontal scrollbar.
<i>wxEVENT_TYPE_SCROLL_PAGEDOWN</i>	Called when the scrollbar is incremented by a page.
<i>wxEVENT_TYPE_SCROLL_PAGEUP</i>	Called when the scrollbar is decremented by a page.
<i>wxEVENT_TYPE_SCROLL_TOP</i>	Called when the scrollbar is set to the top.
<i>wxEVENT_TYPE_SCROLL_BOTTOM</i>	Called when the scrollbar is set to the bottom.
<i>wxEVENT_TYPE_SCROLL_THUMBTRACK</i>	Called when the the scrollbar is being dragged.

Note that not all these messages will be received on a given platform. For example, under XView, only the thumbtrack event is ever generated, and so unless the application keeps track of where the scrollbar was previously, some scrolling optimisations are not possible.

Under Windows and Motif, all messages may be generated.

## **wxCanvas::Scroll**

**void Scroll(int x\_pos, int y\_pos)**

Scrolls a canvas so the view start is at the given point. 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 *wxCanvas::SetScrollbars* (page 66).

## **wxCanvas::SetBackground**

**void SetBackground(wxBrush \*brush)**

Sets the current background brush for the canvas, used for the pixels between dotted or dashed lines. The brush should not be deleted while being used by a canvas. All brushes are deleted automatically when the application terminates.

See also *wxBrush* (page 51).

**wxCanvas::SetClippingRegion****void SetClippingRegion(float x, float y, float width, float height)**

Sets the clipping region for the canvas. The clipping region is a rectangular area to which drawing is restricted. Possible uses for the clipping region are for clipping text or for speeding up canvas redraws when only a known area of the screen is damaged.

See also *wxCanvas::DestroyClippingRegion* (page 58).

**wxCanvas::SetBrush****void SetBrush(wxBrush \*brush)**

Sets the current brush for the canvas. The brush is not copied, so you should not delete the brush unless the canvas pen has been set to another brush, or to NULL. Note that all pens and brushes are automatically deleted when the program is exited.

See also *wxBrush* (page 51).

**wxCanvas::SetFont****void SetFont(wxFont \*font)**

Sets the current font for the canvas. The font is not copied, so you should not delete the font unless the canvas pen has been set to another font, or to NULL.

See also *wxFont* (page 158), *wxCanvas::DrawText* (page 60).

**wxCanvas::SetLogicalFunction****void SetLogicalFunction(int function)**

Sets the current logical function for the canvas. This determines how a source pixel (from a pen or brush colour, or source device context if using *wxDC::Blit* (page 108)) 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>wxSRC_AND</code>	<code>AND src (MS Windows Blit only: equivalent to</code>
<code>SRCAND)</code>	
<code>wxSRC_OR</code>	<code>OR src (MS Windows Blit only: equivalent to</code>
<code>SRCPAINT)</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.

## **wxCanvas::SetPen**

**void SetPen(wxPen \*pen)**

Sets the current pen for the canvas. The pen is not copied, so you should not delete the pen unless the canvas pen has been set to another pen, or to `NULL`. Note that all pens and brushes are automatically deleted when the program is exited.

See also *wxPen* (page 237).

## **wxCanvas::SetScrollbars**

**void SetScrollbars(int horiz\_pixels, int vert\_pixels, int x\_length, int y\_length, int x\_page, int y\_page, int x\_pos = 0, int y\_pos = 0)**

Sets up vertical and/or horizontal scrollbars. 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 effectively sets the size of the 'virtual canvas'. The third pair gives the number of scroll steps in a 'page', i.e. amount moved when pressing above or below the scrollbar control, or using page up/page down.

`x_pos` and `y_pos` optionally specify a position to scroll to immediately.

Either `x_length` `y_length` can be zero to specify no scrollbar.

For example, the following gives a canvas horizontal and vertical scrollbars with 20 pixels per scroll step, a size of 50 steps (1000 pixels) in each direction, and 4 steps (80 pixels) to a page.

```
canvas->SetScrollbars(20, 20, 50, 50, 4, 4);
```

See also *wxCanvas::EnableScrolling* (page 60), *wxCanvas::GetScrollUnitsPerPage* (page 62), *wxCanvas::GetVirtualSize* (page 62).

**wxCanvas::SetScrollPage****void SetScrollPage(int orient, int page)**

Sets the lines per page for a scrollbar. Pass wxHORIZONTAL or wxVERTICAL to indicate the scrollbar whose lines per page is to be set.

**wxCanvas::SetScrollPos****void SetScrollPos(int orient, int pos)**

Sets the position (in scroll units) of a scrollbar. Pass wxHORIZONTAL or wxVERTICAL to indicate the scrollbar whose position is to be set.

**wxCanvas::SetScrollRange****void SetScrollRange(int orient, int range)**

Sets the maximum position (in scroll units) of a scrollbar. Pass wxHORIZONTAL or wxVERTICAL to indicate the scrollbar whose range is to be set.

**wxCanvas::SetTextBackground****void SetTextBackground(wxColour \*colour)**

Sets the current text background colour for the canvas. The colour is copied by this function. See also *wxCanvas::SetTextForeground* (page 67).

**wxCanvas::SetTextForeground****void SetTextForeground(wxColour \*colour)**

Sets the current text foreground colour for the canvas. The colour is copied by this function. See also *wxCanvas::SetTextBackground* (page 67).

**wxCanvas::ViewStart****void ViewStart(int \*x, int \*y)**

Get the position at which the visible portion of the canvas starts. If either of the scrollbars is not at the home position, x and/or y will be greater than zero. Combined with *wxWindow::GetClientSize* (page 321), the application can use this function to efficiently redraw only the visible portion of the canvas. 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 *wxCanvas::SetScrollbars* (page 66).

**wxCanvas::WarpPointer****void WarpPointer(int x, int y)**

Moves the pointer to the given position on the canvas.

**9.8. wxCanvasDC: wxDC**

A canvas device context is automatically created when a canvas is created. It can be retrieved from a canvas with `wxCanvas::GetDC` (page 61) and then drawn into. See `wxDC` (page 108) for further information on device contexts.

**wxCanvasDC::wxCanvasDC****void wxCanvasDC(wxCanvas \*canvas)**

Constructor for internal use only.

**wxCanvasDC::GetClippingBox****void wxCanvasDC(float \*x, float \*y, float \*width, float \*height)**

Gets the current rectangular clipping region.

**9.9. wxCheckBox: wxItem**

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

**wxCheckBox::wxCheckBox****void wxCheckBox(void)**

Constructor, used when deriving from this class.

**void wxCheckBox(wxPanel \*parent, wxFunction func, char \*label,  
int x = -1, int y = -1, int width = -1, int height = -1,  
long style = 0, char \*name = "checkBox")****void wxCheckBox(wxPanel \*parent, wxFunction func, wxBitmap \*bitmap,  
int x = -1, int y = -1, int width = -1, int height = -1,  
long style = 0, char \*name = "checkBox")**

Constructor, creating and showing a checkbox.

`func` may be NULL; otherwise it is used as the callback for the check box. Note that the cast (wxFunction) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration.

In the second form, if `label` is non-NULL, it is used as the label for the checkbox. In the third form, a bitmap is provided instead of a text label.

The parameters `x` and `y` are used to specify an absolute position, or a position after the previous

panel item if omitted or default.

If *width* or *height* are omitted (or are less than zero), an appropriate size will be used for the check box.

The *style* parameter is reserved for future use.

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual checkboxes.

### **wxCheckBox::~~wxCheckBox**

**void ~wxCheckBox(void)**

Destructor, destroying the checkbox.

### **wxCheckBox::Create**

**Bool Create(wxPanel \*parent, wxFunction func, char \*label,  
int x = -1, int y = -1, int width = -1, int height = -1,  
long style = 0, char \*name = "checkBox")**

**Bool Create(wxPanel \*parent, wxFunction func, char \*bitmap,  
int x = -1, int y = -1, int width = -1, int height = -1,  
long style = 0, char \*name = "checkBox")**

Creates the checkbox for two-step construction. Derived classes should call or replace this function. See *wxCheckBox::wxCheckBox* (page 68) for details.

### **wxCheckBox::GetValue**

**Bool GetValue(void)**

Gets the state of the checkbox, TRUE if it is checked, FALSE otherwise.

### **wxCheckBox::SetLabel**

**void SetLabel(wxBitmap \*label)**

Sets the bitmap for a bitmap checkbox.

### **wxCheckBox::SetValue**

**void SetValue(Bool state)**

Sets the checkbox to the given state: if the state is TRUE, the check is on, otherwise it is off.

## **9.10. wxChoice: wxItem**

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. Under XView and Motif, all selections are visible when the menu is displayed. Under MS Windows, a scrolling list is displayed when the user wants to change the selection. Note that under XView, creating a choice item with a large number of strings takes a long time due to the inefficiency of Sun's implementation of the XView choice item.

See also *wxListBox* (page 201).

## **wxChoice::wxChoice**

### **void wxChoice(void)**

Constructor, for use by derived classes.

```
void wxChoice(wxPanel *parent, wxFunction func, char *label,  
int x = -1, int y = -1, int width = -1, int height = -1,  
int n, char *choices[],  
long style = 0, char *name = "choice")
```

Constructor, creating and showing a choice.

*func* may be NULL; otherwise it is used as the callback for the choice. Note that the cast (*wxFunction*) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration.

If *label* is non-NULL, it is used as the label for the choice item.

The parameters *x* and *y* are used to specify an absolute position, or a position after the previous panel item if omitted or default.

If *width* or *height* are omitted (or are less than zero), an appropriate size will be used for the choice.

*n* is the number of possible choices, and *choices* is an array of strings of size *n*. *wxWindows* allocates its own memory for these strings so the calling program must deallocate the array itself.

The *style* parameter is a bitlist of the following:

**wxFIXED\_LENGTH**      Allows the values of a column of items to be left-aligned. Create an item with this style, and pad out your labels with spaces to the same length. The item labels will initially be created with a string of identical characters, positioning all the values at the same x-position. Then the real label is restored.

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual choice items.

## **wxChoice::~~wxChoice**

### **void ~wxChoice(void)**

Destructor, destroying the choice item.

**wxChoice::Append****void Append(char \* item)**

Adds the item to the end of the choice item. *item* must be deallocated by the calling program, i.e. *wxWindows* makes its own copy.

**wxChoice::Clear****void Clear(void)**

Clears the strings from the choice item. Under XView, this is done by deleting and reconstructing the item, but it doesn't redisplay properly until the user refreshes the window.

**wxChoice::Create****Bool Create(wxPanel \*parent, wxFunction func, char \*label,  
int x = -1, int y = -1, int width = -1, int height = -1, int n, char \*choices[]  
long style = 0, char \*name = "choice")**

Creates the choice for two-step construction. Derived classes should call or replace this function. See *wxChoice::wxChoice* (page 70) for further details.

**wxChoice::FindString****int FindString(char \*s)**

Finds a choice matching the given string, returning the position if found, or -1 if not found.

**wxChoice::GetColumns****int GetColumns(void)**

Gets the number of columns in this choice item.

This is implemented for XView and Motif only.

**wxChoice::GetSelection****int GetSelection(void)**

Gets the id (position) of the selected string.

**wxChoice::GetString****char \* GetString(int n)**



Returns a temporary pointer to the string at position *n*.

### **wxChoice::GetStringSelection**

**char \* GetStringSelection(void)**

Gets the selected string. This must be copied by the calling program if long term use is to be made of it.

### **wxChoice::Number**

**int Number(void)**

Returns the number of strings in the choice item.

### **wxChoice::SetColumns**

**void SetColumns(int *n* = 1)**

Sets the number of columns in this choice item.

This is implemented for XView and Motif only.

### **wxChoice::SetSelection**

**void SetSelection(int *n*)**

Sets the choice by passing the desired string position.

### **wxChoice::SetStringSelection**

**void SetStringSelection(char \* *s*)**

Sets the choice by passing the desired string.

### **wxChoice::GetString**

**char \* GetString(int *n*)**

Returns a temporary pointer to the string at position *n*.

## **9.11. wxClassInfo**

See also *Overview* (page 371)

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`.

**wxClassInfo::wxClassInfo**

**void 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.

**wxClassInfo::CreateObject**

**wxObject \* CreateObject(void)**

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

**wxClassInfo::FindClass**

**static wxClassInfo \* FindClass(char \*name)**

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

**wxClassInfo::GetBaseClassName1**

**char \* GetBaseClassName1(void)**

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

**wxClassInfo::GetBaseClassName2**

**char \* GetBaseClassName2(void)**

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

**wxClassInfo::GetClassName**

**char \* GetClassName(void)**

Returns the string form of the class name.

**wxClassInfo::GetSize**

**int GetSize(void)**

Returns the size of the class.

**wxClassInfo::InitializeClasses****static void InitializeClasses(void)**

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

**wxClassInfo::IsKindOf****Bool IsKindOf(wxClassInfo \*info)**

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

**9.12. wxClient: wxIPCObject**

See also *Interprocess communications overview* (page 378)

A wxClient 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 wxConnection and another from wxClient. The custom wxConnection class will intercept communications in a 'conversation' with a server, and the custom wxServer is required so that a user-overridden *wxClient::OnMakeConnection* (page 74) member can return a wxConnection of the required class, when a connection is made.

See also *wxServer* (page 277), *wxConnection* (page 91), the chapter on interprocess communication in the user manual, and the programs in *samples/ipc*.

**wxClient::wxClient****void wxClient(void)**

Constructs a client object.

**wxClient::MakeConnection****wxConnection \* MakeConnection(char \*host, char \*service, char \*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 wxConnection object will be returned. The type of wxConnection returned can be altered by overriding the *wxClient::OnMakeConnection* (page 74) member to return your own derived connection object.

**wxClient::OnMakeConnection****wxConnection \* OnMakeConnection(void)**

The type of *wxConnection* (page 91) returned from a *wxClient::MakeConnection* (page 74) call can be altered by deriving the **OnMakeConnection** member to return your own derived

connection object. By default, an ordinary `wxConnection` 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 `wxConnection::OnAdvise` (page 92). You may also want to store application-specific data in instances of the new class.

### **wxClient::ValidHost**

**Bool ValidHost(char \*host)**

Returns TRUE if this is a valid host name, FALSE otherwise. This always returns TRUE under MS Windows.

## **9.13. wxClipboard: wxObject**

There is one `wxClipboard` object referenced by the pointer `wxTheClipboard`, initialized by calling `wxInitClipboard` (page 348). Under X, clipboard manipulation must be done by using this class, and such code will work under MS Windows also. Under MS Windows, you have the alternative of using the normal clipboard functions.

The documentation for this class will be expanded in due course. At present, `wxClipboard` is only used in the `wxMediaWindow` add-on library.

See also `wxClipboardClient` (page 76), `wxInitClipboard` (page 348).

### **wxClipboard::GetClipboardClient**

**wxClipboardClient \* GetClipboardClient(void)**

Get the clipboard client directly. Will be NULL if clipboard data is a string, or if some other application owns the clipboard. This can be useful for shortcutting data translation, if the clipboard user can check for a specific client.

### **wxClipboard::GetClipboardData**

**char \* GetClipboardData(char \*format, long \*length, long time)**

Get data from the clipboard.

### **wxClipboard::GetClipboardString**

**char \* GetClipboardString(long time)**

Get the data from the clipboard in the format "TEXT".

### **wxClipboard::SetClipboardClient**

**void SetClipboardClient(wxClipboardClient \*client, long time)**

Set the clipboard data owner.

**wxClipboard::SetClipboardString****void SetClipboardString(char \*data, long time)**

Set the clipboard string; does not require a client.

**9.14. wxClipboardClient: wxObject**

Implemented under X and MS Windows, a clipboard client holds data belonging to the clipboard. For plain text, a client is not necessary.

wxClipboardClient is an abstract class for which the virtual functions BeingReplaced and GetData must be overridden.

See also *wxClipboard* (page 75), *wxInitClipboard* (page 348).

**wxClipboardClient::formats****wxStringList formats**

This list should be filled in with strings indicating the formats this client can provide. Almost all clients will provide "TEXT". Format names should be 4 characters long, so things will work out on the Macintosh.

**wxClipboardClient::BeingReplaced****void BeingReplaced(void)**

This method is called when the client is losing the selection.

**wxClipboardClient::GetData****char \* GetData(char \*format, long \*size)**

This method is called when someone wants the data this client is supplying to the clipboard.

*format* is a string indicating the format of the data - one of the strings from the "formats" list.

*size* should be filled with the size of the resulting data. In the case of text, *size* does not count the NULL terminator.

**9.15. wxColour: wxObject**

A colour is an object representing a Red, Green, Blue (RGB) combination of primary colours, and is used to determine drawing colours. See the entry for *wxColourDatabase* (page 79) 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.

**wxColour::wxColour****void wxColour(char red, char green, char blue)****void wxColour(char \* colour\_name)**

Construct a colour object from the RGB values or using a colour name (uses **wxTheColourDatabase**).

**wxColour::operator =****wxColour& operator =(wxColour& src)**

Assignment from source to destination colour.

**wxColour::Blue****unsigned char Blue(void)**

Returns the blue intensity.

**wxColour::Get****void Get(char \* red, char \* green, char \* blue)**

Gets the RGB values---pass pointers to three char variables.

**wxColour::Green****unsigned char Green(void)**

Returns the green intensity.

**wxColour::Ok****Bool Ok(void)**

Returns TRUE if the colour object is valid.

**wxColour::Red****unsigned char Red(void)**

Returns the red intensity.

**wxColour::Set**

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

Sets the RGB value.

### **9.16. wxColourData: wxObject**

See also *wxColourDialog overview* (page 386)

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

#### **wxColourData::wxColourData**

**void wxColourData(void)**

Constructor. Initializes the custom colours to white, the *dataColour* member to black, and the *chooseFull* member to TRUE.

#### **wxColourData::~~wxColourData**

**void ~wxColourData(void)**

Destructor.

#### **wxColourData::GetChooseFull**

**Bool GetChooseFull(void)**

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.

#### **wxColourData::GetColour**

**wxColour& GetColour(void)**

Gets the current colour associated with the colour dialog.

The default colour is black.

#### **wxColourData::GetCustomColour**

**wxColour& GetCustomColour(int i)**

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.

**wxColourData::SetChooseFull****void SetChooseFull(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.

**wxColourData::SetColour****void SetColour(wxColour& *colour*)**

Sets the default colour for the colour dialog.

The default colour is black.

**wxColourData::SetCustomColour****void SetColour(int *i*, 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.

**wxColourData::operator =****void operator =(const wxColourData& *data*)**

Assignment operator for the colour data.

**9.17. wxColourDatabase: wxObject**

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**.

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.



wxWindows' colour handling under XView and Motif prior to Version 1.50 was poor, due to a bug in the code which allocates colours. This has now been fixed and a greater range of colours may be allocated. However, if a very wide range of colours is used in an application, wxWindows may still fail to allocate a colour, so it is best to choose a fixed number of colours, pens or brushes rather than allocate colours when needed.

See also *wxColour* (page 76).

## **wxColourDatabase::wxColourDatabase**

**void wxColourDatabase(void)**

Constructs the colour database. Should not need to be used by an application.

## **wxColourDatabase::FindColour**

**wxColour \* FindColour(char \*colour\_name)**

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

## **wxColourDatabase::FindName**

**char \* FindName(wxColour& colour)**

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

## **wxColourDatabase::Initialize**

**void Initialize(void)**

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

## **9.18. wxColourDialog: wxDialogBox**

See also *Overview* (page 386)

This class represents the colour chooser dialog. This is available under Motif and Windows. Under XView there seem to be some problems, probably related to modal dialogs.

## **wxColourDialog::wxColourDialog**

**void 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.

## **wxColourDialog::~~wxColourDialog**

**void ~wxColourDialog(void)**

Destructor.

**wxColourDialog::GetColourData**

**wxColourData& GetColourData(void)**

Returns the *colour data* (page 78) associated with the colour dialog.

**wxColourDialog::Show**

**Bool Show(Bool flag)**

Shows the dialog, returning TRUE if the user pressed Ok, and FALSE otherwise.

### 9.19. wxColourMap: wxObject

Colourmap functionality is incomplete, and will be extended in the future. Currently, colourmaps may be returned from some wxWindows libraries that load bitmaps (e.g. wxImage, DIB). To display a bitmap, its colourmap should normally be set for that window.

There are some strict rules for colourmap useage. A colourmap should never be deleted before being deselected from a window or device context (although it may be used for several windows and device contexts simultaneously). So, call *wxDC::SetColourMap* (page 117) with a NULL argument to make sure that its original (probably system) colourmap is restored.

If you are relying on wxWindows to clean up your bitmaps on program exit, then you must be extra vigilant about cleaning up colourmaps before bitmaps (and windows) are deleted. So it may not be an option to use global objects, where you cannot be sure of the order that C++ destroys objects; use dynamically created and destroyed objects instead.

**wxColourMap::wxColourMap**

**void wxColourMap(void)**

Constructor.

**wxColourMap::~~wxColourMap**

**void ~wxColourMap(void)**

Destructor.

If you have to delete the colourmap (for example, you are creating a lot of them), then call *wxDC::SetColourMap* (page 117) with a NULL argument to ensure that the old colourmap is restored, and the current colourmap is selected out of the device context.

**wxColourMap::Create**

**Bool Create**(const int *n*, const char \**red*,  
const char \**green*, const char \**blue*)

Creates a colourmap from arrays of size *n*, one for each red, blue or green component. Implemented only under Windows.

## 9.20. wxComboBox: wxItem

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 elements are numbered from zero.

See also *wxChoice* (page 69), *wxListBox* (page 201).

The callback function specified for the combobox item will be called for the following events:

- wxEVENT\_TYPE\_COMBOBOX\_COMMAND (the selection has changed)

*Note:* this is an experimental panel item, and is implemented for Windows and Motif only. There are some problems with the Motif implementation, which uses a contributed widget.

### wxComboBox::wxComboBox

**void wxComboBox**(void)

Constructor, for deriving classes.

**void wxComboBox**(wxPanel \**parent*, wxFunction *func*, char \**label*, char \**value* = "", int *x* = -1, int *y* = -1, int *width* = -1, int *height* = -1, int *n*, char \**choices*[], long *style* = 0, char \**name* = "comboBox")

Constructor, creating and showing a combobox.

*func* may be NULL; otherwise it is used as the callback for the combobox. Note that the cast (wxFunction) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration.

If *label* is non-NULL, it will be used as the combobox label.

*value* is the value to place in the edit field.

The parameters *x* and *y* are used to specify an absolute position, or a position after the previous panel item if omitted or default.

If *width* or *height* are omitted (or are less than zero), an appropriate size will be used for the combobox.

*n* is the number of possible choices, and *choices* is an array of strings of size *n*. wxWindows allocates its own memory for these strings so the calling program must deallocate the array itself.

*style* is a bit list of some of the following.

**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 (Windows only).

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual comboboxes.

### **wxComboBox::~~wxComboBox**

**void ~wxComboBox(void)**

Destructor, destroying the combobox.

### **wxComboBox::Append**

**void Append(char \* item)**

Adds the item to the end of the combobox. *item* must be deallocated by the calling program, i.e. *wxWindows* makes its own copy.

**void Append(char \* item, char \*client\_data)**

Adds the item to the end of the combobox, associating the given data with the item. *item* must be deallocated by the calling program.

### **wxComboBox::Clear**

**void Clear(void)**

Clears all strings from the combobox.

### **wxComboBox::Create**

**Bool Create(wxPanel \*parent, wxFunction func, char \*label,  
char \*value = "", int x = -1, int y = -1,  
int width = -1, int height = -1, int n, char \*choices[],  
long style = 0, char \*name = "comboBox")**

Creates the combobox for two-step construction. Derived classes should call or replace this function. See *wxComboBox::wxComboBox* (page 82) for further details.

### **wxComboBox::Copy**

**void Copy(void)**

Copies the selected text to the clipboard under Motif and MS Windows.

### **wxComboBox::Cut**

**void Cut(void)**

Copies the selected text to the clipboard and removes the selection. Windows and Motif only.

### **wxComboBox::Delete**

**void Delete(int *n*)**

Delete the *n*th element in the combobox.

### **wxComboBox::Deselect**

**void Deselect(int *n*)**

Deselects the given item in the combobox.

### **wxComboBox::FindString**

**int FindString(int *char* \**s*)**

Finds a choice matching the given string, returning the position if found, or -1 if not found.

### **wxComboBox::GetClientData**

**char \* GetClientData(int *n*)**

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

### **wxComboBox::GetInsertionPoint**

**long GetInsertionPoint(void)**

Returns the insertion point. Windows and Motif only.

### **wxComboBox::GetLastPosition**

**long GetLastPosition(void)**

Returns the last position in the text field. Windows and Motif only.

### **wxComboBox::GetSelection**

**int GetSelection(void)**

Gets the id (position) of the selected string.

**wxComboBox::GetString****char \* GetString(int *n*)**

Returns a temporary pointer to the string at position *n*.

**wxComboBox::GetStringSelection****char \* GetStringSelection(void)**

Gets the selected string - for single selection comboboxes only. This must be copied by the calling program if long term use is to be made of it.

**wxComboBox::GetValue****char \* GetValue(void)**

Gets a pointer to the current value. Copy this for long-term use.

**wxComboBox::Number****int Number(void)**

Returns the number of items in the combobox list.

**wxComboBox::Paste****void Paste(void)**

Pastes text from the clipboard to the text item. Windows and Motif only.

**wxComboBox::Remove****void Remove(long *from*, long *to*)**

Removes the text between the two positions. Windows and Motif only.

**wxComboBox::SetClientData****void SetClientData(int *n*, char \**data*)**

Associates the given client data pointer with the given item.

**wxComboBox::Replace****void Replace(long from, long to, char \*value)**

Replaces the text between two positions with the given text. Windows and Motif only.

**wxComboBox::SetInsertionPoint****void SetInsertionPoint(long pos)**

Sets the insertion point. Windows only.

**wxComboBox::SetInsertionPointEnd****void SetInsertionPointEnd(void)**

Sets the insertion point at the end of the text item. Windows and Motif only.

**wxComboBox::SetSelection****void SetSelection(int n, Bool select = TRUE)**

Selects or deselects the given item.

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

Selects the text between the two positions. Windows and Motif only.

**wxComboBox::SetValue****void SetValue(char \* value)**

Sets the text for the editable field. *value* must be deallocated by the calling program.

**9.21. wxCommand: wxObject**

See also *Overview* (page 375)

`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.

**wxCommand::wxCommand****void wxCommand(Bool canUndo = FALSE, char \*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.

### **wxCommand::~~wxCommand**

**void ~wxCommand(void)**

Destructor.

### **wxCommand::CanUndo**

**Bool CanUndo(void)**

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

### **wxCommand::Do**

**Bool Do(void)**

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.

### **wxCommand::GetName**

**char \* GetName(void)**

Returns the command name.

### **wxCommand::Undo**

**Bool Undo(void)**

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.

## 9.22. wxCommandEvent: wxEvent

This event class contains information about panel item command events. It is passed to *wxFunction* (page 180) panel item callbacks. It can also be constructed by an application and used with *wxSendEvent* (page 338) to simulate a user command in a panel item.

### wxCommandEvent::clientData

**char \* clientData**

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

### wxCommandEvent::commandInt

**int 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.

### wxCommandEvent::commandString

**char \* commandString**

Contains a string corresponding to a listbox or choice selection.

### wxCommandEvent::extraLong

**long 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.

### wxCommandEvent::wxCommandEvent

**void wxCommandEvent(WXTYPE *commandEventType*)**

Constructor. *commandEventType* may be one of the following:

- **wxEVENT\_TYPE\_BUTTON\_COMMAND**
- **wxEVENT\_TYPE\_CHECKBOX\_COMMAND**
- **wxEVENT\_TYPE\_CHOICE\_COMMAND**

- `wxEVENT_TYPE_LISTBOX_COMMAND`
- `wxEVENT_TYPE_LISTBOX_DCLICK_COMMAND`
- `wxEVENT_TYPE_TEXT_COMMAND`
- `wxEVENT_TYPE_TEXT_ENTER_COMMAND`
- `wxEVENT_TYPE_MULTITEXT_COMMAND`
- `wxEVENT_TYPE_MENU_COMMAND`
- `wxEVENT_TYPE_SLIDER_COMMAND`
- `wxEVENT_TYPE_RADIOBOX_COMMAND`
- `wxEVENT_TYPE_SET_FOCUS`
- `wxEVENT_TYPE_KILL_FOCUS`

### **`wxCommandEvent::Checked`**

**`Bool Checked(void)`**

Returns TRUE or FALSE for a checkbox selection event.

### **`wxCommandEvent::GetClientData`**

**`char * GetClientData(void)`**

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

### **`wxCommandEvent::GetSelection`**

**`int GetSelection(void)`**

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

### **`wxCommandEvent::GetString`**

**`char * GetString(void)`**

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

### **`wxCommandEvent::IsSelection`**

**`Bool IsSelection(void)`**

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

## **9.23. `wxCommandProcessor`: `wxObject`**

See also *Overview* (page 376)

`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.

**wxCommandProcessor::wxCommandProcessor****void 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.

**wxCommandProcessor::~~wxCommandProcessor****void ~wxCommandProcessor(void)**

Destructor.

**wxCommandProcessor::CanUndo****Bool CanUndo(void)**

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

**wxCommandProcessor::ClearCommands****void ClearCommands(void)**

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

**wxCommandProcessor::Do****Bool Do(void)**

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

**wxCommandProcessor::GetCommands****wxList& GetCommands(void)**

Returns the list of commands.

**wxCommandProcessor::GetMaxCommands****int GetMaxCommands(void)**

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

**wxCommandProcessor::GetEditMenu**

**wxMenu \* GetEditMenu(void)**

Returns the edit menu associated with the command processor.

**wxCommandProcessor::Initialize****void Initialize(void)**

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).

**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.

**wxCommandProcessor::Submit****Bool Submit(wxCommand \*command, Bool storeIt)**

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.

**wxCommandProcessor::Undo****Bool Undo(void)**

Undoes the command just executed.

**9.24. wxConnection: wxObject**

See also *Interprocess communications overview* (page 378)

A `wxConnection` object represents the connection between a client and a server. It can be created by making a connection using a `wxClient` (page 74) object, or by the acceptance of a connection by a `wxServer` (page 277) object. The bulk of a DDE (Dynamic Data Exchange) conversation (available in both Windows and UNIX) is controlled by calling members in a `wxConnection` object or by overriding its members.

An application should normally derive a new connection class from `wxConnection`, in order to override the communication event handlers to do something interesting.

See also *wxClient* (page 74), *wxServer* (page 277).

## **wxConnection::wxConnection**

**void wxConnection(void)**

**void wxConnection(char \*buffer, int size)**

Constructs a connection object. If no user-defined connection object is to be derived from *wxConnection*, 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 *wxServer::OnAcceptConnection* (page 278) and/or *wxClient::OnMakeConnection* (page 74) members should be replaced by functions which construct the new connection object. If the arguments of the *wxConnection* constructor are void, then a default buffer is associated with the connection. Otherwise, the programmer must provide a buffer and size of the buffer for the connection object to use in transactions.

## **wxConnection::Advise**

**Bool Advise(char \*item, char \*data, int size = -1, int 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 *wxConnection::OnAdvise* (page 92) member to be called. Returns TRUE if successful.

## **wxConnection::Execute**

**Bool Execute(char \*data, int size = -1, int 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 *wxConnection::Poke* (page 93) in that respect). Causes the server connection's *wxConnection::OnExecute* (page 93) member to be called. Returns TRUE if successful.

## **wxConnection::Disconnect**

**Bool Disconnect(void)**

Called by the client or server application to disconnect from the other program; it causes the *wxConnection::OnDisconnect* (page 93) 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.

## **wxConnection::OnAdvise**

**Bool OnAdvise(char \*topic, char \*item, char \*data, int size, int format)**

Message sent to the client application when the server notifies it of a change in the data associated with the given item.

**wxConnection::OnDisconnect****Bool OnDisconnect(void)**

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.

**wxConnection::OnExecute****Bool OnExecute(char \*topic, char \*data, int size, int 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.

**wxConnection::OnPoke****Bool OnPoke(char \*topic, char \*item, char \*data, int size, int format)**

Message sent to the server application when the client notifies it to accept the given data.

**wxConnection::OnRequest****char \* OnRequest(char \*topic, char \*item, int \*size, int format)**

Message sent to the server application when the client calls *wxConnection::Request* (page 94). The server should respond by returning a character string from **OnRequest**, or NULL to indicate no data.

**wxConnection::OnStartAdvise****Bool OnStartAdvise(char \*topic, char \*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.

**wxConnection::OnStopAdvise****Bool OnStopAdvise(char \*topic, char \*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.

**wxConnection::Poke****Bool Poke(char \*item, char \*data, int size = -1, int 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 `wxConnection::OnPoke` (page 93) member to be called. Returns TRUE if successful.

### **wxConnection::Request**

**char \* Request(char \*item, int \*size, int format = wxCF\_TEXT)**

Called by the client application to request data from the server. Causes the server connection's `wxConnection::OnRequest` (page 93) member to be called. Returns a character string (actually a pointer to the connection's buffer) if successful, NULL otherwise.

### **wxConnection::StartAdvise**

**Bool StartAdvise(char \*item)**

Called by the client application to ask if an advise loop can be started with the server. Causes the server connection's `wxConnection::OnStartAdvise` (page 93) member to be called. Returns TRUE if the server okays it, FALSE otherwise.

### **wxConnection::StopAdvise**

**Bool StopAdvise(char \*item)**

Called by the client application to ask if an advise loop can be stopped. Causes the server connection's `wxConnection::OnStopAdvise` (page 93) member to be called. Returns TRUE if the server okays it, FALSE otherwise.

## **9.25. wxCursor: wxBitmap**

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* (page 183) 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` (page 336) is also available for MS Windows use.

Run the *hello* demo program to see what stock cursors are available.

### **wxCursor::wxCursor**

**void wxCursor(void)**

Default constructor.

**void wxCursor(short bits[], int width, int height, int hotSpotX=-1, int hotSpotY=-1, char**

*\*maskBits=NULL)*

Construct a cursor by passing an array of bits (XView and Motif only). *maskBits* is used only under Motif.

If either *hotSpotX* or *hotSpotY* is -1, the hotspot will be the centre of the cursor image (values ignored under XView).

**void wxCursor(char \*cursorName, long flags, int hotSpotX=0, int hotSpotY=0)**

Construct a cursor by passing a string resource name or filename. Under Motif, *flags* defaults to `wxBITMAP_TYPE_XBM | wxBITMAP_DISCARD_COLOURMAP`. Under Windows, it defaults to `wxBITMAP_TYPE_CUR_RESOURCE | wxBITMAP_DISCARD_COLOURMAP`.

*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.

Under X, the permitted cursor types in the *flags* bitlist 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 `wx_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 `wx_setup.h`). Specify  
                                  *hotSpotX* and *hotSpotY*.

**void wxCursor(int id)**

Create a cursor by passing a stock cursor id. The following stock cursor ids may be used:

- `wxCURSOR_ARROW`
- `wxCURSOR_BULLSEYE`
- `wxCURSOR_CHAR`
- `wxCURSOR_CROSS`
- `wxCURSOR_HAND`
- `wxCURSOR_IBEAM`
- `wxCURSOR_LEFT_BUTTON`
- `wxCURSOR_MAGNIFIER`
- `wxCURSOR_MIDDLE_BUTTON`
- `wxCURSOR_NO_ENTRY`
- `wxCURSOR_PAINT_BRUSH`
- `wxCURSOR_PENCIL`
- `wxCURSOR_POINT_LEFT`
- `wxCURSOR_POINT_RIGHT`
- `wxCURSOR_QUESTION_ARROW`
- `wxCURSOR_RIGHT_BUTTON`
- `wxCURSOR_SIZENESW`
- `wxCURSOR_SIZENS`
- `wxCURSOR_SIZENWSE`



- wxCURSOR\_SIZEWE
- wxCURSOR\_SIZING
- wxCURSOR\_SPRAYCAN
- wxCURSOR\_WAIT
- wxCURSOR\_WATCH

### **wxCursor::~wxCursor**

#### **void ~wxCursor(void)**

Destroys the cursor. Unlike an icon, 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.

### **9.26. wxDatabase: wxObject**

See also *Overview* (page 394)

Every database object represents an ODBC connection. The connection may be closed and reopened.

### **wxDatabase::wxDatabase**

#### **void wxDatabase(void)**

Constructor. The constructor of the first wxDatabase instance of an application initializes the ODBC manager.

### **wxDatabase::~wxDatabase**

#### **void ~wxDatabase(void)**

Destructor. Resets and destroys any associated wxRecordSet instances.

The destructor of the last wxDatabase instance will deinitialize the ODBC manager.

### **wxDatabase::BeginTrans**

#### **Bool BeginTrans(void)**

Not implemented.

### **wxDatabase::Cancel**

#### **void Cancel(void)**

Not implemented.

**wxDatabase::CanTransact**

**Bool CanTransact(void)** Not implemented.

**wxDatabase::CanUpdate**

**Bool CanUpdate(void)**

Not implemented.

**wxDatabase::Close**

**Bool Close(void)**

Resets the statement handles of any associated wxRecordSet objects, and disconnects from the current data source.

**wxDatabase::CommitTrans**

**Bool CommitTrans(void)**

Commits previous transactions. Not implemented.

**wxDatabase::ErrorOccured**

**Bool ErrorOccured(void)**

Returns TRUE if the last action caused an error.

**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.

**wxDatabase::GetDatabaseName**

**char \* GetDatabaseName(void)**

Returns the name of the database associated with the current connection.

**wxDatabase::GetDataSource**

**char \* GetDataSource(void)**

Returns the name of the connected data source.

### **wxDatabase::GetErrorClass**

**char \* GetErrorClass(void)**

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.

### **wxDatabase::GetErrorCode**

**wxRETCODE GetErrorCode(void)**

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.

### **wxDatabase::GetErrorMessage**

**char \* GetErrorMessage(void)** Returns the last error message returned by the ODBC manager.

### **wxDatabase::GetErrorNumber**

**long GetErrorNumber(void)**

Returns the last native error. A native error is an ODBC driver dependent error number.

### **wxDatabase::GetHDBC**

**HDBC GetHDBC(void)**

Returns the current ODBC database handle.

### **wxDatabase::GetHENV**

**HENV GetHENV(void)**

Returns the ODBC environment handle.

### **wxDatabase::GetInfo**

**Bool GetInfo(long infoType, long \*buf)**

**Bool GetInfo(long infoType, char \*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.

**wxDatabase::GetPassword**

**char \* GetPassword(void)**

Returns the password of the current user.

**wxDatabase::GetUsername**

**char \* GetUsername(void)**

Returns the current username.

**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).

**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).

**wxDatabase::InWaitForDataSource**

**Bool InWaitForDataSource(void)**

Not implemented.

**wxDatabase::IsOpen****Bool IsOpen(void)**

Returns TRUE if a connection is open.

**wxDatabase::Open**

**Bool Open(char \*datasource, Bool exclusive = FALSE, Bool readOnly = TRUE, char \*username = "ODBC", char \*password = "")**

Connect to a data source. *datasource* contains the name of the ODBC data source. The parameters *exclusive* and *readOnly* are not used.

**wxDatabase::OnSetOptions**

**void OnSetOptions(wxRecordSet \*recordSet)**

Not implemented.

**wxDatabase::OnWaitForDataSource**

**void OnWaitForDataSource(Bool stillExecuting)**

Not implemented.

**wxDatabase::RollbackTrans**

**Bool RollbackTrans(void)**

Sends a rollback to the ODBC driver. Not implemented.

**wxDatabase::SetDataSource**

**void SetDataSource(char \*s)**

Sets the name of the data source. Not implemented.

**wxDatabase::SetLoginTimeout**

**void SetLoginTimeout(long seconds)**

Sets the time to wait for an user login. Not implemented.

**wxDatabase::SetPassword**

**void SetPassword(char \*s)**

Sets the password of the current user. Not implemented.

### **wxDatabase::SetSynchronousMode**

**void SetSynchronousMode**(**Bool** *synchronous*)

Toggles between synchronous and asynchronous mode. Currently only synchronous mode is supported, so this function has no effect.

### **wxDatabase::SetQueryTimeout**

**void SetQueryTimeout**(**long** *seconds*)

Sets the time to wait for a response to a query. Not implemented.

### **wxDatabase::SetUsername**

**void SetUsername**(**char** \**s*)

Sets the name of the current user. Not implemented.

## **9.27. wxDate: wxObject**

A class for manipulating dates.

### **wxDate::wxDate**

**void wxDate**(**void**)

Default constructor.

**void wxDate**(**wxDate&** *date*)

Copy constructor.

**void wxDate**(**const int** *month*, **const int** *day*, **const int** *year*)

Constructor.

*month* is a number from 1 to 12.

*day* is a number from 1 to 31.

*year* is a year, such as 1995, 2005.

**void wxDate**(**const 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.

**void wxDate**(**const char** \**date*)

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");
```

### **wxDate::~~wxDate**

**void ~wxDate(void)**

Destructor.

### **wxDate::AddMonths**

**wxDate& AddMonths(int months=1)**

Adds the given number of months to the date, returning a reference to 'this'.

### **wxDate::AddWeeks**

**wxDate& AddWeeks(int weeks=1)**

Adds the given number of weeks to the date, returning a reference to 'this'.

### **wxDate::AddYears**

**wxDate& AddYears(int years=1)**

Adds the given number of months to the date, returning a reference to 'this'.

### **wxDate::FormatDate**

**char \* FormatDate(const int type=-1) const**

Formats the date according to *type* if not -1, or according to the current display type if -1.

*type* can be -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.

The return value is a pointer to a statically-allocated character string.

### **wxDate::GetDay**

**int GetDay(void) const**

Returns the numeric day (in the range 1 to 31).

**wxDate::GetDayOfWeek**

**int GetDayOfWeek(void) const**

Returns the integer day of the week (in the range 1 to 7).

**wxDate::GetDayOfWeekName**

**char \* GetDayOfWeekName(void)**

Returns the name of the day of week. Do not delete the storage returned.

**wxDate::GetDayOfYear**

**long GetDayOfYear(void) const**

Returns the day of the year (from 1 to 365).

**wxDate::GetDaysInMonth**

**int GetDaysInMonth(void) const**

Returns the number of days in the month (in the range 1 to 31).

**wxDate::GetFirstDayOfMonth**

**int GetFirstDayOfMonth(void) const**

Returns the day of week that is first in the month (in the range 1 to 7).

**wxDate::GetJulianDate**

**long GetJulianDate(void) const**

Returns the Julian date.

**wxDate::GetMonth**

**int GetMonth(void) const**

Returns the month number (in the range 1 to 12).



**wxDate::GetMonthEnd****wxDate GetMonthEnd(void)**

Returns the date representing the last day of the month.

**wxDate::GetMonthName****char \* GetMonthName(void)**

Returns the name of the month. Do not delete the returned storage.

**wxDate::GetMonthStart****wxDate GetMonthStart(void)**

Returns the date representing the first day of the month.

**wxDate::GetWeekOfMonth****int GetWeekOfMonth(void)**

Returns the week of month (in the range 1 to 6).

**wxDate::GetWeekOfYear****int GetWeekOfYear(void)**

Returns the week of year (in the range 1 to 52).

**wxDate::GetYear****int GetYear(void) const**

Returns the year as an integer (such as '1995').

**wxDate::GetYearEnd****wxDate GetYearEnd(void)**

Returns the date representing the last day of the year.

**wxDate::GetYearStart****wxDate GetYearStart(void)**

Returns the date representing the first day of the year.

**wxDate::IsLeapYear****Bool IsLeapYear(void) const**

Returns TRUE if the year of this date is a leap year.

**wxDate::Set****wxDate& Set(void)**

Sets the date to current system date, returning a reference to 'this'.

**wxDate& Set(long julian)**

Sets the date to the given Julian date, returning a reference to 'this'.

**wxDate& 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.

**wxDate::SetFormat****void SetFormat(const int format)**

Sets the current format type.

*format* can be -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.

**wxDate::SetOption****int SetOption(const int option, const Bool enable=TRUE)**

Enables or disables an option for formatting. *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.

**wxDate::operator char \*****operator char \*(void)**

Conversion operator, to convert wxDate to char \* by calling FormatDate.

**wxDate::operator +****wxDate operator +(const long i)****wxDate operator +(const int i)**

Adds an integer number of days to the date, returning a date.

**wxDate::operator -****wxDate operator -(const long i)****wxDate operator -(const 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.

**wxDate::operator +=****wxDate& operator +=(const long i)**

Postfix operator: adds an integer number of days to the date, returning a reference to 'this' date.

**wxDate::operator -=****wxDate& operator -=(const long i)**

Postfix operator: subtracts an integer number of days from the date, returning a reference to 'this' date.

**wxDate::operator ++****wxDate& operator ++(void)**

Increments the date (postfix or prefix).

**wxDate::operator --**

**wxDate& operator --(void)**

Decrements the date (postfix or prefix).

**wxDate::operator <**

**friend Bool operator <(const wxDate& date1, const wxDate& date2)**

Function to compare two dates, returning TRUE if *date1* is earlier than *date2*.

**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*.

**wxDate::operator >**

**friend Bool operator >(const wxDate& date1, const wxDate& date2)**

Function to compare two dates, returning TRUE if *date1* is later than *date2*.

**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*.

**wxDate::operator ==**

**friend Bool operator ==(const wxDate& date1, const wxDate& date2)**

Function to compare two dates, returning TRUE if *date1* is equal to *date2*.

**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*.

**wxDate::operator <<**

**friend ostream& operator <<(ostream& os, const wxDate& date)**

Function to output a wxDate to an ostream.

## 9.28. wxDC: wxObject

See also *Overview* (page 383)

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 canvas has a device context 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.

### wxDC::wxDC

**void wxDC(void)**

Constructor.

### wxDC::~~wxDC

**void ~wxDC(void)**

Destructor.

### wxDC::BeginDrawing

**void BeginDrawing(void)**

Allows optimization of drawing code under MS Windows. Enclose drawing primitives between **BeginDrawing** and **EndDrawing** calls.

Drawing to a wxDialogBox 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.

### wxDC::Blit

**Bool Blit(float xdest, float ydest, float width, float height,  
wxDC \*source, float xsrc, float ysrc, int logical\_func, Bool transparent = FALSE)**

Copy from a source DC to this DC, specifying the destination coordinates, size of area to copy, source DC, source coordinates, and logical function (see *wxDC::SetLogicalFunction* (page 118)). See *wxMemoryDC* (page 205) for typical usage.

If *transparent* is TRUE and *source* is a wxMemoryDC with a transparent bitmap selected into it, the function will draw the source transparently. At present, the only way to create a transparent bitmap is to load a transparent XPM.

There is partial support for Blit in wxPostScriptDC, under X.

**wxDC::Clear****void Clear(void)**

Clears the device context using the current background brush.

**wxDC::CrossHair****void CrossHair(float x, float y)**

Displays a cross hair using the current pen. This is a vertical and horizontal line the height and width of the canvas, centred on the given point.

**wxDC::DestroyClippingRegion****void DestroyClippingRegion(void)**

Destroys the current clipping region so that none of the DC is clipped. See also *wxDC::SetClippingRegion* (page 117).

**wxDC::DeviceToLogicalX****float DeviceToLogicalX(int x)**

Convert device X coordinate to logical coordinate, using the current mapping mode.

**wxDC::DeviceToLogicalXRel****float DeviceToLogicalXRel(int x)**

Convert device X coordinate to relative logical coordinate, using the current mapping mode. Use this function for converting a width, for example.

**wxDC::DeviceToLogicalY****float DeviceToLogicalY(int y)**

Converts device Y coordinate to logical coordinate, using the current mapping mode.

**wxDC::DeviceToLogicalYRel****float DeviceToLogicalYRel(int y)**

Convert device Y coordinate to relative logical coordinate, using the current mapping mode. Use this function for converting a height, for example.

**wxDC::DrawArc****void DrawArc(float x1, float y1, float x2, float y2, float xc, float yc)**

Draws an arc, 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.

**wxDC::DrawEllipse****void DrawEllipse(float x, float y, float width, float 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.

**wxDC::DrawEllipticArc****void DrawEllipticArc(float x, float y, float width, float height, float start, float 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 canvas 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.

**wxDC::DrawIcon****void DrawIcon(wxIcon \*icon, float x, float 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 canvas.

**wxDC::DrawLine****void DrawLine(float x1, float y1, float x2, float y2)**

Draws a line from the first point to the second. The current pen is used for drawing the line.

**wxDC::DrawLines****void DrawLines(int n, wxPoint points[], float xoffset = 0, float yoffset = 0)****void DrawLines(wxList \*points, float xoffset = 0, float 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.

### **wxDC::DrawPolygon**

```
void DrawPolygon(int n, wxPoint points[], float xoffset = 0, float yoffset = 0,  
    int fill_style = wxODDEVEN_RULE)
```

```
void DrawPolygon(wxList *points, float xoffset = 0, float 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.

### **wxDC::DrawPoint**

```
void DrawPoint(float x, float y)
```

Draws a point using the current pen.

### **wxDC::DrawRectangle**

```
void DrawRectangle(float x, float y, float width, float 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.

### **wxDC::DrawRoundedRectangle**

```
void DrawRoundedRectangle(float x, float y, float width, float height, float 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.



**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(float x1, float y1, float x2, float y2, float x3, float y3)**

Draws a three-point spline using the current pen.

**wxDC::DrawText****void DrawText(char \*text, float x, float 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* (page 115) for how to get the dimensions of a text string, which can be used to position the text more precisely.

**wxDC::EndDoc****void EndDoc(void)**

Ends a document (only relevant when outputting to a printer).

**wxDC::EndDrawing****void EndDrawing(void)**

Allows optimization of drawing code under MS Windows. Enclose drawing primitives between **BeginDrawing** and **EndDrawing** calls.

**wxDC::EndPage****void EndPage(void)**

Ends a document page (only relevant when outputting to a printer).

**wxDC::FloodFill****void FloodFill(float x, float 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.

### **`wxDC::GetBackground`**

**`wxBrush * GetBackground(void)`**

Gets the brush used for painting the background (see `wxDC::SetBackground` (page 117)).

### **`wxDC::GetBrush`**

**`wxBrush * GetBrush(void)`**

Gets the current brush (see `wxDC::SetBrush` (page 117)).

### **`wxDC::GetCharHeight`**

**`float GetCharHeight(void)`**

Gets the character height of the currently set font.

### **`wxDC::GetCharWidth`**

**`float GetCharWidth(void)`**

Gets the average character width of the currently set font.

### **`wxCanvas::GetClippingBox`**

**`void GetClippingBox(float *x, float *y, float *width, float *height)`**

Gets the rectangle surrounding the current clipping region.

### **`wxDC::GetFont`**

**`wxFont * GetFont(void)`**

Gets the current font (see `wxDC::SetFont` (page 118)).

### **`wxDC::GetLogicalFunction`**

**`int GetLogicalFunction(void)`**

Gets the current logical function (see `wxDC::SetLogicalFunction` (page 118)).

**wxDC::GetMapMode****int GetMapMode(void)**

Gets the *mapping mode* for the device context (see *wxDC::SetMapMode* (page 118)).

**wxDC::GetOptimization****Bool GetOptimization(void)**

Returns TRUE if device context optimization is on. See *wxDC::SetOptimization* (page 119) for details.

**wxDC::GetPen****wxPen \* GetPen(void)**

Gets the current pen (see *wxDC::SetPen* (page 119)).

**wxDC::GetPixel****Bool GetPixel(float x, float 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.

**wxDC::GetSize****void GetSize(float \*width, float \*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:

```
float w, h;
dc.GetSize(&w, &h);
float scaleX=(float)(maxX/w);
float scaleY=(float)(maxY/h);
dc.SetUserScale(min(scaleX,scaleY),min(scaleX,scaleY));
```

**wxDC::GetTextBackground****wxColour& GetTextBackground(void)**

Gets the current text background colour (see *wxDC::SetTextBackground* (page 119)).

**wxDC::GetTextExtent**

```
void GetTextExtent(char *string, float *w, float *h,  
    float *descent = NULL, float *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* (page 158), *wxDC::SetFont* (page 118).

**wxDC::GetTextForeground**

```
wxColour& GetTextForeground(void)
```

Gets the current text foreground colour (see *wxDC::SetTextForeground* (page 119)).

**wxDC::IntDrawLine**

```
void IntDrawLine(int x1, int y1, int x2, int y2)
```

Draws a line from the first point to the second. The current pen is used for drawing the line.

**wxDC::IntDrawLines**

```
void IntDrawLines(int n, wxIntPoint points[], int xoffset = 0, int yoffset = 0)
```

Draw lines using an array of *points* of size *n*. The current pen is used for drawing the lines. The programmer is responsible for deleting the list of points.

**wxDC::LogicalToDeviceX**

```
int LogicalToDeviceX(float x)
```

Converts logical X coordinate to device coordinate, using the current mapping mode.

**wxDC::LogicalToDeviceXRel**

```
int LogicalToDeviceXRel(float x)
```

Converts logical X coordinate to relative device coordinate, using the current mapping mode. Use this for converting a width, for example.

**wxDC::LogicalToDeviceY****int LogicalToDeviceY(float y)**

Converts logical Y coordinate to device coordinate, using the current mapping mode.

**wxDC::LogicalToDeviceYRel****int LogicalToDeviceYRel(float y)**

Converts logical Y coordinate to relative device coordinate, using the current mapping mode. Use this for converting a height, for example.

**wxDC::MaxX****float MaxX(void)**

Gets the maximum horizontal extent used in drawing commands so far.

**wxDC::MaxY****float MaxY(void)**

Gets the maximum vertical extent used in drawing commands so far.

**wxDC::MinX****float MinX(void)**

Gets the minimum horizontal extent used in drawing commands so far.

**wxDC::MinY****float MinY(void)**

Gets the minimum vertical extent used in drawing commands so far.

**wxDC::Ok****Bool Ok(void)**

Returns TRUE if the DC is ok to use.

**wxDC::SetDeviceOrigin****void SetDeviceOrigin(float x, float 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.

### **wxDC::SetBackground**

**void SetBackground(wxBrush \*brush)**

Sets the current background brush for the DC. Do not delete the brush; it will be deleted automatically when the application terminates.

### **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.

### **wxDC::SetClippingRegion**

**void SetClippingRegion(float x, float y, float width, float height)**

Sets the clipping region for the DC. The clipping region is a rectangular area to which drawing is restricted. Possible uses for the clipping region are for clipping text or for speeding up canvas redraws when only a known area of the screen is damaged.

See also `wxDC::DestroyClippingRegion` (page 109).

### **wxDC::SetColourMap**

**void SetColourMap(wxColourMap \*colourMap)**

If this is a canvas DC or memory DC, assigns the given colourmap to the window or bitmap associated with the DC. If the argument is `NULL`, the current colourmap is selected out of the device context, and the original colourmap restored, allowing the current colourmap to be destroyed safely.

See `wxColourMap` (page 81) for further details.

### **wxDC::SetBrush**

**void SetBrush(wxBrush \*brush)**

Sets the current brush for the DC. The brush is not copied, so you should not delete the brush unless the DC pen has been set to another brush, or to `NULL`. Note that all pens and brushes are automatically deleted when the program is exited.

If the argument is `NULL`, 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* (page 51).

## **wxDC::SetFont**

**void SetFont(wxFont \*font)**

Sets the current font for the DC. The font is not copied, so you should not delete the font unless the DC pen has been set to another font, or to NULL.

If the argument is NULL, 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* (page 158).

## **wxDC::SetLogicalFunction**

**void SetLogicalFunction(int function)**

Sets the current logical function for the canvas. This determines how a source pixel (from a pen or brush colour, or source device context if using *wxDC::Blit* (page 108)) 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.

## **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` (page 120)) 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:

MM_TWIPS	Each logical unit is 1/20 of a point, or 1/1440 of an inch.
MM_POINTS	Each logical unit is a point, or 1/72 of an inch.
MM_METRIC	Each logical unit is 1 mm.
MM_LOMETRIC	Each logical unit is 1/10 of a mm.
MM_TEXT	Each logical unit is 1 pixel.

## **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.

## **wxDC::SetPen**

**void SetPen(wxPen \**pen*)**

Sets the current pen for the DC. The pen is not copied, so you should not delete the pen unless the DC pen has been set to another pen, or to NULL. Note that all pens and brushes are automatically deleted when the program is exited.

If the argument is NULL, the current pen is selected out of the device context, and the original pen restored, allowing the current pen to be destroyed safely.

## **wxDC::SetTextBackground**

**void SetTextBackground(wxColour \**colour*)**

Sets the current text background colour for the DC.

## **wxDC::SetTextForeground**



**void SetTextForeground(wxColour \*colour)**

Sets the current text foreground colour for the DC.

**wxDC::SetUserScale**

**void SetUserScale(float x\_scale, floaty\_scale)**

Sets the user scaling factor, useful for applications which require 'zooming'.

**wxDC::StartDoc**

**Bool StartDoc(char \*message)**

Starts a document (only relevant when outputting to a printer). Message is a message to show whilst printing.

**wxDC::StartPage**

**Bool StartPage(void)**

Starts a document page (only relevant when outputting to a printer).

## 9.29. wxDebugContext

See also *Overview* (page 398)

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 `USE_DEBUG_CONTEXT` to 0 in `wx_setup.h`

**wxDebugContext::Check**

**int Check(void)**

Checks the memory blocks for errors, starting from the currently set checkpoint. Returns the number of errors, so a value of zero represents success.

Returns -1 if an error was detected that prevents further checking.

**wxDebugContext::Dump**

**Bool Dump(void)**

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.

**wxDebugContext::GetCheckPrevious****Bool GetCheckPrevious(void)**

Returns TRUE if the memory allocator checks all previous memory blocks for errors. By default, this is FALSE since it slows down execution considerably.

**wxDebugContext::GetDebugMode****Bool GetDebugMode(void)**

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.

**wxDebugContext::GetLevel****int GetLevel(void)**

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.

**wxDebugContext::GetStream****ostream& GetStream(void)**

Returns the output stream associated with the debug context.

**wxDebugContext::GetStreamBuf****streambuf \* GetStreamBuf(void)**

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.

**wxDebugContext::HasStream****Bool HasStream(void)**

Returns TRUE if there is a stream currently associated with the debug context.

**wxDebugContext::PrintClasses****Bool PrintClasses(void)**

Prints a list of the classes declared in this application, giving derivation and whether instances of this class can be dynamically created.

**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.

If *detailed* is TRUE, the function will also print how many objects of each class have been allocated, and the space taken by these class instances.

**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.

If *all* is TRUE, the checkpoint is reset to include all memory allocations since the program started.

**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.

**wxDebugContext::SetFile****Bool SetFile(char \**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).

**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.

**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.

**wxDebugContext::SetStandardError****Bool SetStandardError(void)**

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.

**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).

Do not set this to NULL.

**9.30. wxDebugStreamBuf: streambuf**

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).

For example:

```
wxDebugStreamBuf streamBuf;  
ostream stream(&streamBuf);  
  
stream << "Hello world!" << endl;
```

**9.31. wxDialogBox: wxPanel**

See also *Overview* (page 385)

A dialog box is similar to a panel, in that it is a window which can be used for placing panel items, 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.

See also *wxPanel* (page 228) and *wxWindow* (page 319) for inherited member functions.

**wxDialogBox::wxDialogBox**

```
void wxDialogBox(wxWindow *parent, char *title, Bool modal=FALSE,  
int x=300, int y=300, int width=500, int height=500,  
    long style = wxDEFAULT_DIALOG_STYLE,  
    char *name = "dialogBox")
```

Constructor. The *parent* of the dialog box can be NULL, a frame or a dialog box.

If *title* is non-NULL, it is placed on the window frame.

If *modal* is TRUE, the dialog box will wait to be dismissed (using `Show(FALSE)`) before returning control to the calling program.

The *style* parameter may be a combination of the following, using the bitwise 'or' operator:

`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 (Motif only).  
`wxTHICK_FRAME`      Display a thick frame around the window (Motif only).  
`wxUSER_COLOURS`      Under Windows, overrides standard control processing to allow setting of the dialog box background colour.  
`wxVSCROLL`      Give the dialog box a vertical scrollbar (XView only).

Note that none take effect under Windows, only `wxVSCROLL` works under XView, and for Motif the MWM (the Motif Window Manager) should be running for any to work.

The *name* parameter is used to associate a name with the window, allowing the application user to set Motif resource values for individual dialog boxes.

## **wxDialogBox::~wxDialogBox**

```
void ~wxDialogBox(void)
```

Destructor. Deletes any panel items before deleting the physical window.

## **wxDialogBox::Centre**

```
void Centre(int direction = wxBOTH)
```

Centres the dialog box on the display. The parameter may be `wxHORIZONTAL`, `wxVERTICAL` or `wxBOTH`.

## **wxDialogBox::Create**

```
void Create(wxFrame *parent, char *title, Bool modal=FALSE,  
int x=300, int y=300, int width=500, int height=500,  
    long style = wxDEFAULT_DIALOG_STYLE,  
    char *name = "dialogBox")
```

Used for two-step dialog box construction. See `wxDialogBox::wxDialogBox` (page 123) for details.

### **wxDialogBox::GetTitle**

**char \* GetTitle(void)**

Gets a temporary pointer to the title of the dialog box.

### **wxDialogBox::Iconize**

**void Iconize(Bool iconize)**

If TRUE, iconizes the dialog box; if FALSE, shows and restores it. Note that in Windows, iconization has no effect since dialog boxes cannot be iconized. However, applications may need to explicitly restore dialog boxes under XView and Motif which have user-iconizable frames, and under Windows calling `Iconize(FALSE)` will bring the window to the front, as does `Show(TRUE)`.

### **wxDialogBox::Iconized**

**Bool Iconized(void)**

Returns TRUE if the dialog box is iconized. Always returns FALSE under Windows for the reasons given above.

### **wxDialogBox::IsModal**

**Bool IsModal(void)**

Returns TRUE if the dialog box is modal, FALSE otherwise.

### **wxDialogBox::OnCharHook**

**Bool OnCharHook(wxKeyEvent& ch)**

This member is called (under Windows only) to allow the window to intercept keyboard events before they are processed by child windows. The window receives this event from the default `wxApp::OnCharHook` (page 46) member function if the window (frame or dialog box) is active. The function should return TRUE to indicate the character has been processed, or FALSE to allow default processing. The default implementation for `wxWindow` returns FALSE, but the `wxDialogBox` implementation checks for `WXK_ESCAPE`, calls `OnClose` and if this returns TRUE, deletes the dialog box.

See also `wxKeyEvent` (page 193), `wxEvtHandler::OnChar` (page 151), `wxEvtHandler::OnCharHook` (page 152).

### **wxDialogBox::SetModal**

**void SetModal(Bool flag)**

Allows the programmer to specify whether the dialog box is modal (`wxDialogBox::Show` blocks control until the dialog is hidden) or modeless (control returns immediately).

**wxDialogBox::SetTitle**

**void SetTitle(char \* title)**

Sets the title of the dialog box.

**wxDialogBox::Show**

**Bool Show(Bool show)**

If *show* is TRUE, the dialog box is shown and brought to the front; otherwise the box is hidden. If *show* is FALSE and the dialog is modal, control is returned to the calling program.

### 9.32. wxDocChildFrame: wxFrame

See also *Document/view overview* (page 372)

The `wxDocChildFrame` class provides a default frame for displaying documents on separate windows.

The class is part of the document/view framework supported by `wxWindows`, and cooperates with the `wxView` (page 315), `wxDocument` (page 140), `wxDocManager` (page 128) and `wxDocTemplate` (page 135) classes.

See the example application in `samples/docview`.

**wxDocChildFrame::childDocument**

**wxDocument \* childDocument**

The document associated with the frame.

**wxDocChildFrame::childView**

**wxView \* childView**

The view associated with the frame.

**wxDocChildFrame::wxDocChildFrame**

**void wxDocChildFrame(wxDocument \*doc, wxView \*view, wxFrame \*parent, char \*title, int x, int y, int width, int height, long style, char \*name)**

Constructor.

**wxDocChildFrame::~~wxDocChildFrame****void ~wxDocChildFrame(void)**

Destructor.

**wxDocChildFrame::GetDocument****wxDocument \* GetDocument(void)**

Returns the document associated with this frame.

**wxDocChildFrame::GetView****wxView \* GetView(void)**

Returns the view associated with this frame.

**wxDocChildFrame::OnActivate****void OnActivate(Bool *active*)**

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.

**wxDocChildFrame::OnClose****Bool OnClose(void)**

Closes and deletes the current view and document.

**wxDocChildFrame::OnMenuCommand****void OnMenuCommand(int *cmd*)**

Passes menu commands to the parent frame (assumed to be a wxDocParentFrame).

**wxDocChildFrame::SetDocument****void SetDocument(wxDocument \**doc*)**

Sets the document for this frame.

**wxDocChildFrame::SetView****void SetView(wxView \**view*)**



Sets the view for this frame.

### 9.33. wxDocManager: wxEvtHandler

See also *Overview* (page 375)

The `wxDocManager` class is part of the document/view framework supported by `wxWindows`, and cooperates with the `wxView` (page 315), `wxDocument` (page 140) and `wxDocTemplate` (page 135) classes.

#### **wxDocManager::currentView**

**wxView \* currentView**

The currently active view.

#### **wxDocManager::defaultDocumentNameCounter**

**int defaultDocumentNameCounter**

Stores the integer to be used for the next default document name.

#### **wxDocManager::fileHistory**

**wxFileHistory \* fileHistory**

A pointer to an instance of `wxFileHistory` (page 156), which manages the history of recently-visited files on the File menu.

#### **wxDocManager::maxDocsOpen**

**int maxDocsOpen**

Stores the maximum number of documents that can be opened before existing documents are closed. By default, this is 10,000.

#### **wxDocManager::mnDocs**

**wxList mnDocs**

A list of all documents.

#### **wxDocManager::mnFlags**

**long mnFlags**

Stores the flags passed to the constructor.

**wxDocManager::mnTemplates****wxList mnTemplates**

A list of all document templates.

**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* (page 132) 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.

**wxDocManager::~~wxDocManager**

**void ~wxDocManager(void)**

Destructor.

**wxDocManager::ActivateView**

**void ActivateView(wxView \*doc, Bool activate, Bool deleting)**

Sets the current view.

**wxDocManager::AddDocument**

**void AddDocument(wxDocument \*doc)**

Adds the document to the list of documents.

**wxDocManager::AddFileToHistory**

**void AddFileToHistory(char \*filename)**

Adds a file to the file history list, if we have a pointer to an appropriate file menu.

**wxDocManager::AssociateTemplate**

**void AssociateTemplate(wxDocTemplate \*temp)**

Adds the template to the document manager's template list.

### **wxDocManager::CreateDocument**

**wxDocument \* CreateDocument(char \*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.

### **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.

### **wxDocManager::DisassociateTemplate**

**void DisassociateTemplate(wxDocTemplate \*temp)**

Removes the template from the list of templates.

### **wxDocManager::FileHistoryLoad**

**void FileHistoryLoad(char \*resourceFile, char \*sectionName)**

Loads the file history from a resource file, using the given section. This must be called explicitly by the application.

### **wxDocManager::FileHistorySave**

**void FileHistorySave(char \*resourceFile, char \*sectionName)**

Saves the file history into a resource file, using the given section. This must be called explicitly by the application.

**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.

**wxDocManager::FindTemplateForPath****wxDocTemplate \* FindTemplateForPath(char \*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.

**wxDocManager::GetCurrentDocument****wxDocument \* GetCurrentDocument(void)**

Returns the document associated with the currently active view (if any).

**wxDocManager::GetCurrentView****wxView \* GetCurrentView(void)**

Returns the currently active view

**wxDocManager::GetDocuments****wxList& GetDocuments(void)**

Returns a reference to the list of documents.

**wxDocManager::GetFileHistory****wxFileHistory \* GetFileHistory(void)**

Returns a pointer to file history.

**wxDocManager::GetMaxDocsOpen****int GetMaxDocsOpen(void)**

Returns the number of documents that can be open simultaneously.

**wxDocManager::GetNoHistoryFiles**

**int GetNoHistoryFiles(void)**

Returns the number of files currently stored in the file history.

**wxDocManager::Initialize****Bool Initialize(void)**

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.

**wxDocManager::MakeDefaultName****Bool MakeDefaultName(char \*buf)**

Copies a suitable default name into *buf*. This is implemented by appending an integer counter to the string `unnamed` and incrementing the counter.

**wxDocManager::OnCreateFileHistory****wxFileHistory \* OnCreateFileHistory(void)**

A hook to allow a derived class to create a different type of file history. Called from *Initialize* (page 132).

**wxDocManager::OnFileClose****void OnFileClose(void)**

Closes and deletes the currently active document.

**wxDocManager::OnFileNew****void OnFileNew(void)**

Creates a document from a list of templates (if more than one template).

**wxDocManager::OnFileOpen****void OnFileOpen(void)**

Creates a new document and reads in the selected file.

### **wxDocManager::OnFileSave**

**void OnFileSave(void)**

Saves the current document by calling wxDocument::Save for the current document.

### **wxDocManager::OnFileSaveAs**

**void OnFileSaveAs(void)**

Calls wxDocument::SaveAs for the current document.

### **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.

### **wxDocManager::RemoveDocument**

**void RemoveDocument(wxDocument \*doc)**

Removes the document from the list of documents.

### **wxDocManager::SelectDocumentPath**

**wxDocTemplate \* SelectDocumentPath(wxDocTemplate \*\*templates, int noTemplates, char \*path, char \*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.

**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`.

**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.

**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.

**9.34. wxDocParentFrame: wxFrame**

See also *Document/view overview* (page 372)

The `wxDocParentFrame` class provides a default top-level frame for applications using the document/view framework.

It cooperates with the `wxView` (page 315), `wxDocument` (page 140), `wxDocManager` (page 128) and `wxDocTemplates` (page 135) classes.

See the example application in `samples/docview`.

**wxDocParentFrame::wxDocParentFrame****void wxDocParentFrame(wxFrame \*parent, char \*title, int x, int y, int width, int height, long style, char \*name)**

Constructor.

**wxDocParentFrame::~~wxDocParentFrame****void ~wxDocParentFrame(void)**

Destructor.

## **wxDocParentFrame::OnClose**

### **Bool OnClose(void)**

Deletes all views and documents. If no user input cancelled the operation, the function returns TRUE 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.

```
Bool wxDocParentFrame::OnClose(void)
{
    // Delete all views and documents
    wxNode *node = docManager->GetDocuments().First();
    while (node)
    {
        wxDocument *doc = (wxDocument *)node->Data();
        wxNode *next = node->Next();

        if (!doc->Close())
            return FALSE;

        // Implicitly deletes the document when the last
        // view is removed (deleted)
        doc->DeleteAllViews();

        // Check document is deleted
        if (docManager->GetDocuments().Member(doc))
            delete doc;

        // This assumes that documents are not connected in
        // any way, i.e. deleting one document does NOT
        // delete another.
        node = next;
    }
    return TRUE;
}
```

## **wxDocParentFrame::OnMenuCommand**

### **void OnMenuCommand(int cmd)**

Processes the wxID\_EXIT and wxID\_FILEN (file history) commands. Other commands are routed to wxDocManager::OnMenuCommand.

## **9.35. wxDocTemplate: wxObject**

See also *Overview* (page 374)

The wxDocTemplate class is used to model the relationship between a document class and a view class.

## **wxDocTemplate::tDefaultExt**



**char \* tDefaultExt**

The default extension for files of this type.

**wxDocTemplate::tDescription**

**char \* tDescription**

A short description of this template.

**wxDocTemplate::tDirectory**

**char \* tDirectory**

The default directory for files of this type.

**wxDocTemplate::tDocClassInfo**

**wxClassInfo \* tDocClassInfo**

Run-time class information that allows document instances to be constructed dynamically.

**wxDocTemplate::tDocTypeName**

**char \* tDocTypeName**

The named type of the document associated with this template.

**wxDocTemplate::tDocumentManager**

**wxDocTemplate \* tDocumentManager**

A pointer to the document manager for which this template was created.

**wxDocTemplate::tFileFilter**

**char \* tFileFilter**

The file filter (such as \*.txt) to be used in file selector dialogs.

**wxDocTemplate::tFlags**

**long tFlags**

The flags passed to the constructor.

**wxDocTemplate::tViewClassInfo****wxClassInfo \* tViewClassInfo**

Run-time class information that allows view instances to be constructed dynamically.

**wxDocTemplate::tViewTypeName****char \* tViewTypeName**

The named type of the view associated with this template.

**wxDocTemplate::wxDocTemplate****void wxDocTemplate(wxDocManager \*manager, char \*descr, char \*filter, char \*dir, char \*ext, char \*docTypeName, char \*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.

**wxDocTemplate::~~wxDocTemplate****void ~wxDocTemplate(void)**

Destructor.

**wxDocTemplate::CreateDocument****wxDocument \* CreateDocument(char \*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.

**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.

**wxDocTemplate::GetDefaultExtension****char \* GetDefaultExtension(void)**

Returns the default file extension for the document data, as passed to the document template constructor.

**wxDocTemplate::GetDescription****char \* GetDescription(void)**

Returns the text description of this template, as passed to the document template constructor.

**wxDocTemplate::GetDirectory****char \* GetDirectory(void)**

Returns the default directory, as passed to the document template constructor.

**wxDocTemplate::GetDocumentManager****wxDocManager \* GetDocumentManager(void)**

Returns a pointer to the document manager instance for which this template was created.

**wxDocTemplate::GetDocumentName****char \* GetDocumentName(void)**

Returns the document type name, as passed to the document template constructor.

**wxDocTemplate::GetFileFilter****char \* GetFileFilter(void)**

Returns the file filter, as passed to the document template constructor.

**wxDocTemplate::GetFlags****long GetFlags(void)**

Returns the flags, as passed to the document template constructor.

**wxDocTemplate::GetViewName****char \* GetViewName(void)**

Returns the view type name, as passed to the document template constructor.

**wxDocTemplate::IsVisible****Bool IsVisible(void)**

Returns TRUE if the document template can be shown in user dialogs, FALSE otherwise.

**wxDocTemplate::SetDefaultExtension****void SetDefaultExtension(char \*ext)**

Sets the default file extension.

**wxDocTemplate::SetDescription****void SetDescription(char \*descr)**

Sets the template description.

**wxDocTemplate::SetDirectory****void SetDirectory(char \*dir)**

Sets the default directory.

### **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.

### **wxDocTemplate::SetFileFilter**

**void SetFileFilter(char \*filter)**

Sets the file filter.

### **wxDocTemplate::SetFlags**

**void SetFlags(long flags)**

Sets the internal document template flags (see the constructor description for more details).

## **9.36. wxDocument: wxEvtHandler**

See also *Overview* (page 373)

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* (page 315), *wxDocTemplate* (page 135) and *wxDocManager* (page 128) classes.

### **wxDocument::documentFile**

**char \* documentFile**

Filename associated with this document (NULL if none).

### **wxDocument::documentModified**

**Bool documentModified**

TRUE if the document has been modified, FALSE otherwise.

### **wxDocument::documentTemplate**

**wxDocTemplate \* documentTemplate**

A pointer to the template from which this document was created.

**wxDocument::documentTitle****char \* documentTitle**

Document title (may be NULL). The document title is used for an associated frame (if any), and is usually constructed by the framework from the filename.

**wxDocument::documentTypeName****char \* 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.

**wxDocument::documentViews****wxList documentViews**

List of wxView instances associated with this document.

**wxDocument::wxDocument****void wxDocument(void)**

Constructor. Define your own default constructor to initialize application-specific data.

**wxDocument::~~wxDocument****void ~wxDocument(void)**

Destructor. Removes itself from the document manager.

**wxDocument::AddView****Bool AddView(wxView \*view)**

If the view is not already in the list of views, adds the view and calls OnChangedViewList.

**wxDocument::Close****Bool Close(void)**

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.

**wxDocument::DeleteAllViews****Bool DeleteAllViews(void)**

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.

**wxDocument::GetCommandProcessor****wxCommandProcessor \* GetCommandProcessor(void)**

Returns a pointer to the command processor associated with this document.

See *wxCommandProcessor* (page 89).

**wxDocument::GetDocumentTemplate****wxDocTemplate \* GetDocumentTemplate(void)**

Gets a pointer to the template that created the document.

**wxDocument::GetDocumentManager****wxDocManager \* GetDocumentManager(void)**

Gets a pointer to the associated document manager.

**wxDocument::GetDocumentName****char \* GetDocumentName(void)**

Gets the document type name for this document. See the comment for *documentTypeName* (page 141).

**wxDocument::GetDocumentWindow****wxWindow \* GetDocumentWindow(void)**

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.

**wxDocument::GetFilename****char \* GetFilename(void)**

Gets the filename associated with this document, or NULL if none is associated.

### **wxDocument::GetFirstView**

**wxView \* GetFirstView(void)**

A convenience function to get the first view for a document, because in many cases a document will only have a single view.

### **wxDocument::GetPrintableName**

**void GetPrintableName(char \*name)**

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**.

### **wxDocument::GetTitle**

**char \* GetTitle(void)**

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.

### **wxDocument::IsModified**

**Bool IsModified(void)**

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* (page 143).

### **wxDocument::LoadObject**

**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.

### **wxDocument::Modify**

**void IsModify(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* (page 143).

### **wxDocument::OnChangedViewList**

**void OnChangedViewList(void)**

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).

### **wxDocument::OnCloseDocument**

**Bool OnCloseDocument(void)**

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`.

### **wxDocument::OnCreate**

**Bool OnCreate(const char \*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.

### **wxDocument::OnCreateCommandProcessor**

**wxCommandProcessor \* OnCreateCommandProcessor(void)**

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* (page 89).

### **wxDocument::OnNewDocument**

**Bool OnNewDocument(void)**

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.

### **wxDocument::OnOpenDocument**

**Bool OnOpenDocument(char \*filename)**

Constructs an input file stream for the given filename (which must not be `NULL`), 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.

### **wxDocument::OnSaveDocument**

**Bool OnSaveDocument(char \*filename)**

Constructs an output file stream for the given filename (which must not be NULL), and calls SaveObject. If SaveObject returns TRUE, the document is set to unmodified; otherwise, an error message box is displayed.

### **wxDocument::OnSaveModified**

**Bool OnSaveModified(void)**

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.

### **wxDocument::RemoveView**

**Bool RemoveView(wxView \*view)**

Removes the view from the document's list of views, and calls OnChangedViewList.

### **wxDocument::Save**

**Bool Save(void)**

Saves the document by calling OnSaveDocument if there is an associated filename, or SaveAs if there is no filename.

### **wxDocument::SaveAs**

**Bool SaveAs(void)**

Prompts the user for a file to save to, and then calls OnSaveDocument.

### **wxDocument::SaveObject**

**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.

### **wxDocument::SetCommandProcessor**

**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* (page 89).

### **wxDocument::SetDocumentName**

**void SetDocumentName(char \*name)**

Sets the document type name for this document. See the comment for *documentTypeName* (page 141).

### **wxDocument::SetDocumentTemplate**

**void SetDocumentTemplate(wxDocTemplate \*templ)**

Sets the pointer to the template that created the document. Should only be called by the framework.

### **wxDocument::SetFilename**

**void SetFilename(char \*filename)**

Sets the filename for this document. Usually called by the framework.

### **wxDocument::SetTitle**

**void SetTitle(char \*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.

## **9.37. wxEnhDialogBox: wxDialogBox**

*wxEnhDialogBox* is derived from *wxDialogBox* (page 123).

The purpose of the *wxEnhDialogBox* class is to make it easy to provide a common look for all dialog boxes of an application. The *wxEnhDialogBox* separates the dialog box into four areas:

- the pin area
- the user area
- the command area
- the status area

For now, these panels are tiled vertically, but in future there may a style flag to allow placement of the command area to the right of the dialog, as is common in Windows applications.

The pin area is borrowed from the pushpin metaphor of *XView*, and can be disabled via a

compilation flag. Again, a flag style is perhaps more judicious and may be implemented in future.

The user area is left free for the application programmer.

The command area contains command buttons which are centered automatically by the *wxEnhDialogBox::Fit* (page 148) method.

The status area provides a way to display status messages, as in XView.

All areas can have distinct fonts sets, currently controlled by a compilation flag. The pushpin can be replaced by a Cancel button (automatically created) if `WANT_CANCEL_BUTTON` is defined when compiling.

Warning: this class is pending revision and debugging. You may find it does not work as advertised.

## **wxEnhDialogBox::wxEnhDialogBox**

```
void wxEnhDialogBox(wxFrame *frame, char *title,
Bool modal = FALSE, wxFunction fun = NULL, int space = -1,
int x = 0, int y = 0, int width = 10, int height = 10,
long style = wxENH_DEFAULT, char *name = "Shell")
```

Constructor. *fun* is called when the user dismiss the window by using the pin or the cancel button. If *space* is greater than zero, it is used as panel horizontal spacing for the command area.

The style parameter may be a combination of the following, using the bitwise 'or' operator.

<code>wxBOTTOM_COMMANDS</code>	Command buttons are on bottom of the dialog.
<code>wxCANCEL_BUTTON_FIRST</code>	The cancel button is the first button.
<code>wxCANCEL_BUTTON_LAST</code>	The cancel button is the last button.
<code>wxCANCEL_BUTTON_SECOND</code>	The cancel button is the second button.
<code>wxCAPTION</code>	Gives a caption to the dialog box.
<code>wxENH_DEFAULT</code>	Equivalent to a combination of <code>wxCAPTION</code> , <code>wxBOTTOM_COMMANDS</code> , <code>wxSTATUS_FOOTER</code> and <code>wxNO_CANCEL_BUTTON</code> .
<code>wxNO_STATUS_FOOTER</code>	No status line is displayed.
<code>wxNO_CANCEL</code>	No cancel button is displayed.
<code>wxRIGHT_COMMANDS</code>	Command buttons are on the right hand side of the dialog.
<code>wxSTATUS_FOOTER</code>	A status line is displayed at the bottom of the dialog.

## **wxEnhDialogBox::~~wxEnhDialogBox**

```
void ~wxEnhDialogBox(void)
```

Destructor.

## **wxEnhDialogBox::userPanel**

```
wxPanel * userPanel
```

User application items must be created in this panel.

**wxEnhDialogBox::SetStatus****void SetStatus(char \*label=NULL)**

Display text in the status area.

**wxEnhDialogBox::AddCmd****wxButton \* AddCmd(char \*label, wxFunction fun=NULL,  
int tag = 0)****wxButton \* AddCmd(wxBitmap \*bitmap, wxFunction fun=NULL,  
int tag = 0)**

Adds a command button in the command area, and returns its identifier. The client data part of the button is initialized with *tag*. Buttons are arranged horizontally, from left to right. If `WANT_CANCEL_BUTTON` is defined at compile time, a first button is automatically created, with the label "Cancel".

**wxEnhDialogBox::GetCmd****wxButton \* GetCmd(int n)**

Returns the identifier of the *n*th wxButton in the command area (starting at 0).

**wxEnhDialogBox::SetPin****void SetPin(Bool flag)**

Set the pushpin to the given state. The state of the pushpin controls the way the `::Show()` method works. Usually, setting the pin to `TRUE` indicates an error. Please note that this works even if you have compiled with `WANT_CANCEL_BUTTON`.

**wxEnhDialogBox::Show****Bool Show(Bool show, Bool flag = FALSE)**

Dismiss or popup the dialog box. If *show* is `FALSE`, the dialog box is dismissed only if the pushpin current state is `FALSE`. If *show* is `TRUE`, the dialog box is popped, and the pushpin is initialized in the state *flag*.

**wxEnhDialogBox::Fit****void Fit(void)**

Fits the dialog box to its contents, and centres command buttons in the command area. The status area is created when `Fit` is called, so do not call `SetStatus` before fitting.

### 9.38. wxEvent: wxObject

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 now an abstract base class for events such as *wxCommandEvent* (page 88) (for panel item commands) and *wxMouseEvent* (page 214) (for mouse events on windows).

#### **wxEvent::wxEvent**

**void wxEvent(void)**

Constructor. Should not need to be used by an application.

#### **wxEvent::~~wxEvent**

**void ~wxEvent(void)**

Destructor. Should not need to be used by an application.

#### **wxEvent::eventClass**

**WXTYPE eventClass**

The C++ class of the event, such as `wxTYPE_COMMAND_EVENT`. A single class may have many 'types'; it would be tedious to define a new C++ class for each type of similar event.

#### **wxEvent::eventHandle**

**char \* eventHandle**

Handle of an underlying windowing system event handle, such as `XEvent`. Not guaranteed to be instantiated.

#### **wxEvent::eventObject**

**wxObject \* eventObject**

The object (usually a window) that the event was generated from, or should be sent to.

#### **wxEvent::eventType**

**WXTYPE eventType**

The type of the event, such as `wxEVENT_TYPE_BUTTON_COMMAND`.

#### **wxEvent::GetEventClass**

**WXTYPE GetEventClass(void)**

Returns the identifier of the given event class, such as wxTYPE\_MOUSE\_EVENT.

**wxEvent::GetEventObject****wxObject \* GetEventObject(void)**

Returns the object associated with the event, if any.

**wxEvent::GetEventType****WXTYPE GetEventType(void)**

Returns the identifier of the given event type, such as wxEVENT\_TYPE\_BUTTON\_COMMAND.

**wxEvent::GetObjectType****WXTYPE GetObjectType(void)**

Returns the type of the object associated with the event, such as wxTYPE\_BUTTON.

**wxEvent::ReadEvent****pure virtual Bool ReadEvent(istream& stream)**

Reads the event from the given input stream.

**wxEvent::WriteEvent****pure virtual Bool WriteEvent(ostream& stream)**

Writes the event to the given output stream.

**9.39. wxEvtHandler: wxObject**

See also *Event handling overview* (page 390)

A class that can handle events from the windowing system. wxWindow (and therefore all window classes) are derived from this class.

**wxEvtHandler::nextHandler****wxEvtHandler \* nextHandler**

Protected member variable pointing the next event handler in the chain.

**wxEvtHandler::previousHandler****wxEvtHandler \* previousHandler**

Protected member variable pointing the previous event handler in the chain.

**wxEvtHandler::wxEvtHandler****void wxEvtHandler(void)**

Constructor.

**wxEvtHandler::~~wxEvtHandler****void ~wxEvtHandler(void)**

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.

**wxEvtHandler::GetClientData****char \* GetClientData(void)**

Gets user-supplied client data. 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.

**wxEvtHandler::GetNextHandler****wxEvtHandler \* GetNextHandler(void)**

Gets the pointer to the next handler in the chain.

**wxEvtHandler::GetPreviousHandler****wxEvtHandler \* GetPreviousHandler(void)**

Gets the pointer to the previous handler in the chain.

**wxEvtHandler::OnActivate****void OnActivate(Bool active)**

Called when a window is activated or deactivated (MS Windows only). If the window is being activated, *active* is TRUE, else it is FALSE.

**wxEvtHandler::OnChar**



**void OnChar(wxKeyEvent& ch)**

Sent to the window when the user has pressed a key. See *wxKeyEvent* (page 193) for details.

Note that the ASCII values do not have explicit key codes: they are passed as ASCII values.

See also *wxEvtHandler::OnEvent* (page 153) for mouse event notification. **OnChar** is currently applicable to canvas and panel subwindows only. On some platforms, it may be implemented for text subwindows (not XView).

**wxEvtHandler::OnCharHook****Bool OnCharHook(wxKeyEvent& ch)**

This member is called (under Windows only) to allow the window to intercept keyboard events before they are processed by child windows. The window receives this event from the default *wxApp::OnCharHook* (page 46) member function if the window (frame or dialog box) is active. The function should return TRUE to indicate the character has been processed, or FALSE to allow default processing. The default implementation for *wxWindow* returns FALSE, but the *wxDialogBox* implementation checks for *WXK\_ESCAPE* and tries to close the dialog.

See also *wxKeyEvent* (page 193), *wxEvtHandler::OnChar* (page 151), *wxDialogBox::OnCharHook* (page 125).

**wxEvtHandler::OnCommand****void OnCommand(wxWindow &win, wxCommandEvent &event)**

This member is called for panel items that do not have a callback function of their own.

**wxEvtHandler::OnClose**

Sent to the frame when the user has tried to close a managed window (i.e., a frame or dialog box) using the window manager (X) or system menu (Windows). If TRUE is returned by *OnClose*, the frame will be deleted by the system, otherwise the attempt will be ignored. Derive your own class to handle this message; the default handler returns FALSE.

**Bool OnClose(void)****wxEvtHandler::OnDefaultAction****void OnDefaultAction(wxItem \*item)**

Called when the user initiates the default action for a panel or dialog box, for example by double clicking on a listbox. *item* is the panel item which caused the default action. See *wxPanel::OnDefaultAction* (page 231).

**wxEvtHandler::OnDropFiles****void OnDropFiles(int n, char \*files[], int x, int y)**

Under Windows, called when files have been dragged from the file manager to the window. *files* is an array of *n* strings, and *x* and *y* give the mouse position where the drop occurred. The window must have previously been enabled for dropping by calling `wxWindow::DragAcceptFiles` (page 320).

### **wxEvtHandler::OnEvent**

**void OnEvent(wxMouseEvent& event)**

Sent to the window when the user has initiated an event with the mouse. Derive your own class to handle this message. So far, only relevant to the `wxCanvas` class. See `wxEvtHandler::OnChar` (page 151) for character events, and also `wxMouseEvent` (page 214) for how to access event information.

### **wxEvtHandler::OnItemEvent**

**void OnItemEvent(wxItem \* item, wxMouseEvent & event)**

Called in user-interface edit mode when a panel item receives a mouse event. The default implementation manages panel item dragging and sizing.

See `wxWindow::SetUserEditMode` (page 327).

### **wxEvtHandler::OnItemLeftClick**

**void OnItemLeftClick(wxItem \*item, int x, int y, int keys)**

Called in user-interface edit mode when the user left-clicks on a panel item. The coordinates (relative to the item) and a flag indicating shift and control key status are passed. *keys* is a bit list of `wxKEY_SHIFT` and `wxKEY_CTRL`.

See also `wxWindow::SetUserEditMode` (page 327).

### **wxEvtHandler::OnItemMove**

**void OnItemMove(wxItem \* item, int x, int y)**

Called in user-interface edit mode when the item has been moved by the user.

See also `wxWindow::SetUserEditMode` (page 327).

### **wxEvtHandler::OnItemRightClick**

**void OnItemRightClick(wxItem \*item, int x, int y, int keys)**

Called in user-interface edit mode when the user right-clicks on a panel item. The coordinates (relative to the item) and a flag indicating shift and control key status are passed. *keys* is a bit list of `wxKEY_SHIFT` and `wxKEY_CTRL`.

See also `wxWindow::SetUserEditMode` (page 327).

### **wxEvtHandler::OnItemSelect**

**void OnItemSelect**(`wxItem *item`, `Bool select`)

Called when a window is selected or deselected. Currently applies only to panel items in user-interface edit mode.

### **wxEvtHandler::OnItemSize**

**void OnItemSize**(`wxItem * item`, `int width`, `int height`)

Called in user-interface edit mode when the item has been resized by the user.

See also `wxWindow::SetUserEditMode` (page 327).

### **wxEvtHandler::OnLeftClick**

**void OnLeftClick**(`int x`, `int y`, `int keys`)

Called in user-interface edit mode when the user left-clicks on the panel background. The coordinates and a flag indicating shift and control key status are passed. `keys` is a bit list of `wxKEY_SHIFT` and `wxKEY_CTRL`.

See also `wxWindow::SetUserEditMode` (page 327).

### **wxEvtHandler::OnRightClick**

**void OnRightClick**(`int x`, `int y`, `int keys`)

Called in user-interface edit mode when the user right-clicks on the panel background. The coordinates and a flag indicating shift and control key status are passed. `keys` is a bit list of `wxKEY_SHIFT` and `wxKEY_CTRL`.

See also `wxWindow::SetUserEditMode` (page 327).

### **wxEvtHandler::OnKillFocus**

**void OnKillFocus**(`void`)

Called when a window's focus is being killed. There are many exceptions to this rule so be careful when relying on it.

### **wxEvtHandler::OnMenuCommand**

**void OnMenuCommand**(`int id`)

Sent to a frame window's event handler when an item on the window's menu has been chosen.

Derive your own frame class to handle this message. See *wxFrame::OnMenuCommand* (page 176).

### **wxEvtHandler::OnMenuSelect**

**void OnMenuSelect(int id)**

Sent to a frame's event handler when an item on the frame's menu has been selected (i.e. the cursor is on the item, but the left button has not been released). Derive your own frame class to handle this message. See *wxFrame::OnMenuSelect* (page 177).

### **wxEvtHandler::OnMove**

**void OnMove(int x, int y)**

Called when a window is moved. Not currently implemented.

### **wxEvtHandler::OnPaint**

**void OnPaint(void)**

Sent to the event handler when the window must be refreshed. Derive your own class to handle this message. So far, only relevant to the *wxCanvas* and *wxPanel* classes.

### **wxEvtHandler::OnScroll**

**void OnScroll(wxCommandEvent& event)**

Override this function to intercept scroll events. Only implemented for the *wxCanvas* class. See *wxCanvas::OnScroll* (page 64).

### **wxEvtHandler::OnSelect**

**void OnSelect(Bool select)**

Called when a window is selected or deselected. Currently applies only to panel items in user-interface edit mode.

### **wxEvtHandler::OnSetFocus**

**void OnSetFocus(void)**

Called when a window's focus is being set. There are many exceptions to this rule so be careful when relying on it.

### **wxEvtHandler::OnSize**

**void OnSize(int x, int y)**

Sent to the event handler when the window has been resized. You may wish to use this for frames to resize their child windows as appropriate. Derive your own class to handle this message. Note that the size passed is of the whole window: call **GetClientSize** for the area which may be used by the application.

**wxEvtHandler::SetClientData****void SetClientData(char \*data)**

Sets user-supplied client data. 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.

**wxEvtHandler::SetNextHandler****void SetNextHandler(wxEvtHandler \*handler)**

Sets the pointer to the next handler.

**wxEvtHandler::SetPreviousHandler****void SetPreviousHandler(wxEvtHandler \*handler)**

Sets the pointer to the previous handler.

**9.40. wxFileHistory: wxObject**

See also *Overview* (page 376)

The wxFileHistory encapsulates a user interface convenience, the list of most recently visited files as shown on a menu (usually the File menu).

**wxFileHistory::fileHistory****char \*\* fileHistory**

A character array of strings corresponding to the most recently opened files.

**wxFileHistory::fileHistoryN****int fileHistoryN**

The number of files stored in the history array.

**wxFileHistory::fileMaxFiles****int fileMaxFiles**

The maximum number of files to be stored and displayed on the menu.

### **wxFileHistory::fileMenu**

**wxMenu \* fileMenu**

The file menu used to display the file history list (if enabled).

### **wxFileHistory::wxFileHistory**

**void wxFileHistory(int maxFiles = 9)**

Constructor. Pass the maximum number of files that should be stored and displayed.

### **wxFileHistory::~~wxFileHistory**

**void ~wxFileHistory(void)**

Destructor.

### **wxFileHistory::AddFileToHistory**

**void AddFileToHistory(char \*filename)**

Adds a file to the file history list, if the object has a pointer to an appropriate file menu.

### **wxFileHistory::FileHistoryLoad**

**void FileHistoryLoad(char \*resourceFile, char \*sectionName)**

Loads the file history from a resource file, using the given section. This must be called explicitly by the application.

### **wxFileHistory::FileHistorySave**

**void FileHistorySave(char \*resourceFile, char \*sectionName)**

Saves the file history into a resource file, using the given section. This must be called explicitly by the application.

### **wxFileHistory::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.

**wxFileHistory::GetMaxFiles****int GetMaxFiles(void)**

Returns the maximum number of files that can be stored.

**wxFileHistory::GetNoHistoryFiles****int GetNoHistoryFiles(void)**

Returns the number of files currently stored in the file history.

**9.41. wxFont: wxObject**

See also *Overview* (page 381)

A font is an object which determines the appearance of text, primarily when drawing text to a canvas or device context.

**wxFont::wxFont****void wxFont(void)****void wxFont(int point\_size, int family, int style, int weight, Bool underline = FALSE, const char \*face\_name = NULL)**

Creates a font object. These are the arguments:

point_size	This is the standard way of referring to text size.
family	A 'family' of related font faces, giving a measure of independence from the actual typefaces available on a computer. Supported families are: <b>wxDEFAULT</b> , <b>wxDECORATIVE</b> , <b>wxROMAN</b> , <b>wxSCRIPT</b> , <b>wxSWISS</b> , <b>wxMODERN</b> . <b>wxMODERN</b> is a fixed pitch font; the others are either fixed or variable pitch.
style	The value can be <b>wxNORMAL</b> , <b>wxSLANT</b> or <b>wxITALIC</b> .
weight	The value can be <b>wxNORMAL</b> , <b>wxLIGHT</b> or <b>wxBOLD</b> .
underlining	The value can be TRUE or FALSE (MS Windows only).
face_name	An optional string specifying the actual typeface to be used. If NULL, a default typeface will chosen based on the family.

If the desired font does not exist, the closest match will be chosen. Under XView, this may result in a number of XView warnings during the matching process; these should be ignored, and will only occur the first time wxWindows attempts to use an absent font in a given size. wxWindows under Motif does the same thing, but silently. Under MS Windows, only scaleable TrueType fonts are used.

Underlining only works under MS Windows at present.

See also *wxDC::SetFont* (page 118), *wxDC::DrawText* (page 112) and *wxDC::GetTextExtent* (page 115).

All fonts are automatically added to the global pointer **wxTheFontList**. Call *wxFontList::FindOrCreateFont* (page 164) to return a previously-created font if possible.

### **wxFont::~~wxFont**

**void ~wxFont(void)**

Destroys a font object. Do not manually destroy a font which has been assigned to a canvas. All GDI objects, including fonts, are automatically destroyed on program exit, so there is no danger of memory leakage as in conventional Windows programming.

If you have to delete the font (for example, you are creating a lot of them), then call *wxDC::SetFont* (page 118) with a NULL argument to ensure that the old font is restored, and the current font is selected out of the device context.

### **wxFont::GetFaceName**

**char \* GetFaceName(void)**

Returns the typeface name associated with the font, or NULL if there is no typeface information.

### **wxFont::GetFamily**

**int GetFamily(void)**

Gets the font family. See *wxFont* (page 158) for a list of valid family identifiers.

### **wxFont::GetFontId**

**int GetFontId(void)**

Returns the font id, if the portable font system is in operation. See *Font overview* (page 381) for further details.

### **wxFont::GetPointSize**

**int GetPointSize(void)**

Gets the point size.

### **wxFont::GetStyle**

**int GetStyle(void)**

Gets the font style. See *wxFont* (page 158) for a list of valid styles.



**wxFont::GetUnderlined****Bool GetUnderlined(void)**

TRUE if the font is underlined.

**wxFont::GetWeight****int GetWeight(void)**

Gets the font weight. See *wxFont* (page 158) for a list of valid weight identifiers.

**9.42. wxFontData: wxObject**

See also *wxFontDialog overview* (page 387)

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

**wxFontData::wxFontData****void wxFontData(void)**

Constructor. Initializes *fontColour* to black, *showHelp* to black, *allowSymbols* to TRUE, *enableEffects* to TRUE, *initialFont* to NULL, *chosenFont* to NULL, *minSize* to 0 and *maxSize* to 0.

**wxFontData::~~wxFontData****void ~wxFontData(void)**

Destructor.

**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.

**wxFontData::GetAllowSymbols****Bool GetAllowSymbols(void)**

Under MS Windows, returns a flag determining whether symbol fonts can be selected. Has no effect on other platforms.

The default value is TRUE.

**wxFontData::GetColour****wxColour& GetColour(void)**

Gets the colour associated with the font dialog.

The default value is black.

**wxFontData::GetChosenFont****wxFont \* GetChosenFont(void)**

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.

**wxFontData::GetEnableEffects****Bool GetEnableEffects(void)**

Determines whether 'effects' are enabled under Windows. This refers to the controls for manipulating colour, strikeout and underline properties.

The default value is TRUE.

**wxFontData::GetInitialFont****wxFont \* GetInitialFont(void)**

Gets the font that will be initially used by the font dialog. This should have previously been set by the application.

**wxFontData::GetShowHelp****Bool GetShowHelp(void)**

Returns TRUE if the Help button will be shown (Windows only).

The default value is FALSE.

**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.

**wxFontData::SetChosenFont****void SetChosenFont(wxFont \*font)**

Sets the font that will be returned to the user (for internal use only).

**wxFontData::SetColour****void SetColour(wxColour& colour)**

Sets the colour that will be used for the font foreground colour.

The default colour is black.

**wxFontData::SetInitialFont****void SetInitialFont(wxFont \*font)**

Sets the font that will be initially used by the font dialog.

**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).

**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.

**wxFontData::operator =****void operator =(const wxFontData& data)**

Assignment operator for the font data.

**9.43. wxFontDialog: wxDialogBox**

See also *Overview* (page 387)

This class represents the font chooser dialog.

`wxFontDialog` is available under Motif and Windows. Under XView there seem to be some problems, probably related to modal dialogs.

### **wxFontDialog::wxFontDialog**

**void 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.

### **wxFontDialog::~~wxFontDialog**

**void ~wxFontDialog(void)**

Destructor.

### **wxFontDialog::GetFontData**

**wxFontData& GetFontData(void)**

Returns the *font data* (page 160) associated with the font dialog.

### **wxFontDialog::Show**

**Bool Show(Bool flag)**

Shows the dialog, returning TRUE if the user pressed Ok, and FALSE otherwise.

If the user cancels the dialog (Show returns FALSE), no font will be created. If the user presses OK (Show returns TRUE), a new `wxFont` will be created and stored in the font dialog's `wxFontData` structure. Retrieve and delete this font if you do not wish to use it. Otherwise, retrieve and use it.

## **9.44. wxFontList: wxList**

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'.

### **wxFontList::wxFontList**

**void wxFontList(void)**

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

**wxFontList::AddFont****void AddFont(wxFont \*font)**

Used by wxWindows to add a font to the list, called in the font constructor.

**wxFontList::FindOrCreateFont****wxFont \* FindOrCreateFont(int point\_size, int family, int style, int weight, Bool underline = FALSE, char \*facename = NULL)**

Finds a font of the given specification, or creates one and adds it to the list. See the *wxFont constructor* (page 158) for details of the arguments.

**wxFontList::RemoveFont****void RemoveFont(wxFont \*font)**

Used by wxWindows to remove a font from the list.

**9.45. wxFontNameDirectory: wxObject**

See also *Overview* (page 382)

There is a single instance of this class, called *wxTheFontNameDirectory*. Its purpose is to manage font names and identifiers for the portable font system, which is mandatory under X and optional under Windows.

**wxFontNameDirectory::wxFontNameDirectory****void wxFontNameDirectory(void)**

Constructor.

**wxFontNameDirectory::~~wxFontNameDirectory****void ~wxFontNameDirectory(void)**

Destructor.

**wxFontNameDirectory::FindOrCreateFontId****int FindOrCreateFontId(const char \*name, int family)**

Returns the font id for the given font name. If the name has not yet been used, the directory tries to initialize the font using specifications from the resources. The given family id is used for the new font if it is created and the resource does not specify a family id.

**wxFontNameDirectory::GetAFMName****char \* GetAFMName(int fontId, int weight, int style)**

Returns the AFM (Adobe Font Metric) filename for the font, with the path or extension.

**wxFontNameDirectory::GetFamily****int GetFamily(int fontId)**

Returns the family for the given font id.

**wxFontNameDirectory::GetFontId****int GetFontId(const char \*name)**

Get the existing font id corresponding to the font name, or 0 if the name has not been initialized previously.

**wxFontNameDirectory::GetFontName****char \* GetFontName(int fontId)**

Returns the font name for the given font id.

**wxFontNameDirectory::GetNewFontId****int GetNewFontId(void)**

Generates a new font id for direct initialization of a font with Initialize.

**wxFontNameDirectory::GetPostScriptName****char \* GetPostScriptName(int fontId, int weight, int style)**

Returns the real PostScript name for the font.

**wxFontNameDirectory::GetScreenName****char \* GetScreenName(int fontId, int weight, int style)**

Returns the platform-specific screen name for the font.

**wxFontNameDirectory::Initialize****void Initialize(int fontId, int family, const char \*name)**

Initializes sets up a new font with the given font id, default family id and font name. Resource specifications are read using this name.

## 9.46. wxForm: wxObject

### 9.46.1. The purpose of the form class

The wxForm provides form-like functionality, relieving the programmer of the tedium of defining all the physical panel items and the callbacks handling out-of-range data. It allows the application writer to write form dialogs quickly (albeit programmatically) with panel items being chosen automatically according to the given constraints. The supplied form demo shows how succinct a form definition can be. A form gets laid out from left to right; the programmer can intersperse new lines and specify item sizes, but for brevity no more control is allowed.

A form does not presuppose a particular type of panel: any window derived from `wxPanel` may be associated with a form, once the form has been built by adding form items. Also, a form reads from and writes to any C++ variables in your program---just supply pointers to the variables, and the form handles the rest.

### 9.46.2. Constraints on form items

Each item in a form may be supplied with zero or more constraints, where the range of possible constraints depends on the data type, and the displayed panel item depends upon the data type and the constraint(s) given. For example, a string form item with a list of possible strings as a constraint will produce a list box on the panel; an integer form item with a range constraint will result in a slider being displayed. The user may define his or her own constraint by passing a function as a constraint which returns FALSE if the constraint was violated, TRUE otherwise. The function should write an appropriate message into the buffer passed to it if the constraint was violated.

### 9.46.3. Form appearance

Once displayed on a panel, a form shows Ok, Cancel, Update, Revert and Help buttons along the top, with the user-supplied items below. When the user presses Ok, the form items are checked for violation of constraints; if any violations are found, an appropriate error message is displayed and the user must correct the mistake (or press Cancel, which leaves the item values as they were after the last Update). Pressing Update also checks the constraints and updates the values, but typically does not dismiss the dialog. Revert causes the displayed values to take on the values at the last Update. Pressing Help cause the OnHelp member to be called, which by default does nothing. By default, the OnOk and OnCancel messages dismiss and delete the dialog box and form, but these may be overridden by the application (see below).

The display-type values which may be passed to a form-item creation function are as follows:

wxFORM_DEFAULT	Let wxWindows choose a suitable panel item.
wxFORM_SINGLE_LIST	Use a single-selection listbox. Default for string item with a one-of constraint.
wxFORM_CHOICE	Use a choice item.
wxFORM_CHECKBOX	Use a checkbox. Default for boolean item.
wxFORM_TEXT	Use a single-line text item. Default for floating point item, and for string and integer items with no constraints.
wxFORM_MULTITEXT	Use a multi-line text item.

`wxFORM_RADIOBOX` Use a radiobox with a one-of constraint.  
`wxFORM_SLIDER` Use a slider. Default for integer item with range constraint.

The `wxFormItem` and `wxFormItemConstraint` classes are not detailed in this manual since their members do not need to be directly accessed by the user. Functions for creating form items and constraints for passing to `wxForm::Add` are given in the next subsection.

#### 9.46.4. Example

The following is an example of a form definition, taken from the form demo. Here, a new form **MyForm** has been derived, and a new member **EditForm** has been defined to edit objects of the type **MyObject**, given a panel to display it on.

```
void MyForm::EditForm(MyObject *object, wxPanel *panel)
{
    Add(wxMakeFormString("string 1", &(object->string1), wxFORM_DEFAULT,
        new
wxList(wxMakeConstraintFunction(MyConstraint), 0)));
    Add(wxMakeFormNewLine());

    Add(wxMakeFormString("string 2", &(object->string2), wxFORM_DEFAULT,
        new wxList(wxMakeConstraintStrings("One", "Two",
"Three", 0), 0)));
    Add(wxMakeFormString("string 3", &(object->string3), wxFORM_CHOICE,
        new wxList(wxMakeConstraintStrings("Pig", "Cow",
"Aardvark", "Gorilla", 0), 0)));

    Add(wxMakeFormNewLine());
    Add(wxMakeFormShort("int 1", &(object->int1), wxFORM_DEFAULT,
        new wxList(wxMakeConstraintRange(0.0, 50.0),
0)));
    Add(wxMakeFormNewLine());

    Add(wxMakeFormFloat("float 1", &(object->float1), wxFORM_DEFAULT,
        new wxList(wxMakeConstraintRange(-100.0, 100.0),
0)));
    Add(wxMakeFormBool("bool 1", &(object->bool1)));
    Add(wxMakeFormNewLine());

    Add(wxMakeFormButton("Test button", (wxFunction)MyButtonProc));

    AssociatePanel(panel);
}
```

#### **wxForm::wxForm**

```
void wxForm(int useButtons = wxFORM_BUTTON_ALL,  
int placeButtons = wxFORM_BUTTON_AT_TOP)
```

Constructor. The value of *useButtons* determines which buttons are automatically created, and must be a bit list of the following identifiers:

- `wxFORM_BUTTON_OK`
- `wxFORM_BUTTON_CANCEL`



- `wxFORM_BUTTON_HELP`
- `wxFORM_BUTTON_UPDATE`
- `wxFORM_BUTTON_REVERT`
- `wxFORM_BUTTON_ALL`

`wxFORM_BUTTON_ALL` is the same as specifying all buttons except `wxFORM_BUTTON_HELP`.

*placeButtons* may be used to specify whether the form buttons are placed the top or bottom of the form, and may be one of:

- `wxFORM_BUTTON_AT_TOP`
- `wxFORM_BUTTON_AT_BOTTOM`

### **`wxForm::~~wxForm`**

**`void ~wxForm(void)`**

Destructor. Does not delete the associated panel or any panel items, but does delete all form items.

### **`wxForm::Add`**

**`void Add(wxFormItem *item, long id = -1)`**

Adds a form item to the form. If an id is given this is associated with the form item; otherwise a new id is generated, by which the item may be identified later.

### **`wxForm::AssociatePanel`**

**`void AssociatePanel(wxPanel *panel)`**

Associates the form with the given panel (or window derived from `wxPanel`, such as `wxDialogBox`). This causes a number of items to be created on the panel using information from the list of form items. The panel should be shown after this has been called.

### **`wxForm::Delete`**

**`Bool Delete(long id)`**

Deletes the given form item by id. Returns `TRUE` if successful.

### **`wxForm::FindItem`**

**`wxNode * FindItem(long id)`**

Given a form item id, returns a list node containing the form item.

**wxFrm::IsEditable****Bool IsEditable(void)**

Returns TRUE if the form can be edited.

**wxFrm::OnCancel****void OnCancel(void)**

This member may be derived by the application. When the user presses the Cancel button, this is called, allowing the application to take action. By default, **OnCancel** deletes the form and the panel associated with it, probably the normal desired behaviour.

**wxFrm::OnHelp****void OnHelp(void)**

This member may be derived by the application. When the user presses the Help button, this is called, allowing the application to take action.

**wxFrm::OnOk****void OnOk(void)**

This member may be derived by the application. When the user presses the OK button, this is called, allowing the application to take action. By default, **OnOk** deletes the form and the panel associated with it, probably the normal desired behaviour. Note that if any form item constraints were violated when the user pressed OK, the member does not get called.

**wxFrm::OnRevert****void OnRevert(void)**

This member may be derived by the application. When the user presses the Revert button, the C++ form item variable values in effect before the last Update are restored. Then this member is called, allowing the application to take further action.

**wxFrm::OnUpdate****void OnUpdate(void)**

This member may be derived by the application. When the user presses the Update button, the C++ form item variable values are updated to the values on the panel. Then this member is called, allowing the application to take further action.

**wxFrm::RevertValues**

**void RevertValues(void)**

Internal function for displaying the C++ form item values in the displayed panel items. Should not need to be called by the user.

**wxForm::Set****Bool Set(long id, wxFormItem \*item)**

Given a form item id, replaces an existing item with that id with the given form item. Returns TRUE if successful.

**wxForm::SetEditable****void SetEditable(Bool id)**

Sets the form to be editable (TRUE) or read-only (FALSE).

**wxForm::UpdateValues****Bool UpdateValues(void)**

Internal function for setting the C++ form item values to the values set in the panel items. Should not need to be called by the user.

**9.46.21. Functions for making form items and constraints**

These functions make form items and their associated constraints for passing to **wxForm::Add**.

**wxFormItem \* wxMakeFormButton(char \*label, wxFunction fun)**

Makes a button with a conventional callback.

**wxFormItem \* wxMakeFormMessage(char \*label)**

Makes a message.

**wxFormItem \* wxMakeFormNewLine(void)**

Adds a newline.

**wxFormItem \* wxMakeFormLong(char \*label, long \*var,  
int item\_type = wxFORM\_DEFAULT, wxList \*constraints = NULL,  
char \*help\_string = NULL, int style = 0, int width = -1,  
int height = -1)**

Makes a long integer form item, given a label, a pointer to the variable holding the value, an item type, and a list of constraints (see below). *style* may be wxHORIZONTAL or wxVERTICAL (for label orientation). *help\_string* is currently not used.

**wxFormItem \* wxMakeFormShort(char \*label, int \*var,**

```
int item_type = wxFORM_DEFAULT, wxList *constraints = NULL,  
char *help_string = NULL, int style = 0, int width = -1,  
int height = -1)
```

Makes an integer form item, given a label, a pointer to the variable holding the value, an item type, and a list of constraints (see below). *style* may be wxHORIZONTAL or wxVERTICAL (for label orientation). *help\_string* is currently not used.

```
wxFormItem * wxMakeFormDouble(char *label, double *var,  
int item_type = wxFORM_DEFAULT, wxList *constraints = NULL,  
char *help_string = NULL, int style = 0, int width = -1,  
int height = -1)
```

```
wxFormItem * wxMakeFormFloat(char *label, float *var,  
int item_type = wxFORM_DEFAULT, wxList *constraints = NULL,  
char *help_string = NULL, int style = 0, int width = -1,  
int height = -1)
```

Makes a floating-point form item, given a label, a pointer to the variable holding the value, an item type, and a list of constraints (see below). *style* may be wxHORIZONTAL or wxVERTICAL (for label orientation). *help\_string* is currently not used.

```
wxFormItem * wxMakeFormBool(char *label, Bool *var,  
int item_type = wxFORM_DEFAULT, wxList *constraints = NULL,  
char *help_string = NULL, int style = 0, int width = -1,  
int height = -1)
```

Makes a boolean form item, given a label, a pointer to the variable holding the value, an item type, and a list of constraints (see below). *style* may be wxHORIZONTAL or wxVERTICAL (for label orientation). *help\_string* is currently not used.

```
wxFormItem * wxMakeFormString(char *label, char **var,  
int item_type = wxFORM_DEFAULT, wxList *constraints = NULL,  
char *help_string = NULL, int style = NULL, int width = -1,  
int height = -1)
```

Makes a string form item, given a label, a pointer to the variable holding the value, an item type, and a list of constraints (see below). *style* may be wxHORIZONTAL or wxVERTICAL (for label orientation). *help\_string* is currently not used.

```
wxFormItemConstraint * wxMakeConstraintStrings(wxList *list)
```

Makes a constraint specifying that the value must be one of the strings given in the list.

```
wxFormItemConstraint * wxMakeConstraintStrings(char *first, ...)
```

Makes a constraint specifying that the value must be one of the strings given in the variable-length argument list, *terminated with a zero*.

```
wxFormItemConstraint * wxMakeConstraintFunction(wxConstraintFunction func)
```

Makes a constraint with a function that gets called when the value is being checked. The function should return FALSE if the constraint was violated, TRUE otherwise. The function should also write an appropriate message into the buffer passed to it if the constraint was violated. The type **wxConstraintFunction** is defined as follows:

```
typedef Bool (*wxConstraintFunction)(int type, char *value, char *label, char *msg)
```

*type* is the type of the item, for instance `wxFORM_STRING`. *value* is the address of the variable containing the value, and should be coerced to the correct type, except for `wxFORM_STRING`, where no coercion is required.

**wxFormItemConstraint \* wxMakeConstraintRange(double lo, double hi)**

Makes a range constraint; can be used for integer and floating point form items.

## 9.47. wxFormItem: wxObject

A form item is a data structure for representing a panel item (control, widget) in a form. It is returned from **wxMakeForm...** function and normally you will not need a handle to it except to pass it to `wxForm::Add` (page 168). However, you may wish to get the actual panel item from the form item, for example to disable the panel item. In which case, save a handle to the form item, then call (for example) `wxFormItem::GetPanellItem` (page 172) after you have associated the form and the panel or dialog box.

### wxFormItem::GetPanellItem

**wxItem \* GetPanellItem(void)**

Gets the panel item for this form item. Must only be called after `wxForm::AssociatePanel` (page 168) has been called.

## 9.48. wxFrame: wxWindow

A frame is a window which contains subwindows of various kinds. It has a title bar and, optionally, a menu bar, and a status line. Depending on the platform, the frame has further menus or buttons relating to window movement, sizing, closing, etc. Most of these events are handled by the host system without need for special handling by the application. However, the application should normally define an `wxFrame::OnClose` (page 176) handler for the frame so that related data and subwindows can be cleaned up.

A frame may contain the subwindows `wxCanvas` (page 57), `wxPanel` (page 228) and `wxTextWindow` (page 302).

Some of the MS Windows issues of Multiple Document Interface (MDI) versus Single Document Interface (SDI) frames are covered in the user manual.

If you wish to have a toolbar on an MDI parent frame, create the toolbar as normal (as a child of the MDI frame), set the appropriate height for it, and call **wxFrame::SetToolBar**. `wxWindows` will now manage the toolbar automatically. **Note:** SDI frame and MDI child frame toolbars must still be managed by the application in an **OnSize** member function.

### wxFrame::wxFrame

```
void wxFrame(wxFrame *parent, char *title, int x = -1, int y = -1,
             int width = -1, int height = -1,
             long style = wxSDI | wxDEFAULT_FRAME, char *name = "frame")
```

Constructor. The *parent* parameter can be NULL or an existing frame; if an existing frame is

used under MS Windows, the child frame is always on top of the parent, and will be iconized when the parent is iconized.

If *title* is non-NULL, it is placed on the window frame.

The style parameter may be a combination of the following, using the bitwise 'or' operator.

**wxICONIZE**      Display the frame iconized (minimized) (Windows only).  
**wxCAPTION**      Puts a caption on the frame (Windows and XView only).  
**wxDEFAULT\_FRAME**    Defined as (**wxMINIMIZE\_BOX** | **wxMAXIMIZE\_BOX** |  
                         **wxTHICK\_FRAME** | **wxSYSTEM\_MENU** | **wxCAPTION**).  
**wxMDI\_CHILD**    Specifies a Windows MDI (multiple document interface) child frame.  
**wxMDI\_PARENT**    Specifies a Windows MDI (multiple document interface) parent frame.  
**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).  
**wxSDI**            Specifies a normal SDI (single document interface) 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).  
**wxRESIZE\_BORDER**    Displays a resizeable border around the window (Motif only).  
**wxTINY\_CAPTION\_HORIZ**    Under MS Windows, displays a small horizontal caption if  
                         USE\_ITSY\_BITS is set to 1 in wx\_setup.h and the Microsoft ItsyBitsy library  
                         has been compiled. Use instead of wxCAPTION.  
**wxTINY\_CAPTION\_VERT**    Under MS Windows, displays a small vertical caption if  
                         USE\_ITSY\_BITS is set to 1 in wx\_setup.h and the Microsoft ItsyBitsy library  
                         has been compiled. Use instead of wxCAPTION.

For Motif, the MWM (the Motif Window Manager) should be running for any styles to work (otherwise all styles take effect).

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

See *MDI versus SDI* (page 9) for a discussion of how the MS Windows Multiple Document Interface convention is supported..

## **wxFrame::~~wxFrame**

**void ~wxFrame(void)**

Destructor. Destroys all child windows and menu bar if present.

## **wxFrame::Centre**

**void Centre(int direction = wxBOTH)**

Centres the frame on the display. The parameter may be **wxHORIZONTAL**, **wxVERTICAL** or **wxBOTH**.

**wxFrame::Command****void Command(int id)**

Simulate a menu command. *id* is the identifier for a menu item.

**wxFrame::Create****void Create(wxFrame \*parent, char \*title, int x = -1, int y = -1,  
int width = -1, int height = -1,  
long style = wxSDI | wxDEFAULT\_FRAME, char \*name = "frame")**

Used in two-step frame construction. See *wxFrame::wxFrame* (page 172) for further details.

**wxFrame::CreateStatusLine****void CreateStatusLine(int number = 1)**

Creates a status line at the bottom of the frame. The width of the status line is the whole width of the frame (adjusted automatically when resizing), and the height and text size are chosen by the host system.

The default is to create one field the width of the frame; specify a value between 1 and 5 to create a multi-field status line.

See also *wxFrame::SetStatusText* (page 179).

**wxFrame::Fit****void Fit(void)**

Reize the frame to just fit around the subwindows. If a frame has only one subwindow, that subwindow will be resized by the default *wxFrame::OnSize* (page 178) member to fit inside the frame.

**wxFrame::GetMenuBar****wxMenuBar \* GetMenuBar(void)**

Returns a pointer to the menubar currently associated with the frame (if any).

**wxFrame::GetTitle****char \* GetTitle(void)**

Gets a temporary pointer to the frame title. See *wxFrame::SetTitle* (page 179).

**wxFrame::GetToolBar**

**wxWindow \* GetToolBar(void)**

Under Windows only, gets the window to be used as a toolbar for this MDI parent window.

**wxFrame::Iconize****void Iconize(Bool *iconize*)**

If TRUE, iconizes the frame; if FALSE, shows and restores it.

**wxFrame::Iconized****Bool Iconized(void)**

Returns TRUE if the frame is iconized.

**wxFrame::LoadAccelerators****void LoadAccelerators(char \**resource*)**

Loads keyboard accelerators for this frame (Windows only). Accelerator tables map keystrokes onto control and menu identifiers, so the programmer does not have to explicitly program this correspondence.

See the hello demo (`hello.cpp` and `hello.rc`) for an example of accelerator usage. This is a fragment from `hello.rc`:

```
#define HELLO_LOAD_FILE 111

menus_accel ACCELERATORS
{
    "^L", HELLO_LOAD_FILE
}
```

If you call `LoadAccelerators`, you need to override `wxFrame::OnActivate` to do nothing.

**wxFrame::Maximize****void Maximize(Bool *maximize*)**

Maximizes the frame if *maximize* is TRUE, otherwise restores it (MS Windows only).

**wxFrame::OnActivate****void OnActivate(Bool *active*)**

Called when a window is activated or deactivated (MS Windows only). If the window is being



activated, *active* is TRUE, else it is FALSE.

If you call `wxFrame::LoadAccelerators`, you need to override this function e.g.

```
void OnActivate(Bool) {};
```

## **wxFrame::OnClose**

### **Bool OnClose(void)**

Sent to the frame when the user has tried to close the window using the window manager (X) or system menu (Windows). If TRUE is returned by **OnClose**, the frame will be deleted by the system, otherwise the attempt will be ignored. Derive your own class to handle this message; the default handler returns FALSE.

This member is a good place to delete other frames which are conceptually or actually subframes of this frame, since if all frames are not deleted, the application cannot exit. This is the top-level **OnClose** handler for the 'hello' demo:

```
Bool MyFrame::OnClose(void)
{
    if (subframe)
        delete subframe;

    return TRUE;
}
```

When quitting the application from inside the application, for example from a menu item, call the frame's **OnClose** member before deleting the frame. `wxWindows` then causes the application to exit without further ado. For example:

```
// Intercept menu commands
void MyFrame::OnMenuCommand(int id)
{
    switch (id)
    {
        ...
        case HELLO_QUIT:
        {
            if (OnClose())
                delete this;
            break;
        }
        ...
    }
}
```

## **wxFrame::OnMenuCommand**

### **void OnMenuCommand(int id)**

Sent to the window when an item on the window's menu has been chosen. Derive your own frame class to handle this message. For example:

```
// Intercept menu commands
void MyFrame::OnMenuCommand(int id)
{
    switch (id)
    {
        case HELLO_LOAD_FILE:
        {
            char *s = wxFileSelector("Load text file", NULL, NULL, NULL,
            "*.txt");
            if (s)
                frame->text_window->LoadFile(s);
            break;
        }
        case HELLO_QUIT:
        {
            if (OnClose())
                delete this;
            break;
        }
        case HELLO_PRINT_EPS:
        {
            wxPostScriptDC dc(NULL, TRUE);
            if (dc.Ok())
            {
                dc.StartDoc("Hello printout");
                dc.StartPage();
                Draw(dc, TRUE);
                dc.EndPage();
                dc.EndDoc();
            }
            break;
        }
        case HELLO_ABOUT:
        {
            (void)wxMessageBox("wxWindows GUI library demo", "About wxHello",
            wxOK|wxCENTRE);
            break;
        }
    }
}
```

## **wxFrame::OnMenuSelect**

**void OnMenuSelect(int id)**

Sent to the window when an item on the window's menu has been selected (i.e. the cursor is on the item, but the left button has not been released). Derive your own frame class to handle this message.

The default **OnMenuSelect** member puts the menu item help string on the status line, if a status line has been created.

This function is called under MS Windows and Motif, but not XView.

## **wxFrame::OnSize**

**void OnSize(int w, int h)**

Called when the user or application resizes the frame. The parameters give the total size of the frame. The default **OnSize** member 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).

## **wxFrame::SetIcon**

**void SetIcon(wxIcon \* icon)**

Sets the icon for this frame, deleting any existing one. The icon is now 'owned' by the frame and will be deleted on frame deletion, so do not delete the icon yourself.

Note an important difference between XView and MS Windows behaviour. In MS Windows, the title of the frame is the icon label, wrapping if necessary for a long title. If the frame title changes, the icon label changes. In XView, the icon label cannot be changed once the icon has been associated with the frame. Also, there is no wrapping, and icon labels must therefore be short.

The best thing to do to accommodate both situations is to have the frame title set to a short string when setting the icon. Then, set the frame title to the desired text. In XView, the icon will keep its short text. In MS Windows, the longer (probably more meaningful) title will be shown.

Instead of using **wxFrame::SetIcon** under Windows, 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, and **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 **wxFrame::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* (page 183).

## **wxFrame::SetMenuBar**

**void SetMenuBar(wxMenuBar \*menuBar)**

Tells the frame to show the given menu bar. 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 MS Windows, a call to `wxFrame::OnSize` is generated, so be sure to initialize data members properly before calling `SetMenuBar`.

See also *wxMenuBar* (page 209), *wxMenu* (page 206).

**wxFrame::SetStatusText****void SetStatusText(char \* text, int number = 0)**

Sets the status line text and redraws the status line. Use an empty (not NULL) string to clear the status line. Optionally use *number* to specify a field in the status line, starting from zero and consistent with the number of fields specified in *wxFrame::CreateStatusLine* (page 174).

**wxFrame::SetStatusWidths****void SetStatusWidths(int n, int \*widths)**

Sets the widths of the fields in the status line. *n* must be the same used in *CreateStatusLine* (page 174). *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.

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.

Windows only.

**wxFrame::SetTitle****void SetTitle(char \* title)**

Sets the frame title. See *wxFrame::GetTitle* (page 174).

**wxFrame::SetToolBar****void SetToolBar(wxWindow \*toolbar)**

Under Windows only, sets the window to be used as a toolbar for this MDI parent window. It is necessary since *wxWindows* does not provide general functionality for application management of an MDI client area.

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, which must itself have been created as an MDI

parent frame.

Please note that SDI frames and MDI child windows must have their toolbars managed by the application.

## **wxFrame::StatusLineExists**

### **Bool StatusLineExists(void)**

Returns TRUE if the status line has previously been created. See *wxFrame::CreateStatusLine* (page 174), *wxFrame::SetStatusText* (page 179).

## **9.49. wxFunction**

The type of a panel item callback function.

### **wxFunction**

```
typedef void (*wxFunction)(wxObject&, wxCommandEvent&)
```

The type of a panel item callback function. The first parameter is a reference to the object, and the second is a command event structure. See *wxItem* (page 191), *wxCommandEvent* (page 88).

## **9.50. wxGauge: wxItem**

A gauge is a horizontal or vertical bar which shows the progress of some operation, or perhaps an amount of something.

Use *SetValue* (page 182) to set the value of the gauge. You can use *wxItem::SetButtonColour* and *wxItem::SetBackgroundColour* to set the gauge value and background colours respectively.

Note that wxWindows support for wxGauge is off by default; to enable it, the file *wx\_setup.h* must be edited, and the gauge code in *contrib/gauge* (for Windows) and *contrib/xmgauge* (for Motif) must be compiled.

### **wxGauge::wxGauge**

#### **void wxGauge(void)**

Constructor, for deriving classes.

```
void wxGauge(wxPanel *parent, char *label,  
             int range, int x = -1, int y = -1,  
             int width = -1, int height = -1,  
             long style = wxHORIZONTAL, char *name = "gauge")
```

Constructor, creating and showing a gauge.

If *label* is non-NULL, it will be used as the gauge label.

*range* is an integer specifying the number of units the gauge is divided into.

The parameters *x* and *y* are used to specify an absolute position, or a position after the previous panel item if omitted or default.

If *width* or *height* are omitted (or are less than zero), an appropriate size will be used for the gauge.

*style* may be a bit list of the following:

`wxGA_HORIZONTAL` Creates a horizontal gauge.  
`wxGA_VERTICAL` Creates a vertical gauge.  
`wxGA_PROGRESSBAR` Under Windows 95, creates a horizontal progress bar.

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual gauges.

### **wxGauge::~~wxGauge**

**void ~wxGauge(void)**

Destructor, destroying the gauge.

### **wxGauge::Create**

**Bool Create(wxPanel \*parent, char \*label,  
int range, int x = -1, int y = -1,  
int width = -1, int height = -1,  
long style = wxHORIZONTAL, char \*name = "gauge")**

Creates the gauge for two-step construction. Derived classes should call or replace this function. See `wxGauge::wxGauge` (page 180) for further details.

### **wxGauge::GetBezelFace**

**int GetBezelFace(void)**

Returns the width of the 3D bezel face.

### **wxGauge::GetRange**

**int GetRange(void)**

Returns the maximum position of the gauge.

### **wxGauge::GetValue**

**int GetValue(void)**

Returns the current position of the gauge.

**wxGauge::SetBezelFace****void SetBezelFace(int width)**

Sets the 3D bezel face width.

**wxGauge::SetRange****void SetRange(int range)**

Sets the range of the gauge.

**wxGauge::SetShadowWidth****void SetShadowWidth(int width)**

Sets the 3D shadow width.

**wxGauge::SetValue****void SetValue(int pos)**

Sets the position of the gauge.

**9.51. wxGroupBox: wxItem**

A group box is a rectangle drawn around other panel items to denote a logical grouping of items.

Currently it is implemented for Windows and Motif only.

**wxGroupBox::wxGroupBox****void wxGroupBox(void)**

Constructor, for deriving classes.

**void wxGroupBox(wxPanel \*parent, char \*label,  
int x = -1, int y = -1,  
int width = -1, int height = -1,  
long style = 0, char \*name = "groupBox")**

Constructor, creating and showing a group box.

If *label* is non-NULL, it will be used as the group box label.

The parameters *x* and *y* are used to specify an absolute position, or a position after the previous panel item if omitted or default.

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual group boxes.

**wxGroupBox::~~wxGroupBox****void ~wxGroupBox(void)**

Destructor, destroying the group box.

**wxGroupBox::Create****Bool Create(wxPanel \*parent, char \*label,  
int x = -1, int y = -1,  
int width = -1, int height = -1,  
long style = 0, char \*name = "groupBox")**

Creates the group box for two-step construction. Derived classes should call or replace this function. See *wxGroupBox::wxGroupBox* (page 182) for further details.

**9.52. wxIcon: wxBitmap**

An icon is a small rectangular bitmap usually used for denoting a minimized application. It is optional (but desirable) to associate a pertinent icon with a frame. Obviously icons in X and MS Windows are created in a different manner, and colour icons in X are difficult to arrange. Therefore, separate icons will 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 X, this will ensure that fresh X resources are allocated for this frame. In MS 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.

The following shows the conditional compilation required to define an icon in X and in MS Windows. The alternative is to use the string version of the icon constructor, which loads a file under X and a resource under MS Windows, but has the disadvantage of requiring the X icon file to be available at run-time. If anyone can invent a scheme or macro which does the following more elegantly and platform-independently, I'd like to see it!

```
#ifdef wx_x
#include "aiai.xbm"
#endif
#ifdef wx_msw
    wxIcon *icon = new wxIcon("aiai");
#endif
#ifdef wx_x
    wxIcon *icon = new wxIcon(aiai_bits, aiai_width, aiai_height);
#endif
```

See also *wxDC::DrawIcon* (page 110), *wxBitmap* (page 48).

**wxIcon::wxIcon****void wxIcon(void)**



Default constructor.

**void wxIcon(char \*\* data)**

Construct an icon by specifying the bits in an included .XPM file (X only). Only available if USE\_XPM\_IN\_X is enabled in wx\_setup.h.

**void wxIcon(char bits[], int width, int height)**

Construct an icon by specifying the bits in an included .XBM file (X only).

For example:

```
#ifdef wx_x
#include "aiai.xbm"
#endif

#ifdef wx_x
    test_icon = new wxIcon(aiai_bits, aiai_width, aiai_height);
#endif
```

**void wxIcon(char \*iconName, long flags)**

Constructor. An icon can be created by passing an array of bits (X only) or by passing a string name. *icon\_name* refers to a filename in X, a resource name in MS Windows.

Construct a cursor by passing a string resource name or filename. Under Motif, *flags* defaults to wxBITMAP\_TYPE\_XBM | wxBITMAP\_DISCARD\_COLOURMAP. Under Windows, it defaults to wxBITMAP\_TYPE\_ICO\_RESOURCE | wxBITMAP\_DISCARD\_COLOURMAP.

Under X, the permitted icon types in the *flags* bitlist are:

wxBITMAP_TYPE_BMP	Load a Windows bitmap file (if USE_IMAGE_LOADING_IN_X is enabled in wx_setup.h).
wxBITMAP_TYPE_GIF	Load a GIF bitmap file (if USE_IMAGE_LOADING_IN_X is enabled in wx_setup.h).
wxBITMAP_TYPE_XBM	Load an X bitmap file.
wxBITMAP_TYPE_XPM	Load an XPM (colour pixmap) file. Only available if USE_XPM_IN_X is enabled in wx_setup.h.

Under Windows, the permitted types are:

wxBITMAP_TYPE_ICO	Load a cursor from a .ico icon file (only if USE_RESOURCE_LOADING_IN_MSW.
	is enabled in wx_setup.h).
wxBITMAP_TYPE_ICO_RESOURCE	Load a Windows resource (as specified in the .rc file).

**wxIcon::~~wxIcon**

**void ~wxIcon(void)**

Destroys the icon. Do not explicitly delete an icon pointer which has been passed to a frame - the

frame will delete the icon when it is destroyed. If assigning a new icon to a frame, the old icon will be destroyed.

### **wxIcon::GetHeight**

**int GetHeight(void)**

Returns the height of the icon.

### **wxIcon::GetWidth**

**int GetWidth(void)**

Returns the width of the icon.

## **9.53. wxHashTable: wxObject**

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. 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* (page 197).

### **wxHashTable::wxHashTable**

**void 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.

### **wxHashTable::~~wxHashTable**

**void ~wxHashTable(void)**

Destroys the hash table.

**wxHashTable::BeginFind****void BeginFind(void)**

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*.

**wxHashTable::Clear****void Clear(void)**

Clears the hash table of all nodes (but as usual, doesn't delete user data).

**wxHashTable::Delete****wxObject \* Delete(long key)****wxObject \* Delete(char \* key)**

Deletes entry in hash table and returns the user's data (if found).

**wxHashTable::Get****wxObject \* Get(long key)****wxObject \* Get(char \* key)**

Gets data from the hash table, using an integer or string key (depending on which hash table constructor was used).

**wxHashTable::MakeKey****long MakeKey(char \*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).

**wxHashTable::Next****wxNode \* Next(void)**

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* (page 221). The user will probably only wish to use

the **wxNode::Data** function to retrieve the data; the node may also be deleted.

### **wxHashTable::Put**

**void Put(long key, wxObject \*object)**

**void Put(char \* 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.

## **9.54. wxHelpInstance: wxClient**

The **wxHelpInstance** class implements the interface by which applications may invoke wxHelp to provide on-line help. Each instance of the class maintains one connection to an instance of wxHelp which belongs to the application, and which is shut down when the **Quit** member of **wxHelpInstance** is called (for example in the **OnClose** member of an application's main frame). Under MS Windows, there is currently only one instance of wxHelp which is used by all applications.

Since there is a DDE link between the two programs, each subsequent request to display a file or section uses the existing instance of wxHelp, rather than starting a new instance each time. wxHelp thus appears to the user to be an extension of the current application. wxHelp may also be invoked independently of a client application.

Normally an application will create an instance of **wxHelpInstance** when it starts, and immediately call **Initialize** to associate a filename with it. wxHelp will only get run, however, just before the first call to display something. See the test program supplied with the wxHelp source.

Include the file `wx_help.h` to use this API, even if you have included `wx.h`.

If you give **TRUE** to the constructor, you can use the native help system where appropriate (currently under Windows only). Omit the file extension to allow wxWindows to choose the appropriate file for the platform.

### **wxHelpInstance::wxHelpInstance**

**void wxHelpInstance(Bool native)**

Constructs a help instance object, but does not invoke wxHelp. If *native* is **TRUE**, tries to use the native help system where possible (Windows Help under MS Windows, wxHelp on other platforms).

### **wxHelpInstance::~~wxHelpInstance**

Destroys the help instance, closing down wxHelp for this application if it is running.

### **wxHelpInstance::Initialize**

**void Initialize(char \*file, int server = -1)**

Initializes the help instance with a help filename, and optionally a server (socket) number (one is chosen at random if this parameter is omitted). Does not invoke wxHelp. This must be called directly after the help instance object is created and before any attempts to communicate with wxHelp.

You may omit the file extension, and in fact this is recommended if you wish to support .xlp files under X and .hlp under Windows.

### **wxHelpInstance::DisplayBlock**

**Bool DisplayBlock(long blockNo)**

If wxHelp is not running, runs wxHelp and displays the file at the given block number. If using Windows Help, displays the file at the given context number.

### **wxHelpInstance::DisplayContents**

**Bool DisplayContents(void)**

If wxHelp is not running, runs wxHelp (or Windows Help) and displays the contents (the first section of the file).

### **wxHelpInstance::DisplaySection**

**Bool DisplaySection(int sectionNo)**

If wxHelp is not running, runs wxHelp and displays the given section. Sections are numbered starting from 1, and section numbers may be viewed by running wxHelp in edit mode.

### **wxHelpInstance::KeywordSearch**

**Bool KeywordSearch(char \*keyWord)**

If wxHelp (or Windows Help) is not running, runs wxHelp (or Windows Help), and searches for sections matching the given keyword. If one match is found, the file is displayed at this section. If more than one match is found, the Search dialog is displayed with the matches (wxHelp) or the first topic is displayed (Windows Help).

### **wxHelpInstance::LoadFile**

**Bool LoadFile(char \*file = NULL)**

If wxHelp (or Windows Help) is not running, runs wxHelp (or Windows Help), and loads the given file. If the filename is not supplied or is NULL, the file specified in **Initialize** is used. If wxHelp 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.

### **wxHelpInstance::OnQuit**

**Bool OnQuit(void)**

Overridable member called when this application's wxHelp is quit (no effect if Windows Help is being used instead).

**wxHelpInstance::Quit****Bool Quit(void)**

If wxHelp is running, quits wxHelp by disconnecting (no effect for Windows Help).

**9.55. wxIndividualLayoutConstraint: wxObject**

See also *Overview and examples* (page 388)

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.

See also *wxLayoutConstraints* (page 196), *wxWindow::SetConstraints* (page 325).

**9.55.1. Edges and relationships**

The *wxEdge* enumerated type specifies the type of edge or dimension of a window.

wxLeft	The left edge.
wxTop	The top edge.
wxRight	The right edge.
wxBottom	The bottom edge.
wxCentreX	The x-coordinate of the centre of the window.
wxCentreY	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.

wxUnconstrained	The edge or dimension is unconstrained (the default for edges).
wxAsIs	The edge or dimension is to be taken from the current window position or size (the default for dimensions).
wxAbove	The edge should be above another edge.
wxBelow	The edge should be below another edge.
wxLeftOf	The edge should be to the left of another edge.
wxRightOf	The edge should be to the right of another edge.
wxSameAs	The edge or dimension should be the same as another edge or dimension.
wxPercentOf	The edge or dimension should be a percentage of another edge or dimension.
wxAbsolute	The edge or dimension should be a given absolute value.

**wxIndividualLayoutConstraint::wxIndividualLayoutConstraint**

**void wxIndividualLayoutConstraint(void)**

Constructor. Not used by the end-user.

**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.

**wxIndividualLayoutConstraint::Absolute****void Absolute(int value)**

Constrains this edge or dimension to be the given absolute value.

**wxIndividualLayoutConstraint::AsIs****void AsIs(void)**

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.

**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.

**wxIndividualLayoutConstraint::Unconstrained****void Unconstrained(void)**

Sets this edge or dimension to be unconstrained, that is, dependent on other edges and dimensions from which this value can be deduced.

**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.

**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.

**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.

**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.

**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`.

**9.56. wxIntPoint: wxObject**

A **wxIntPoint** is a useful data structure for graphics operations. It contains integer point `x` and `y` members. See also *wxPoint* (page 240) for a floating point version.

**wxIntPoint::wxIntPoint****void wxIntPoint(void)****void wxIntPoint(int x, int y)**

Create a point.

**int x****int y**

Members of the **wxIntPoint** object.

**9.57. wxItem: wxWindow**



This is the base class for any widget or control which can be placed on a panel or dialog box.

The following styles may be used for any panel item:

**wxFIXED\_LENGTH**     The label of the item is created with a width proportional to the length of the label string, regardless of proportional font in use. This allows alignment of items if all items are given labels of the same length.

### **wxItem::Centre**

**void Centre**(int *direction* = wxHORIZONTAL)

Centres the frame on the panel or dialog box. The parameter may be wxHORIZONTAL, wxVERTICAL or wxBOTH.

You may still use **Fit** in conjunction with this call, but call **Fit** first before centring items.

### **wxItem::Command**

**void Command**(wxCommandEvent *event*)

Simulate the effect of the user issuing a command to the item. See *wxCommandEvent* (page 88).

### **wxItem::GetBackgroundColour**

**wxColour \* GetBackgroundColour**(void)

Gets the item background colour.

### **wxItem::GetButtonColour**

**wxColour \* GetButtonColour**(void)

Gets the item button colour.

### **wxItem::GetLabelColour**

**wxColour \* GetLabelColour**(void)

Gets the item label colour.

### **wxItem::GetLabel**

**char \* GetLabel**(void)

Gets a temporary pointer to the item's label.

**wxItem::SetBackgroundColour****void SetBackgroundColour(wxColour& colour)**

Sets the item background colour (Motif and Windows only).

**wxItem::SetButtonColour****void SetButtonColour(wxColour& colour)**

Specifies the default colour for drawing value text (Motif and Windows). wxButton items do not respond to this setting under Windows.

**wxItem::SetButtonFont****void SetButtonFont(wxFont \*font)**

Sets the item value font (not XView).

**wxItem::SetLabel****void SetLabel(char \*label)**

Sets the item's label. A copy of the label is taken.

**wxItem::SetLabelColour****void SetLabelColour(wxColour& colour)**

Sets the item label's colour (Motif and Windows only).

**wxItem::SetLabelFont****void SetLabelFont(wxFont \*font)**

Sets the item label font (not XView).

**9.58. wxKeyEvent: wxEvent**

This event class contains information about key events. See *wxCanvas::OnChar* (page 62).

**wxKeyEvent::controlDown****Bool controlDown**

Returns TRUE if control is pressed down.

**wxKeyEvent::keyCode****long keyCode**

Virtual keycode. An enumerated type, one of:

```
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
```

```
WXK_F2  
WXK_F3  
WXK_F4  
WXK_F5  
WXK_F6  
WXK_F7  
WXK_F8  
WXK_F9  
WXK_F10  
WXK_F11  
WXK_F12  
WXK_F13  
WXK_F14  
WXK_F15  
WXK_F16  
WXK_F17  
WXK_F18  
WXK_F19  
WXK_F20  
WXK_F21  
WXK_F22  
WXK_F23  
WXK_F24  
WXK_NUMLOCK  
WXK_SCROLL
```

## **wxKeyEvent::shiftDown**

### **Bool shiftDown**

Returns TRUE if shift is pressed down.

## **wxKeyEvent::wxKeyEvent**

### **void wxKeyEvent(WXTYPE keyEventType)**

Constructor. Currently, the only valid event type is wxEVENT\_TYPE\_CHAR.

## **wxKeyEvent::ControlDown**

### **Bool ControlDown(void)**

Returns TRUE if the control key was down at the time of the key event.

## **wxKeyEvent::KeyCode**

### **long KeyCode(void)**

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 `wx_defs.h` for a full list of the virtual key codes.

**wxKeyEvent::Position****void Position(float \*x, float \*y)**

Obtains the position at which the key was pressed.

**wxKeyEvent::ShiftDown****Bool ShiftDown(void)**

Returns TRUE if the shift key was down at the time of the key event.

**9.59. wxLayoutConstraints: wxObject**

See also *Overview and examples* (page 388)

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.

See also *wxIndividualLayoutConstraint* (page 189), *wxWindow::SetConstraints* (page 325).

**wxLayoutConstraints::wxLayoutConstraints****void wxLayoutConstraints(void)**

Constructor.

**wxLayoutConstraints::bottom****wxIndividualLayoutConstraint bottom**

Constraint for the bottom edge.

**wxLayoutConstraints::centreX****wxIndividualLayoutConstraint centreX**

Constraint for the horizontal centre point.

**wxLayoutConstraints::centreY****wxIndividualLayoutConstraint centreY**

Constraint for the vertical centre point.

**wxLayoutConstraints::height****wxIndividualLayoutConstraint height**

Constraint for the height.

**wxLayoutConstraints::left****wxIndividualLayoutConstraint left**

Constraint for the left-hand edge.

**wxLayoutConstraints::right****wxIndividualLayoutConstraint right**

Constraint for the right-hand edge.

**wxLayoutConstraints::top****wxIndividualLayoutConstraint top**

Constraint for the top edge.

**wxLayoutConstraints::width****wxIndividualLayoutConstraint width**

Constraint for the width.

**9.60. wxList: wxObject**

This class provides 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* (page 185) for a faster method of storage when random access is required.

It is very common to iterate on a list as follows:

```
...
wxPoint *point1 = new wxPoint(100, 100);
wxPoint *point2 = new wxPoint(200, 200);

wxList SomeList;
SomeList.Append(point1);
SomeList.Append(point2);

...

wxNode *node = SomeList.First();
while (node)
{
    wxPoint *point = (wxPoint *)node->Data();
    ...
    node = node->Next();
}
```

To delete nodes in a list as the list is being traversed, replace

```
...
node = node->Next();
...
```

with

```
...
delete point;
delete node;
node = SomeList.First();
...
```

See *wxNode* (page 221) 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.

## **wxList::wxList**

**void wxList(void)**

**void wxList(unsigned int *key\_type*)**

**void wxList(int *n*, wxObject \**objects*[])**

**void 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`.

### **wxList::~~wxList**

**void ~wxList(void)**

Destroys list. Also destroys any remaining nodes, but does not destroy client data held in the nodes.

### **wxList::Append**

**wxNode \* Append(wxObject \*object)**

**wxNode \* Append(long key, wxObject \*object)**

**wxNode \* Append(char \*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.

### **wxList::Clear**

**void Clear(void)**

Clears the list (but does not delete the client data stored with each node).

### **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`.

### **wxList::DeleteNode**

**Bool DeleteNode(wxNode \*node)**

Deletes the given node from the list, returning `TRUE` if successful.

### **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.

**wxList::Find**

**wxNode \* Find(long key)**

**wxNode \* Find(char \*key)**

Returns the node whose stored key matches *key*. Use on a keyed list only.

**wxList::First**

**wxNode \* First(void)**

Returns the first node in the list (NULL if the list is empty).

**wxList::Insert**

**wxNode \* Insert(wxObject \*object)**

Insert object at front of list.

**wxNode \* Insert(wxNode \*position, wxObject \*object)**

Insert object before *position*.

**wxList::Last**

**wxNode \* Last(void)**

Returns the last node in the list (NULL if the list is empty).

**wxList::Member**

**wxNode \* Member(wxObject \*object)**

Returns the node associated with *object* if it is in the list, NULL otherwise.

**wxList::Nth**

**wxNode \* Nth(int n)**

Returns the *n*th node in the list, indexing from zero (NULL if the list is empty or the *n*th node could not be found).

**wxList::Number****int Number(void)**

Returns the number of elements in the list.

**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);
}
```

**9.61. wxListBox: wxItem**

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 wxEVENT\_TYPE\_LISTBOX\_COMMAND for single clicks, and wxEVENT\_TYPE\_LISTBOX\_DCLICK\_COMMAND for double clicks. Another way of intercepting double clicks is to override *wxPanel::OnDefaultAction* (page 231).

Please note that under XView, the height of a listbox cannot be set accurately, since internally, the number of rows must be used to set the height. So it is unlikely that the value returned by *GetSize* will be the same as passed to *SetSize*.

See also *wxChoice* (page 69).

## **wxListBox::wxListBox**

### **void wxListBox(void)**

Constructor, for deriving classes.

```
void wxListBox(wxPanel *parent, wxFunction func, char *label,  
    Bool multiple_selection = wxSINGLE, int x = -1, int y = -1,  
    int width = -1, int height = -1, int n, char *choices[],  
    long style = 0, char *name = "listBox")
```

Constructor, creating and showing a list box.

*func* may be NULL; otherwise it is used as the callback for the list box. Note that the cast (*wxFunction*) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration.

If *label* is non-NULL, it will be used as the listbox label.

The parameters *x* and *y* are used to specify an absolute position, or a position after the previous panel item if omitted or default.

If *width* or *height* are omitted (or are less than zero), an appropriate size will be used for the list box.

*n* is the number of possible choices, and *choices* is an array of strings of size *n*. *wxWindows* allocates its own memory for these strings so the calling program must deallocate the array itself.

*multiple\_selection* is a bit list of some of the following:

<code>wxSINGLE</code>	Single-selection list.
<code>wxMULTIPLE</code>	Multiple-selection list.
<code>wxEXTENDED</code>	Extended-selection list (Motif and Windows).

*style* is a bit list of some of the following. Note that *style* should now be used for all listbox styles, in preference to using the *multiple\_selection* argument. However, the styles in *multiple\_selection* still work for backward compatibility.

<code>wxNEEDED_SB</code>	Create scrollbars if needed.
<code>wxLB_NEEDED_SB</code>	Same as <code>wxNEEDED_SB</code> .
<code>wxALWAYS_SB</code>	Create scrollbars immediately.
<code>wxLB_ALWAYS_SB</code>	Same as <code>wxALWAYS_SB</code> .
<code>wxLB_SINGLE</code>	Single-selection list.
<code>wxLB_MULTIPLE</code>	Multiple-selection list.
<code>wxLB_EXTENDED</code>	Extended-selection list (Motif and Windows).
<code>wxHSCROLL</code>	Create horizontal scrollbar if contents are too wide (Windows only).
<code>wxFIXED_LENGTH</code>	Allows the values of a column of items to be left-aligned. Create an item with this style, and pad out your labels with spaces to the same length. The item labels will initially be created with a string of identical characters, positioning all the values at the same x-position. Then the real label is restored.

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual listboxes.

### **wxListBox::~~wxListBox**

**void ~wxListBox(void)**

Destructor, destroying the list box.

### **wxListBox::Append**

**void Append(char \* item)**

Adds the item to the end of the list box. *item* must be deallocated by the calling program, i.e. wxWindows makes its own copy.

**void Append(char \* item, char \*client\_data)**

Adds the item to the end of the list box, associating the given data with the item. *item* must be deallocated by the calling program.

### **wxListBox::Clear**

**void Clear(void)**

Clears all strings from the list box.

### **wxListBox::Create**

**Bool Create(wxPanel \*parent, wxFunction func, char \*label,  
Bool multiple\_selection = FALSE, int x = -1, int y = -1,  
int width = -1, int height = -1, int n, char \*choices[],  
long style = 0, char \*name = "listBox")**

Creates the listbox for two-step construction. Derived classes should call or replace this function. See *wxListBox::wxListBox* (page 202) for further details.

### **wxListBox::Delete**

**void Delete(int n)**

Delete the *n*th element in the list box.

### **wxListBox::Deselect**

**void Deselect(int n)**

Deselects the given item in the list box.

**wxListBox::FindString****int FindString(int *char* \*s)**

Finds a choice matching the given string, returning the position if found, or -1 if not found.

**wxListBox::GetClientData****char \* GetClientData(int *n*)**

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

**wxListBox::GetSelection****int GetSelection(void)**

Gets the id (position) of the selected string - for single selection list boxes only.

**wxListBox::GetSelections****int GetSelections(int \*\**selections*)**

Gets an array containing the positions of the selected strings. The number of selections is returned. Pass a pointer to an integer array, and do not deallocate the returned array.

**wxListBox::GetString****char \* GetString(int *n*)**

Returns a temporary pointer to the string at position *n*.

**wxListBox::GetStringSelection****char \* GetStringSelection(void)**

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.

**wxListBox::Number****int Number(void)**

Returns the number of items in the listbox.

**wxListBox::Selected**

**Bool Selected(int *n*)**

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

**wxListBox::Set**

**void Set(int *n*, char \**choices*[], char \**clientData*[] = NULL)**

Clears the list box and adds the given strings. Deallocate the array from the calling program after this function has been called.

**wxListBox::SetClientData**

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

Associates the given client data pointer with the given item.

**wxListBox::SetFirstItem**

**void SetFirstItem(int *n*)**

**void SetFirstItem(char \**item*)**

Set the specified item to be the first visible item (not XView).

**wxListBox::SetSelection**

**void SetSelection(int *n*, Bool *select* = TRUE)**

Selects or deselects the given item.

**wxListBox::SetString**

**void SetString(int *n*, char \* *s*)**

Sets the value of the given string.

**wxListBox::SetStringSelection**

**void SetStringSelection(char \* *s*)**

Sets the choice by passing the desired string.

**9.62. wxMemoryDC: wxCanvasDC**

A memory device context provides a means to draw graphics onto a bitmap.

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 before a bitmap can be reselected into another memory DC.

See also *wxBitmap* (page 48), *wxDC* (page 108), *wxCanvasDC* (page 68).

### **wxMemoryDC::wxMemoryDC**

**void wxMemoryDC(void)**

Constructs a new memory device context.

**void wxMemoryDC(wxCanvasDC \*oldDC)**

Constructs a new memory device context with similar attributes to the given canvas 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.

### **wxMemoryDC::SelectObject**

**void SelectObject(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 canvas. For this purpose, you may find *wxDC::DrawIcon* (page 110) easier to use instead.

If the argument is *NULL*, the current bitmap is selected out of the device context, and the original bitmap restored, allowing the current bitmap to be destroyed safely.

## **9.63. wxMenu: wxWindow**

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.

See also *wxFrame::OnMenuCommand* (page 176) and *wxWindow::PopupMenu* (page 324).

**wxMenu::wxMenu****void wxMenu(char \*title = NULL, wxFunction func = NULL)**

The first argument is presently ignored. The second argument is a callback function if the menu is used as a popup using *wxWindow::PopupMenu* (page 324).

**wxMenu::~~wxMenu****void ~wxMenu(void)**

Destructor, destroying the menu.

**wxMenu::Append****void Append(int id, char \* item, char \*helpString = NULL, Bool checkable = FALSE)****void Append(int id, char \* item, wxMenu \*submenu, char \*helpString = NULL)**

Adds the item to the end of the menu. *item* must be deallocated by the calling program. If the second form is used, the given menu will be a pullright submenu (must be created already). From version 1.50k, this can be used dynamically, i.e. after initial creation of a menu or menubar.

Each form can take an optional help string, which can be accessed using **wxMenu::GetHelpString**. The default **wxFrame::OnMenuSelect** member uses this help string to give help on the menu item currently under the cursor.

See the `hello.cpp` demo for an example of using **Append** dynamically to implement a file history facility. See also *wxMenu::SetLabel* (page 209).

**wxMenu::AppendSeparator****void AppendSeparator(void)**

Adds a separator to the end of the menu. Under XView, this appears as a space.

**wxMenu::Break****void Break(void)**

Inserts a break in a menu, causing the next appended item to appear in a new column.

**wxMenu::Check****void Check(int id, Bool flag)**

If *flag* is TRUE, checks the given menu item, else unchecks it.



**wxMenu::Checked****Bool Checked(int id)**

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

**wxMenu::Enable****void Enable(int id, Bool flag)**

If *flag* is TRUE, enables the given menu item, else disables it (greys it). MS Windows, Motif, XView.

**wxMenu::FindItem****int FindItem(char \*itemString)**

Finds the menu item id for a menu item string, or -1 if none found. Any special menu codes are stripped out of source and target strings before matching.

**wxMenu::FindItemForId****wxMenuItem \* FindItemForId(int itemId)**

Finds the menu item object associated with the given menu item identifier, returning NULL if not found.

**wxMenu::GetHelpString****char \* GetHelpString(int itemId)**

Gets a temporary pointer to the help string associated with the menu item identifier (or NULL if there is no help string or the item was not found).

**wxMenu::GetLabel****char \* GetLabel(int id)**

Gets a temporary pointer to the label of the given menu item; copy this for long-term use. *id* is the identifier given to *wxMenu::Append* (page 207).

**wxMenu::GetTitle****char \* GetTitle(void)**

Gets a temporary pointer to the title of the menu.

**wxMenu::SetHelpString****void SetHelpString(int itemId, char \*helpString)**

Sets the help string associated with the menu item identifier.

**wxMenu::SetLabel****void SetLabel(int id, char \*label)**

Sets the label of the given menu item (using the identifier used to append the item to the menu).

See the `hello.cpp` demo for an example of using this command to implement a file history facility. See also `wxMenu::Append` (page 207).

**wxMenu::SetTitle****void SetTitle(char \*title)**

Sets the title of the menu.

**9.64. wxMenuBar: wxWindow**

A menu bar is a series of menus accessible from the top of a frame. Selecting a title pulls down a menu; selecting a menu item causes a *MenuSelection* message to be passed to the frame with the menu item integer id as the only argument.

**wxMenuBar::wxMenuBar****void wxMenuBar(void)****void wxMenuBar(int n, wxMenu \*menus[], char \*titles[])**

Construct a menu bar. In the second form, the calling program must have created an array of menus and an array of titles. Do not use the submenus again after this call.

**wxMenuBar::~~wxMenuBar****void ~wxMenuBar(void)**

Destructor, destroying the menu bar and removing it from the parent frame (if any).

**wxMenuBar::Append****void Append(wxMenu \*menu, char \*title)**

Adds the item to the end of the menu bar. Do not use *menu* after this call: it will be deallocated by `wxWindows`.

**wxMenuBar::Check****void Check(int id, Bool flag)**

If *flag* is TRUE, checks the given menu item, else unchecks it. MS Windows, Motif only. Only use this when the menu bar has been associated with a frame; otherwise, use the wxMenu equivalent call.

**wxMenuBar::Checked****Bool Checked(int id)**

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

**wxMenuBar::Enable****void Enable(int id, Bool flag)**

If *flag* is TRUE, enables the given menu item, else disables it (greys it). MS Windows, Motif only. Only use this when the menu bar has been associated with a frame; otherwise, use the wxMenu equivalent call.

**wxMenuBar::EnableTop****void EnableTop(int pos, Bool flag)**

If *flag* is TRUE, enables the menu at the given position, else disables it (greys it). Only use this when the menu bar has been associated with a frame.

**wxMenuBar::FindMenuItem****int FindMenuItem(char \*menuString, char \*itemString)**

Finds the menu item id for a menu name/menu item string pair, or -1 if none found. Any special menu codes are stripped out of source and target strings before matching.

**wxMenuBar::FindItemById****wxMenuItem \* FindItemById(int itemId)**

Finds the menu item object associated with the given menu item identifier, returning NULL if not found.

**wxMenuBar::GetHelpString****char \* GetHelpString(int itemId)**

Gets the help string associated with the menu item identifier (or NULL if there is no help string or the item was not found).

### **wxMenuBar::GetLabel**

**char \* GetLabel(int itemId)**

Returns a temporary pointer to the label of the given menu item. Use only after the menubar has been associated with a frame with **wxFrame::SetMenuBar**.

### **wxMenuBar::GetLabelTop**

**char \* GetLabelTop(int pos)**

Returns a temporary pointer to the label of the given top-level menu. *pos* is the position of a menu on the menu bar. Use only after the menubar has been associated with a frame with **wxFrame::SetMenuBar**.

### **wxMenuBar::SetHelpString**

**void SetHelpString(int itemId, char \*helpString)**

Sets the help string associated with the menu item identifier.

### **wxMenuBar::SetLabel**

**void SetLabel(int itemId, char \*label)**

Sets the label of the given menu item. Use only after the menubar has been associated with a frame with **wxFrame::SetMenuBar**.

### **wxMenuBar::SetLabelTop**

**void SetLabelTop(int pos, char \*label)**

Sets the label of the given top-level menu. *pos* is the position of a menu on the menu bar. Use only after the menubar has been associated with a frame with **wxFrame::SetMenuBar**.

## **9.65. wxMessage: wxItem**

A message is a simple line of text which may be displayed in a panel. It does not respond to mouse clicks.

### **wxMessage::wxMessage**

**void wxMessage(void)**

Constructor, used for deriving classes.

```
void wxMessage(wxPanel *panel, char *message, int x = -1, int y = -1,  
    long style = 0, char *name = "message")
```

```
void wxMessage(wxPanel *panel, char *message, int x, int y,  
    int x, int y, long style, char *name)
```

```
void wxMessage(wxPanel *panel, wxBitmap *bitmap, int x = -1, int y = -1,  
    long style = 0, char *name = "message")
```

```
void wxMessage(wxPanel *panel, wxBitmap *bitmap, int x, int y,  
    int x, int y, long style, char *name)
```

Creates and displays a simple text message. *message* is the initial value of the message.

The parameters *x* and *y* are used to specify an absolute position, or a position after the previous panel item if omitted or default.

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual message items.

### **wxMessage::~~wxMessage**

```
void ~wxMessage(void)
```

Destroys the message.

### **wxMessage::Create**

```
Bool Create(wxPanel *panel, char *message, int x = -1, int y = -1,  
    int width = -1, int height = -1, long style = 0, char *name = "message")
```

```
Bool Create(wxPanel *panel, wxBitmap *bitmap, int x = -1, int y = -1,  
    int width = -1, int height = -1, long style = 0, char *name = "message")
```

Creates the message for two-step construction. Derived classes should call or replace this function. See *wxMessage::wxMessage* (page 211) for further details.

## **9.66. wxMetaFile: wxObject**

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*.

### **wxMetaFile::wxMetaFile**

```
void wxMetaFile(char *filename = NULL)
```

Constructor. If a filename is given, the Windows disk metafile is read in. Check whether this was performed successfully by using the *wxMetaFile::Ok* (page 213) member.

**wxMetaFile::~~wxMetaFile****void ~wxMetaFile(void)**

Destructor.

**wxMetaFile::Ok****Bool Ok(void)**

Returns TRUE if the metafile is valid.

**wxMetaFile::Play****Bool Play(wxDC \*dc)**

Plays the metafile into the given device context, returning TRUE if successful.

**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 an 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;
    }
}
```

**9.67. wxMetaFileDC: wxDC**

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` (page 214) 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` (page 212)).

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` (page 336) after closing your disk-based metafile device context.

### **wxMetaFileDC::wxMetaFileDC**

**void wxMetaFileDC(char \*filename = NULL)**

Constructor. If no filename is passed, the metafile is created in memory.

### **wxMetaFileDC::~~wxMetaFileDC**

**void ~wxMetaFileDC(void)**

Destructor.

### **wxMetaFileDC::Close**

**wxMetaFile \* Close(void)**

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).

## **9.68. wxMouseEvent: wxEvent**

This event class contains information about mouse events, particularly events received by canvases. See `wxCanvas::OnEvent` (page 63).

### **wxMouseEvent::controlDown**

**Bool controlDown**

TRUE if control key is pressed down.

### **wxMouseEvent::leftDown**

**Bool leftDown**

TRUE if the left mouse button is currently pressed down.

### **wxMouseEvent::middleDown**

**Bool middleDown**

TRUE if the middle mouse button is currently pressed down.

**wxMouseEvent::rightDown****Bool rightDown**

TRUE if the right mouse button is currently pressed down.

**wxMouseEvent::leftDown****Bool leftDown**

TRUE if the left mouse button is currently pressed down.

**wxMouseEvent::shiftDown****Bool shiftDown**

TRUE if shift is pressed down.

**wxMouseEvent::x****float x**

X-coordinate of the event.

**wxMouseEvent::y****float y**

Y-coordinate of the event.

**wxMouseEvent::wxMouseEvent****void wxMouseEvent(WXTYPE *mouseEventType*)**

Constructor. Valid event types are:

- **wxEVENT\_TYPE\_ENTER\_WINDOW**
- **wxEVENT\_TYPE\_LEAVE\_WINDOW**
- **wxEVENT\_TYPE\_LEFT\_DOWN**
- **wxEVENT\_TYPE\_LEFT\_UP**
- **wxEVENT\_TYPE\_LEFT\_DCLICK**
- **wxEVENT\_TYPE\_MIDDLE\_DOWN**
- **wxEVENT\_TYPE\_MIDDLE\_UP**
- **wxEVENT\_TYPE\_MIDDLE\_DCLICK**
- **wxEVENT\_TYPE\_RIGHT\_DOWN**
- **wxEVENT\_TYPE\_RIGHT\_UP**
- **wxEVENT\_TYPE\_RIGHT\_DCLICK**
- **wxEVENT\_TYPE\_MOTION**



Note that double clicks in canvases are only processed if you call `wxWindow::SetDoubleClick` (page 325) with a value of `TRUE`.

### **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.

### **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).

Under MS Windows, a double click always follows a down-up sequence. On the other supported platforms (WIN32, Motif, but not XView) the double click event occurs on its own. See also `wxCanvas::AllowDoubleClick` (page 57).

### **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).

### **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).

### **wxMouseEvent::ControlDown**

**Bool ControlDown**(void)

Returns `TRUE` if the control key was down at the time of the event.

### **wxMouseEvent::Dragging**

**Bool Dragging(void)**

Returns TRUE if this was a dragging event (motion while a button is depressed).

**wxMouseEvent::Entering****Bool Entering(void)**

Returns TRUE if the mouse was entering the canvas (MS Windows and Motif).

See also *wxMouseEvent::Leaving* (page 217).

**wxMouseEvent::IsButton****Bool IsButton(void)**

Returns TRUE if the event was a mouse button event (not necessarily a button down event - that may be tested using *ButtonDown*).

**wxMouseEvent::Leaving****Bool Leaving(void)**

Returns TRUE if the mouse was leaving the canvas (MS Windows and Motif).

See also *wxMouseEvent::Entering* (page 217).

**wxMouseEvent::LeftDClick****Bool LeftDClick(void)**

Returns TRUE if the event was a left double click.

**wxMouseEvent::LeftDown****Bool LeftDown(void)**

Returns TRUE if the left mouse button changed to down.

**wxMouseEvent::LeftIsDown****Bool LeftIsDown(void)**

Returns TRUE if the left mouse button is currently down, independent of the current event type.

**wxMouseEvent::LeftUp**

**Bool LeftUp(void)**

Returns TRUE if the left mouse button changed to up.

**wxMouseEvent::MiddleDClick****Bool MiddleDClick(void)**

Returns TRUE if the event was a middle double click.

**wxMouseEvent::MiddleDown****Bool MiddleDown(void)**

Returns TRUE if the middle mouse button changed to down.

**wxMouseEvent::MiddleIsDown****Bool MiddleIsDown(void)**

Returns TRUE if the middle mouse button is currently down, independent of the current event type.

**wxMouseEvent::MiddleUp****Bool MiddleUp(void)**

Returns TRUE if the middle mouse button changed to up.

**wxMouseEvent::Moving****Bool Moving(void)**

Returns TRUE if this was a motion event (no buttons depressed).

**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 canvas, 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 canvas mouse event), use **wxDC::LogicalToDeviceX** and **wxDC::LogicalToDeviceY**.

For example, the following code calculates screen pixel coordinates from the frame position, canvas view start (assuming the canvas 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);
canvas->ViewStart(&x1, &y1);
int mouse_x = (int)(canvas->GetDC()->LogicalToDeviceX(event_x + x -
x1);
int mouse_y = (int)(canvas->GetDC()->LogicalToDeviceY(event_y + y -
y1);

char *choice = wxGetSingleChoice("Menu", "Pick a node action",
                                no_choices, choices, frame, mouse_x,
mouse_y);
```

### **wxMouseEvent::RightDClick**

#### **Bool RightDClick(void)**

Returns TRUE if the event was a right double click.

### **wxMouseEvent::RightDown**

#### **Bool RightDown(void)**

Returns TRUE if the right mouse button changed to down.

### **wxMouseEvent::RightIsDown**

#### **Bool RightIsDown(void)**

Returns TRUE if the right mouse button is currently down, independent of the current event type.

### **wxMouseEvent::RightUp**

#### **Bool RightUp(void)**

Returns TRUE if the right mouse button changed to up.

### **wxMouseEvent::ShiftDown**

#### **Bool ShiftDown(void)**

Returns TRUE if the shift key was down at the time of the event.

## **9.69. wxMultiText: wxText**

Members as for **wxText**, but allowing multiple lines of text.

The *style* parameter can be a bit list of the following:

**wxHSCROLL** A horizontal scrollbar will be displayed. If **wxHSCROLL** is omitted, only a vertical scrollbar is displayed, and lines will be wrapped. This parameter is ignored under XView.

**wxTE\_READONLY** The text is read-only (not XView).

**wxFIXED\_LENGTH** Allows the values of a column of items to be left-aligned. Create an item with this style, and pad out your labels with spaces to the same length. The item labels will initially be created with a string of identical characters, positioning all the values at the same x-position. Then the real label is restored.

### **wxMultiText::GetLineLength**

**int GetLineLength(long lineNo)**

Returns the number of characters in the given line. Windows and Motif only.

### **wxMultiText::GetLineText**

**int GetLineText(long lineNo, char \*buf)**

Copies the text at the given line into *buf*, returning the number of characters copied. Windows and Motif only.

### **wxMultiText::GetNumberOfLines**

**long GetNumberOfLines(void)**

Returns the number of lines. Windows and Motif only.

### **wxMultiText::GetValue**

**char \* GetValue(void)**

**void GetValue(char \*buffer, int bufferSize)**

The first form gets a temporary pointer to the current value; copy this for long-term use. The second form copies the value into a buffer, for situations where a lot of text is returned (more than the capacity of the small buffer used for the first form - about 1000 characters).

### **wxMultiText::PositionToXY**

**void PositionToXY(long pos, long x, long y)**

Converts index position to character and line position. Windows and Motif only.

### **wxMultiText::ShowPosition**

**void ShowPosition(long pos)**

Scrolls the text so that *pos* is visible. Windows and Motif only.

### **wxMultiText::XYToPosition**

**long XYToPosition(long x, long y)**

Converts character and line position to index position. Windows and Motif only.

### **9.70. wxNode: wxObject**

A node structure used in linked lists (see *wxList* (page 197)).

#### **wxNode::Data**

**wxObject \* Data(void)**

Retrieves the client data pointer associated with the node. This will have to be cast to the correct type.

#### **wxNode::Next**

**wxNode \* Next(void)**

Retrieves the next node (NULL if at end of list).

#### **wxNode::Previous**

**wxNode \* Previous(void)**

Retrieves the previous node (NULL if at start of list).

#### **wxNode::SetData**

**void SetData(wxObject \*data)**

Sets the data associated with the node (usually the pointer will have been set when the node was created).

### **9.71. 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.

From *wxWindows* 1.62, *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.

See also *wxClassInfo* (page 72).

**wxObject::\_\_type****WXTYPE \_\_type**

*OBSOLETE MEMBER.* Please see the *run time class information* (page 370) for an alternative type system.

Data member used for storing dynamic type information. Most wxWindows classes set this member to an appropriate type, which may be overridden in derived classes. Optionally set this in your constructor. The type may be checked using `::wxSubType` (page 350).

Note the double underscore prefixing this name, in order to minimize clashes with application code. There is no accessor function for this member, and its scope is public.

See also *wxTypeTree* (page 313).

*NOTE:* This typing scheme will soon become obsolete since there is now a better system using `DECLARE...` and `IMPLEMENT...` macros to register run-time type information.

**wxObject::Dump****void Dump(ostream& stream)**

A virtual function that should be redefined by derived classes to allow dumping of memory states. 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.

**wxObject::GetClassInfo****wxClassInfo \* GetClassInfo(void)**

This virtual function is redefined for every class that requires run-time type information.

**wxObject::IsKindOf****Bool IsKindOf(wxClassInfo \*info)**

Determines whether this class is a subclass of (or the same class as) the given class. E.g.:

```
Bool tmp = obj->IsKindOf(CLASSINFO(wxFram));
```

**wxObject::LoadObject****istream& LoadObject(istream& stream)**

The basis for a future persistent storage scheme.

### **wxObject::SaveObject**

**ostream& SaveObject(istream& stream)**

The basis for a future persistent storage scheme.

### **wxObject::operator new**

**void \* new(size\_t size, char \*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.

### **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.

## **9.72. wxPageSetupData: wxObject**

This class holds a variety of information related to *wxPageSetupDialog* (page 227).

### **wxPageSetupData::wxPageSetupData**

**void wxPageSetupData(void)**

Constructor.

### **wxPageSetupData::~~wxPageSetupData**

**void ~wxPageSetupData(void)**

Destructor.

### **wxPageSetupData::EnableHelp**

**void EnableHelp(Bool flag)**

Enables or disables the 'Help' button (Windows only).

### **wxPageSetupData::EnableMargins**



**void EnableMargins(Bool flag)**

Enables or disables the margin controls (Windows only).

**wxPageSetupData::EnableOrientation**

**void EnableOrientation(Bool flag)**

Enables or disables the orientation control (Windows only).

**wxPageSetupData::EnablePaper**

**void EnablePaper(Bool flag)**

Enables or disables the paper size control (Windows only).

**wxPageSetupData::EnablePrinter**

**void EnablePrinter(Bool flag)**

Enables or disables the **Printer** button, which invokes a printer setup dialog.

**wxPageSetupData::GetPaperSize**

**wxPoint GetPaperSize(void)**

Returns the paper size in millimetres.

**wxPageSetupData::GetMarginTopLeft**

**wxPoint GetMarginTopLeft(void)**

Returns the left (x) and top (y) margins.

**wxPageSetupData::GetMarginBottomRight**

**wxPoint GetMarginBottomRight(void)**

Returns the right (x) and bottom (y) margins.

**wxPageSetupData::GetMinMarginTopLeft**

**wxPoint GetMinMarginTopLeft(void)**

Returns the left (x) and top (y) minimum margins the user can enter (Windows only).

**wxPageSetupData::GetMinMarginBottomRight****wxPoint GetMinMarginBottomRight(void)**

Returns the right (x) and bottom (y) minimum margins the user can enter (Windows only).

**wxPageSetupData::GetOrientation****int GetOrientation(void)**

Returns the orientation, which can be wxPORTRAIT or wxLANDSCAPE.

**wxPageSetupData::GetDefaultMinMargins****Bool GetDefaultMinMargins(void)**

Returns TRUE if the page setup dialog will take its minimum margin values from the currently selected printer properties. Windows only.

**wxPageSetupData::GetEnableMargins****Bool GetEnableMargins(void)**

Returns TRUE if the margin controls are enabled (Windows only).

**wxPageSetupData::GetEnableOrientation****Bool GetEnableOrientation(void)**

Returns TRUE if the orientation control is enabled (Windows only).

**wxPageSetupData::GetEnablePaper****Bool GetEnablePaper(void)**

Returns TRUE if the paper size control is enabled (Windows only).

**wxPageSetupData::GetEnablePrinter****Bool GetEnablePrinter(void)**

Returns TRUE if the printer setup button is enabled.

**wxPageSetupData::GetEnableHelp****Bool GetEnableHelp(void)**

Returns TRUE if the printer setup button is enabled.

### **wxPageSetupData::GetDefaultInfo**

**Bool GetDefaultInfo(void)**

Returns TRUE if the dialog will simply return default printer information (such as orientation) instead of showing a dialog. Windows only.

### **wxPageSetupData::SetPaperSize**

**void SetPaperSize(const wxPoint& size)**

Sets the paper size in millimetres.

### **wxPageSetupData::SetMarginTopLeft**

**void GetMarginTopLeft(const wxPoint& pt)**

Sets the left (x) and top (y) margins.

### **wxPageSetupData::SetMarginBottomRight**

**void SetMarginBottomRight(const wxPoint& pt)**

Sets the right (x) and bottom (y) margins.

### **wxPageSetupData::SetMinMarginTopLeft**

**void SetMinMarginTopLeft(const wxPoint& pt)**

Sets the left (x) and top (y) minimum margins the user can enter (Windows only).

### **wxPageSetupData::SetMinMarginBottomRight**

**void SetMinMarginBottomRight(const wxPoint& pt)**

Sets the right (x) and bottom (y) minimum margins the user can enter (Windows only).

### **wxPageSetupData::SetOrientation**

**void SetOrientation(int orientation)**

Sets the orientation, which can be wxPORTRAIT or wxLANDSCAPE.

**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.

**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.

**9.73. wxPageSetupDialog: wxDialogBox**

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* (page 223) object associated with the dialog.

Note that the OK and Cancel buttons do not destroy the dialog; this must be done by the application.

**wxPageSetupDialog::wxPageSetupDialog****void 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.

**wxPageSetupDialog::~~wxPageSetupDialog****void ~wxPageSetupDialog(void)**

Destructor.

**wxPageSetupDialog::GetPageSetupData****wxPageSetupData& GetPageSetupData(void)**

Returns the *page setup data* (page 223) associated with the dialog.

## **wxPageSetupDialog::Show**

### **Bool Show(Bool flag)**

Shows the dialog, returning TRUE if the user pressed Ok, and FALSE otherwise.

## **9.74. wxPanel: wxCanvas**

A panel is a subwindow of a frame in which *panel items* can be placed to allow the user to view and set controls. Panel items include messages, text items, list items, and check boxes. Use **Fit** to fit the panel around its items.

Because wxPanel inherits from wxCanvas (in implementations that permit it, such as XView, Motif, and Windows) it has a device context, and can be drawn on. There are some restrictions, however:

- The following wxCanvas members cannot be used: SetScrollbars, Scroll, GetVirtualSize.
- The device context associated with wxDialogBox behaves slightly differently: drawing to it requires enclosing code in BeginDrawing, EndDrawing calls. This is because under Windows, dialog box device contexts are not 'retained' and settings would be lost if the device context were retrieved and released for each drawing operations.

## **wxPanel::wxPanel**

### **void wxPanel(void)**

Constructor, for deriving classes.

**void wxPanel(wxWindow \*parent, int x = -1, int y = -1, int width = -1, int height = -1, long style = 0, char \*name = "panel")**

Constructor.

The parameters *x*, *y*, *width* and *height* can be omitted on construction if the position and size will later be set (for example by a application frame's **OnSize** callback, or if there is only one subwindow for the frame, in which case the subwindow fills the frame).

The style parameter may be a combination of the following, using the bitwise 'or' operator.

wxBORDER      Draws a thin border around the panel.  
wxUSER\_COLOURS      Under Windows, overrides standard control processing to allow setting of the panel background colour.  
wxVSCROLL      Gives the dialog box a vertical scrollbar (XView only).

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual panels.

The parent window may be a panel in Motif and Windows, but not XView.

## **wxPanel::~~wxPanel**

### **void ~wxPanel(void)**

Destructor. Deletes any panel items before deleting the physical window.

### **wxPanel::Create**

```
void Create(wxWindow *parent, int x = -1, int y = -1, int width = -1, int height = -1,  
            long style = 0, char *name = "panel")
```

Used in two-step panel construction. See *wxPanel::wxPanel* (page 228) for further details.

### **wxPanel::CreateItem**

```
wxItem * CreateItem(wxItemResource *resource, wxResourceTable *table)
```

Virtual function that is called by *wxPanel::LoadFromResource* to create an item from a resource. Override this if you must create items of different class from the usual ones.

### **wxPanel::DrawAllStaticItems**

```
void DrawAllStaticItems(void)
```

Draws all the wxWindows static items associated with this panel. This is experimental code.

### **wxPanel::Fit**

```
void Fit(void)
```

Resize the panel to just fit around the panel items. Also works for dialog boxes.

### **wxPanel::GetButtonFont**

```
wxFont * GetButtonFont(void)
```

Get the current font for drawing panel item values.

### **wxPanel::GetCursor**

```
void GetCursor(int *x, int *y)
```

Gets the current panel 'cursor' position, i.e. where the next panel item will be placed.

### **wxPanel::GetDefaultItem**

```
wxButton * GetDefaultItem(void)
```

Retrieves the default button, previously set with *wxButton::SetDefault* (page 55).

**wxPanel::GetHorizontalSpacing****int GetHorizontalSpacing(void)**

Gets the horizontal spacing for placing items on a panel.

**wxPanel::GetBackgroundColour****wxColour \* GetBackgroundColour(void)**

Gets the default item background colour.

**wxPanel::GetButtonColour****wxColour \* GetButtonColour(void)**

Gets the default item button colour.

**wxPanel::GetLabelColour****wxColour \* GetLabelColour(void)**

Gets the default item label colour.

**wxPanel::GetLabelFont****wxFont \* GetLabelFont(void)**

Get the current font for drawing panel item labels.

**wxPanel::GetPanelDC****wxPanelDC \* GetPanelDC(void)**

Returns the panel device context. You may also get the device context using `wxCanvasDC::GetDC`. Since `wxCanvasDC` and `wxPanelDC` offer the same interface, either call will be adequate to get a suitable device context.

**wxPanel::GetVerticalSpacing****int GetVerticalSpacing(void)**

Gets the vertical spacing for placing items on a panel.

**wxPanel::LoadFromResource**

**Bool LoadFromResource(wxWindow \*parent, char \*name)**

Loads the contents of a panel or dialog box from a wxWindows resource.

See also *wxWindows resource functions* (page 355) and *the wxWindows resource system* (page 414).

**wxPanel::NewLine****void NewLine(void)**

Cause the next item to be positioned at the beginning of the next line, using the current vertical spacing. More than one new line in succession causes extra vertical spacing to be inserted.

**wxPanel::OnCommand****void OnCommand(wxWindow &win, wxCommandEvent &event)**

This member is called for panel items that do not have a callback function of their own. It must be overridden when using wxWindows resources, for example.

See also *wxWindows resource formats* (page 414).

**wxPanel::OnDefaultAction****void OnDefaultAction(wxItem \*item)**

Called when the user initiates the default action for a panel or dialog box, for example by double clicking on a listbox. *item* is the panel item which caused the default action.

The default behaviour for this member is to either send a double click event to the item if it is a listbox, or to retrieve the default button for the panel, and send it a command event as if the user had clicked on the button. This gives default listbox double-click behaviour under Motif and MS Windows. The default code is as follows:

```
void wxbPanel::OnDefaultAction(wxItem *initiatingItem)
{
    if (initiatingItem->IsKindOf(CLASSINFO(wxListBox)) &&
        initiatingItem->callback)
    {
        wxListBox *lbox = (wxListBox *)initiatingItem;
        wxCommandEvent event(wxEVENT_TYPE_LISTBOX_DCLICK_COMMAND);
        event.commandInt = -1;
        if ((lbox->GetWindowStyleFlag() & wxLB_SINGLE) ||
            (lbox->GetSelectionMode() == wxSINGLE))
        {
            event.commandString = copystring(lbox->GetStringSelection());
            event.commandInt = lbox->GetSelection();
            event.clientData =
                lbox->wxListBox::GetClientData(event.commandInt);
        }
        event.eventObject = lbox;
    }
}
```



```
lbox->ProcessCommand(event);

if (event.commandString)
    delete[] event.commandString;
return;
}

wxButton *but = GetDefaultItem();
if (but)
{
    wxCommandEvent event(WXEVENT_TYPE_BUTTON_COMMAND);
    but->Command(event);
}
}
```

### **wxPanel::OnEvent**

**void OnEvent(wxMouseEvent & event)**

Called when the panel receives a mouse event. The default implementation manages panel item dragging and sizing if in user-interface edit mode. It also sends panel mouse clicks to the application-overridable member functions `OnLeftClick` and `OnRightClick`, again only in user-interface edit mode.

See also `wxWindow::SetUserEditMode` (page 327).

### **wxPanel::OnItemEvent**

**void OnItemEvent(wxItem \* item, wxMouseEvent & event)**

Called in user-interface edit mode when the item receives a mouse event. The default implementation manages panel item dragging and sizing.

See also `wxWindow::SetUserEditMode` (page 327).

### **wxPanel::OnItemLeftClick**

**void OnItemLeftClick(int x, int y, int keys)**

Called in user-interface edit mode when the user left-clicks on a panel item. The coordinates (relative to the item) and a flag indicating shift and control key status are passed. `keys` is a bit list of `wxKEY_SHIFT` and `wxKEY_CTRL`.

See also `wxWindow::SetUserEditMode` (page 327).

### **wxPanel::OnItemMove**

**void OnItemMove(wxItem \* item, int x, int y)**

Called in user-interface edit mode when the item has been moved by the user.

See also *wxWindow::SetUserEditMode* (page 327).

### **wxPanel::OnItemRightClick**

**void OnItemRightClick(int x, int y, int keys)**

Called in user-interface edit mode when the user right-clicks on a panel item. The coordinates (relative to the item) and a flag indicating shift and control key status are passed. *keys* is a bit list of *wxKEY\_SHIFT* and *wxKEY\_CTRL*.

See also *wxWindow::SetUserEditMode* (page 327).

### **wxPanel::OnItemSize**

**void OnItemSize(wxItem \* item, int width, int height)**

Called in user-interface edit mode when the item has been resized by the user.

See also *wxWindow::SetUserEditMode* (page 327).

### **wxPanel::OnLeftClick**

**void OnLeftClick(int x, int y, int keys)**

Called in user-interface edit mode when the user left-clicks on the panel background. The coordinates and a flag indicating shift and control key status are passed. *keys* is a bit list of *wxKEY\_SHIFT* and *wxKEY\_CTRL*.

See also *wxWindow::SetUserEditMode* (page 327).

### **wxPanel::OnRightClick**

**void OnRightClick(int x, int y, int keys)**

Called in user-interface edit mode when the user right-clicks on the panel background. The coordinates and a flag indicating shift and control key status are passed. *keys* is a bit list of *wxKEY\_SHIFT* and *wxKEY\_CTRL*.

See also *wxWindow::SetUserEditMode* (page 327).

### **wxPanel::OnPaint**

**void OnPaint(void)**

Sent to the panel when it receives an expose event. If you wish to draw on the panel, you may derive your own class to handle this message.

The standard `wxPanel::OnPaint` implementation contains code to draw custom static items on the panel, and also to draw selection handles for panel items if necessary. If you wish to use this functionality, call `wxPanel::OnPaint` from your own `OnPaint` handler, or call the individual `DrawAllStaticItems` and `PaintSelectionHandles` functions.

### **wxPanel::PaintSelectionHandles**

**void PaintSelectionHandles(void)**

Paints the selection handles for panel items if user interface editing mode is on. This function is called automatically by the default `wxPanel::OnPaint` handler.

See `wxWindow::SetUserEditMode` (page 327).

### **wxPanel::SetHorizontalSpacing**

**void SetHorizontalSpacing(int *sp*)**

Sets the horizontal spacing for placing items on a panel.

### **wxPanel::SetLabelPosition**

**void SetLabelPosition(int *position*)**

Determines the current method of placing labels on panel items: if *position* is `wxHORIZONTAL`, labels are placed to the left of the item value. If *position* is `wxVERTICAL`, the label is placed above the item value. The default behaviour is to have horizontal label placing.

Under MS Windows, this function works for **wxText**, **wxChoice** and **wxListBox**. Under XView, absolute positioning must be used for the `wxVERTICAL` position to work in some cases. This is because of some strange behaviour in XView where setting a horizontal layout orientation but a vertical label position causes items after list box to appear too low on the panel. So, where it is necessary to have vertical labels, use absolute positioning where results are not as expected.

### **wxPanel::SetBackgroundColour**

**void SetBackgroundColour(wxColour& *colour*)**

Specifies the default colour for drawing panel item backgrounds (Motif and Windows).

### **wxPanel::SetButtonColour**

**void SetButtonColour(wxColour& *colour*)**

Specifies the default colour for drawing value text (Motif and Windows). `wxButton` items do not respond to this setting under Windows.

### **wxPanel::SetButtonFont**

**void SetButtonFont(wxFont \*font)**

Specifies the default font for drawing panel item values (Motif and Windows).

### **wxPanel::SetHorizontalSpacing**

**void SetHorizontalSpacing(int sp)**

Sets the horizontal spacing for placing items on a panel.

### **wxPanel::SetLabelColour**

**void SetLabelColour(wxColour& colour)**

Specifies the default colour for drawing panel item labels (Motif and Windows).

### **wxPanel::SetLabelFont**

**void SetLabelFont(wxFont \*font)**

Specifies the font for drawing panel item labels (Motif and Windows).

### **wxPanel::SetVerticalSpacing**

**void SetVerticalSpacing(int sp)**

Sets the vertical spacing for placing items on a panel.

### **wxPanel::Tab**

**void Tab(int pixels)**

Tabs by the given number of pixels.

## **9.75. wxPanelDC: wxDC**

A panel device context is automatically created when a panel or dialog box is created. It can be retrieved from a panel with *wxCanvas::GetDC* (page 61) or *wxPanel::GetPanelDC* (page 230) and then drawn into. See *wxDC* (page 108) for further information on device contexts.

## **9.76. wxPathList: wxList**

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 *FileNameFromPath* global function to extract the filename from the path.

**wxPathList::wxPathList****void wxPathList(void)**

Constructor.

**wxPathList::AddEnvList****void AddEnvList(char \*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.

**wxPathList::Add****void Add(char \*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).

**wxPathList::EnsureFileAccessible****void EnsureFileAccessible(char \*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.

**wxPathList::FindValidPath****char \* FindValidPath(char \*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.

**wxPathList::Member****Bool Member(char \*file)**

TRUE if the path is in the path list (ignoring case).

**9.77. wxPen: wxObject**

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. On a monochrome display, the default behaviour is to show all non-white pens as black. To change this, set the **Colour** member of the device context to TRUE, and select appropriate colours.

The style may be one of `wxSOLID`, `wxDOT`, `wxLONG_DASH`, `wxSHORT_DASH` and `wxDOT_DASH`. The names of these styles should be self explanatory.

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`* (page 239).

## **`wxPen::wxPen`**

**`void wxPen(void)`**

**`void wxPen(wxColour&colour, int width, int style)`**

**`void wxPen(char *colour_name, int width, int style)`**

Constructs a pen, uninitialized, initialized with an RGB colour, a width and a style, or initialized using a colour name, a width and a style. 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`.

## **`wxPen::~~wxPen`**

**`void ~wxPen(void)`**

Destructor, destroying the pen. Note that pens should very rarely be deleted since windows may contain pointers to them. All pens will be deleted when the application terminates.

If you have to delete the pen (for example, you are creating a lot of them), then call *`wxDC::SetPen`* (page 119) with a `NULL` argument to ensure that the old pen is restored, and the current pen is selected out of the device context.

## **`wxPen::GetCap`**

**`int GetCap(void)`**

Returns the pen cap style, which may be one of **`wxCAP_ROUND`**, **`wxCAP_PROJECTING`** and **`wxCAP_BUTT`**. The default is **`wxCAP_ROUND`**.

## **`wxPen::GetColour`**

**`wxColour& GetColour(void)`**

Returns a reference to the pen colour.

**wxPen::GetDashes****int GetDashes(wxDash \*\*dashes)**

Gets an array of dashes (defined as char in X, DWORD under Windows). *dashes* is a pointer to the array (not allocated by the application). The function returns the number of dashes associated with this pen.

**wxPen::GetJoin****int GetJoin(void)**

Returns the pen join style, which may be one of **wxJOIN\_BEVEL**, **wxJOIN\_ROUND** and **wxJOIN\_MITER**. The default is **wxJOIN\_ROUND**.

**wxPen::GetStipple****wxBitmap \* GetStipple(void)**

Gets the stipple bitmap.

**wxPen::GetStyle****int GetStyle(void)**

Returns the pen style.

**wxPen::GetWidth****int GetWidth(void)**

Returns the pen width.

**wxPen::SetCap****void SetCap(intcap\_style)**

Sets the pen cap style, which may be one of **wxCAP\_ROUND**, **wxCAP\_PROJECTING** and **wxCAP\_BUTT**. The default is **wxCAP\_ROUND**.

**wxPen::SetColour****void SetColour(wxColour &colour)****void SetColour(char \*colour\_name)****void SetColour(int red, int green, int blue)**

The pen's colour is changed to the given colour.

### **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.

Sorry, I don't yet have information as to how the dashes work.

### **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**.

### **wxPen::SetStipple**

**void SetStipple(wxBitmap \* *stipple*)**

Sets the bitmap for stippling.

### **wxPen::SetStyle**

**void SetStyle(int *style*)**

Set the pen style (wxSOLID or wxTRANSPARENT).

### **wxPen::SetWidth**

**void SetWidth(int *width*)**

Set the pen width.

## **9.78. wxPenList: wxList**

A pen list is a list containing all pens which have been created. 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 is best 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'.

### **wxPenList::wxPenList**

**void wxPenList(void)**



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

### **wxPenList::AddPen**

**void AddPen(wxPen \*pen)**

Used by wxWindows to add a pen to the list, called in the pen constructor.

### **wxPenList::FindOrCreatePen**

**wxPen \* FindOrCreatePen(wxColour \*colour, int width, int style)**

**wxPen \* FindOrCreatePen(char \*colour\_name, int width, int style)**

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

### **wxPenList::RemovePen**

**void RemovePen(wxPen \*pen)**

Used by wxWindows to remove a pen from the list.

## **9.79. wxPoint: wxObject**

A **wxPoint** is a useful data structure for graphics operations. It simply contains floating point *x* and *y* members. See also *wxIntPoint* (page 191) for an integer version.

### **wxPoint::wxPoint**

**void wxPoint(void)**

**void wxPoint(float x, float y)**

Create a point.

**float x**

**float y**

Members of the **wxPoint** object.

## **9.80. wxPostScriptDC: wxDC**

This defines the wxWindows Encapsulated PostScript device context, which can write PostScript files on any platform. See *wxDC* (page 108) for descriptions of the member functions.

### **wxPostScriptDC::wxPostScriptDC**

```
void wxPostScriptDC(char *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* (page 338) for functions to set and get PostScript printing settings.

### **wxPostScriptDC::GetStream**

```
ostream * GetStream(void)
```

Returns the stream currently being used to write PostScript output. Use this to insert any PostScript code that is outside the scope of `wxPostScriptDC`.

### **9.81. wxPreviewCanvas: wxCanvas**

A preview canvas is the default canvas used by the print preview system to display the preview.

See also *wxPreviewFrame* (page 244), *wxPreviewControlBar* (page 241), *wxPrintPreview* (page 253).

### **wxPreviewCanvas::wxPreviewCanvas**

```
void wxPreviewCanvas(wxPrintPreview *preview, wxWindow *parent, int x = -1, int y = -1,  
    int width = -1, int height = -1,  
    long style = 0, char *name = "canvas")
```

Constructor.

### **wxPreviewCanvas::~~wxPreviewCanvas**

```
void ~wxPreviewCanvas(void)
```

Destructor.

### **wxPreviewCanvas::OnPaint**

```
void OnPaint(void)
```

Calls `wxPrintPreview::PaintPage` to refresh the canvas.

### **9.82. wxPreviewControlBar: wxPanel**

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.

See also *wxPreviewFrame* (page 244), *wxPreviewCanvas* (page 241), *wxPrintPreview* (page 253).

### **wxPreviewControlBar::buttonFlags**

**long buttonFlags**

Protected data member, containing the button flags (see the constructor for details).

### **wxPreviewControlBar::buttonFont**

**static wxFont \* buttonFont**

Protected data member, pointing to the font used for the buttons.

### **wxPreviewControlBar::closeButton**

**wxButton \* closeButton**

Protected data member, pointing to the close button.

### **wxPreviewControlBar::nextPageButton**

**wxButton \* nextPageButton**

Protected data member, pointing to the next page button.

### **wxPreviewControlBar::previousPageButton**

**wxButton \* previousPageButton**

Protected data member, pointing to the previous page button.

### **wxPreviewControlBar::printPreview**

**wxPrintPreview \* printPreview**

Protected data member, pointing to the associated print preview object.

### **wxPreviewControlBar::zoomControl**

**wxChoice \* zoomControl**

Protected data member, pointing to the zoom control.

**wxPreviewControlBar::wxPreviewControlbar**

```
void wxPreviewControlBar(wxPrintPreview *preview, long buttons, wxWindow *parent, int x  
= -1, int y = -1, int width = -1, int height = -1,  
long style = 0, char *name = "panel")
```

Constructor.

The buttons parameter may be a combination of the following, using the bitwise 'or' operator.

```
wxPREVIEW_PRINT    Create a print button.  
wxPREVIEW_NEXT     Create a next page button.  
wxPREVIEW_PREVIOUS Create a previous page button.  
wxPREVIEW_ZOOM     Create a zoom control.  
wxPREVIEW_DEFAULT  Equivalent to a combination of wxPREVIEW_PREVIOUS,  
                   wxPREVIEW_NEXT and wxPREVIEW_ZOOM.
```

**wxPreviewControlBar::~~wxPreviewControlBar**

```
void ~wxPreviewControlBar(void)
```

Destructor.

**wxPreviewControlBar::CreateButtons**

```
void CreateButtons(void)
```

Creates buttons, according to value of the button style flags.

**wxPreviewControlBar::GetPrintPreview**

```
wxPrintPreview * GetPrintPreview(void)
```

Gets the print preview object associated with the control bar.

**wxPreviewControlBar::GetZoomControl**

```
int GetZoomControl(void)
```

Gets the current zoom setting in percent.

**wxPreviewControlBar::OnPaint**

```
void OnPaint(void)
```

Draws a black border on the bottom of the control.

**wxPreviewControlBar::SetZoomControl****void SetZoomControl(int percent)**

Sets the zoom control.

**9.83. wxPreviewFrame: wxFrame**

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.

See also *wxPreviewCanvas* (page 241), *wxPreviewControlBar* (page 241), *wxPrintPreview* (page 253).

**wxPreviewFrame::controlBar****wxPreviewControlBar \* controlBar**

Protected data member, pointing to the preview control bar.

**wxPreviewFrame::previewCanvas****wxCanvas \* previewCanvas**

Protected data member, pointing to the preview canvas.

**wxPreviewFrame::printPreview****wxPrintPreview \* printPreview**

Protected data member, pointing to the print preview object.

**wxPreviewFrame::wxPreviewFrame****void wxPreviewFrame(wxPrintPreview \*preview, wxFrame \*parent, char \*title, int x = -1, int y = -1, int width = -1, int height = -1, long style = wxSDI | wxDEFAULT\_FRAME, char \*name = "frame")**

Constructor. Pass a print preview object plus other normal frame arguments.

**wxPreviewFrame::~~wxPreviewFrame****void ~wxPreviewFrame(void)**

Destructor.

**wxPreviewFrame::CreateControlBar**

**void CreateControlBar(void)**

Creates a wxPreviewControlBar. Override this function to allow a user-defined preview control bar object to be created.

**wxPreviewFrame::CreateCanvas****void CreateCanvas(void)**

Creates a wxPreviewCanvas. Override this function to allow a user-defined preview canvas object to be created.

**wxPreviewFrame::Initialize****void Initialize(void)**

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.

**wxPreviewFrame::OnClose****Bool OnClose(void)**

Enables the other frames in the application, and deletes the print preview object, implicitly deleting any printout objects associated with the print preview object.

**9.84. wxPrintData: wxObject**

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

**wxPrintData::wxPrintData****void wxPrintData(void)**

Constructor.

**wxPrintData::~wxPrintData****void ~wxPrintData(void)**

Destructor.

**wxPrintData::EnableHelp****void EnableHelp(Bool flag)**

Enables or disables the 'Help' button.

### **wxPrintData::EnablePageNumbers**

**void EnablePageNumbers(Bool *flag*)**

Enables or disables the 'Page numbers' controls.

### **wxPrintData::EnablePrintToFile**

**void EnablePrintToFile(Bool *flag*)**

Enables or disables the 'Print to file' checkbox.

### **wxPrintData::EnableSelection**

**void EnableSelection(Bool *flag*)**

Enables or disables the 'Selection' radio button.

### **wxPrintData::GetAllPages**

**Bool GetAllPages(void)**

Returns TRUE if the user requested that all pages be printed.

### **wxPrintData::GetCollate**

**Bool GetCollate(void)**

Returns TRUE if the user requested that the document(s) be collated.

### **wxPrintData::GetFromPage**

**int GetFromPage(void)**

Returns the *from* page number, as entered by the user.

### **wxPrintData::GetMaxPage**

**int GetMaxPage(void)**

Returns the *maximum* page number.

### **wxPrintData::GetMinPage**

**int GetMinPage(void)**

Returns the *minimum* page number.

**wxPrintData::GetNoCopies**

**int GetNoCopies(void)**

Returns the number of copies requested by the user.

**wxPrintData::GetToPage**

**int GetToPage(void)**

Returns the *to* page number, as entered by the user.

**wxPrintData::SetCollate**

**void SetCollate(Bool flag)**

Sets the 'Collate' checkbox to TRUE or FALSE.

**wxPrintData::SetFromPage**

**void SetFromPage(int page)**

Sets the *from* page number.

**wxPrintData::SetMaxPage**

**void SetMaxPage(int page)**

Sets the *maximum* page number.

**wxPrintData::SetMinPage**

**void SetMinPage(int page)**

Sets the *minimum* page number.

**wxPrintData::SetNoCopies**

**void SetNoCopies(int n)**

Sets the default number of copies to be printed out.



**wxPrintData::SetPrintToFile****void SetPrintToFile(Bool *flag*)**

Sets the 'Print to file' checkbox to TRUE or FALSE.

**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).

**wxPrintData::SetToPage****void SetToPage(int *page*)**

Sets the *to* page number.

**9.85. wxPrintDialog: wxDialogBox**

See also *Overview* (page 387)

This class represents the print and print setup common dialogs. You may obtain a *wxPrinterDC* (page 250) device context from a successfully dismissed print dialog.

**wxPrintDialog::wxPrintDialog****void 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.

**wxPrintDialog::~~wxPrintDialog****void ~wxPrintDialog(void)**

Destructor. If *wxPrintDialog::GetPrintDC* has *not* been called, the device context obtained by the dialog (if any) will be deleted.

**wxPrintDialog::GetPrintData****wxPrintData& GetPrintData(void)**

Returns the *print data* (page 245) associated with the print dialog.

**wxPrintDialog::GetPrintDC**

**wxDC \* GetPrintDC(void)**

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.

**wxPrintDialog::Show****Bool Show(Bool flag)**

Shows the dialog, returning TRUE if the user pressed Ok, and FALSE otherwise. After this function is called, a device context may be retrievable using `wxPrintDialog::GetDC`.

**9.86. wxPrinter: wxObject**

See also *Printing framework overview* (page 377)

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.

See also `wxPrinterDC` (page 250), `wxPrintDialog` (page 248), `wxPrintout` (page 251), `wxPrintPreview` (page 253).

**wxPrinter::wxPrinter****void 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.

**wxPrinter::~~wxPrinter****void ~wxPrinter(void)**

Destructor.

**wxPrinter::Abort****Bool Abort(void)**

Returns TRUE if the user has aborted the print job.

**wxPrinter::CreateAbortWindow****void CreateAbortWindow(wxWindow \*parent, wxPrintout \*printout)**

Creates the default printing abort window, with a cancel button.

**wxPrinter::GetPrintData****wxPrintData& GetPrintData(void)**

Returns the *print data* (page 245) associated with the printer object.

**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).

**wxPrinter::PrintDialog****Bool PrintDialog(wxWindow \*parent)**

Invokes the print dialog.

**wxPrinter::ReportError****void ReportError(wxWindow \*parent, wxPrintout \*printout, char \*message)**

Default error-reporting function.

**wxPrinter::Setup****void Setup(wxWindow \*parent)**

Invokes the print setup dialog.

**9.87. wxPrinterDC: wxDC**

A printer device context is specific to Windows, and allows access to any printer with a Windows driver. See wxDC (page 108) for further information on device contexts, and wxDC::GetSize (page 114) for advice on achieving the correct scaling for the page.

**wxPrinterDC::wxPrinterDC****void wxPrinterDC(char \*driver, char \*device, char \*output, Bool interactive = TRUE)**

Constructor. With three NULLs, 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.

### 9.88. wxPrintout: wxObject

See also *Printing framework overview* (page 377)

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.

See also *wxPrinterDC* (page 250), *wxPrintDialog* (page 248), *wxPrinter* (page 249), *wxPrintPreview* (page 253).

#### **wxPrintout::wxPrintout**

**void wxPrintout(char \*title = "Printout")**

Constructor. Pass an optional title argument (currently unused).

#### **wxPrintout::~~wxPrintout**

**void ~wxPrintout(void)**

Destructor.

#### **wxPrintout::GetDC**

**wxDC \* GetDC(void)**

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.

#### **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.

#### **wxPrintout::GetPageSizeMM**

**void GetPageSizeMM(int \*w, int \*h)**

Returns the size of the printer page in millimetres.

**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` (page 114) 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.

**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.

**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.

**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.

**wxPrintout::IsPreview****Bool IsPreview(void)**

Returns TRUE if the printout is currently being used for previewing.

**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`.

**wxPrintout::OnEndDocument****void OnEndDocument(void)**

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.

**wxPrintout::OnBeginPrinting****void OnBeginPrinting(void)**

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).

**wxPrintout::OnEndPrinting****void OnEndPrinting(void)**

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).

**wxPrintout::OnPreparePrinting****void OnPreparePrinting(void)**

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.

**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.

**9.89. wxPrintPreview: wxObject**

See also *Printing framework overview* (page 377)

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.

See also *wxPrinterDC* (page 250), *wxPrintDialog* (page 248), *wxPrintout* (page 251), *wxPrinter* (page 249), *wxPreviewCanvas* (page 241), *wxPreviewControlBar* (page 241), *wxPreviewFrame*

(page 244).

### **wxPrintPreview::wxPrintPreview**

**void 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).

### **wxPrintPreview::~~wxPrintPreview**

**void ~wxPrinter(void)**

Destructor. Deletes both print preview objects, so do not destroy these objects in your application.

### **wxPrintPreview::DrawBlankPage**

**Bool DrawBlankPage(wxCanvas \*canvas)**

Draws a representation of the blank page into the canvas. Used internally.

### **wxPrintPreview::GetCanvas**

**wxCanvas \* GetCanvas(void)**

Gets the canvas used for displaying the print preview image.

### **wxPrintPreview::GetCurrentPage**

**int GetCurrentPage(void)**

Gets the page currently being previewed.

### **wxPrintPreview::GetFrame**

**wxFrame \* GetFrame(void)**

Gets the frame used for displaying the print preview canvas and control bar.

### **wxPrintPreview::GetMaxPage**

**int GetMaxPage(void)**

Returns the maximum page number.

### **wxPrintPreview::GetMinPage**

**int GetMinPage(void)**

Returns the minimum page number.

### **wxPrintPreview::GetPrintData**

**wxPrintData& GetPrintData(void)**

Returns a reference to the internal print data.

### **wxPrintPreview::GetPrintout**

**wxPrintout \* GetPrintout(void)**

Gets the preview printout object associated with the wxPrintPreview object.

### **wxPrintPreview::GetPrintoutForPrinting**

**wxPrintout \* GetPrintoutForPrinting(void)**

Gets the printout object to be used for printing from within the preview interface, or NULL if none exists.

### **wxPrintPreview::Ok**

**Bool Ok(void)**

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).

### **wxPrintPreview::PaintPage**

**Bool PaintPage(wxCanvas \*canvas)**

This refreshes the preview canvas with the preview image. It must be called from the preview canvas's OnPaint member.



The implementation simply blits the preview bitmap onto the canvas, creating a new preview bitmap if none exists.

### **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.

### **wxPrintPreview::RenderPage**

**Bool RenderPage**(**int** *pageNum*)

Renders a page into a wxMemoryDC. Used internally by wxPrintPreview.

### **wxPrintPreview::SetCanvas**

**void SetCanvas**(**wxCanvas** \**canvas*)

Sets the canvas to be used for displaying the print preview image.

### **wxPrintPreview::SetCurrentPage**

**void SetCurrentPage**(**int** *pageNum*)

Sets the current page to be previewed.

### **wxPrintPreview::SetFrame**

**void SetFrame**(**wxFrame** \**frame*)

Sets the frame to be used for displaying the print preview canvas and control bar.

### **wxPrintPreview::SetPrintout**

**void SetPrintout**(**wxPrintout** \**printout*)

Associates a printout object with the wxPrintPreview object.

### **wxPrintPreview::SetZoom**

**void SetZoom**(**int** *percent*)

Sets the percentage preview zoom, and refreshes the preview canvas accordingly.

## **9.90. wxQueryCol: wxObject**

See also *Overview* (page 395)

Every ODBC data column is represented by an instance of this class.

### **wxQueryCol::wxQueryCol**

**void wxQueryCol(void)**

Constructor. Sets the attributes of the column to default values.

### **wxQueryCol::~~wxQueryCol**

**void ~wxQueryCol(void)**

Destructor. Deletes the wxQueryField list.

### **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).

### **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.

### **wxQueryCol::GetData**

**void \* GetData(int field)**

Returns a pointer to the data of the field.

### **wxQueryCol::GetName**

**char \* GetName(void)**

Returns the name of a column.

### **wxQueryCol::GetType**

**short GetType(void)**

Returns the data type of a column.

**wxQueryCol::GetSize****long GetSize(int field)**

Return the size of the data of the field field.

**wxQueryCol::IsRowDirty****Bool IsRowDirty(int field)**

Returns TRUE if the given field has been changed, but not saved.

**wxQueryCol::IsNullable****Bool IsNullable(void)** Returns TRUE if a column may contain no data.**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.

**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.

**wxQueryCol::SetName****void SetName(char \*name)**

Sets the name of a column. Only useful when creating new tables or appending columns.

**wxQueryCol::SetNullable****void SetNullable(Bool nullable)**

Determines whether a column may contain no data. Only useful when creating new tables or appending columns.

**wxQueryCol::SetFieldDirty****void SetFieldDirty(int field, Bool dirty = TRUE)**

Sets the dirty tag of a given field.

**wxQueryCol::SetType**

**void SetType(short type)** Sets the data type of a column. Only useful when creating new tables or appending columns.

**9.91. wxQueryField: wxObject**

See also *Overview* (page 395)

Represents the data item for one or several columns.

**wxQueryField::wxQueryField**

**void wxQueryField(void)**

Constructor. Sets type and size of the field to default values.

**wxQueryField::~~wxQueryField**

**void ~wxQueryField(void)**

Destructor. Frees the associated memory depending on the field type.

**wxQueryField::AllocData**

**Bool AllocData(void)**

Allocates memory depending on the size and type of the field.

**wxQueryField::ClearData**

**void ClearData(void)**

Deletes the contents of the field buffer without deallocating the memory.

**wxQueryField::GetData**

**void \* GetData(void)**

Returns a pointer to the field buffer.

**wxQueryField::GetSize**

**long GetSize(void)**

Returns the size of the field buffer.

**wxQueryField::GetType****short GetType(void)**

Returns the type of the field (currently SQL\_CHAR, SQL\_VARCHAR or SQL\_INTEGER).

**wxQueryField::IsDirty****Bool IsDirty(void)**

Returns TRUE if the data of a field has been changed, but not saved.

**wxQueryField::SetData****Bool SetData(void \*data, long sz)**

Allocates memory of the size sz and copies the contents of d into the field buffer.

**wxQueryField::SetDirty****void SetDirty(Bool dirty = TRUE)**

Sets the dirty tag of a field.

**wxQueryField::SetSize****void SetSize(long size)**

Resizes the field buffer. Stored data will be lost.

**wxQueryField::SetType****void SetType(short type)**

Sets the type of the field. Currently the types SQL\_CHAR, SQL\_VARCHAR and SQL\_INTEGER are supported.

**9.92. wxRadioBox: wxItem**

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.

**wxRadioBox::wxRadioBox****void wxRadioBox(void)**

Constructor, for use by derived classes.

**void wxRadioBox(wxPanel \*parent, wxFunction func, char \*label,  
int x = -1, int y = -1, int width = -1, int height = -1,  
int n, char \*choices[], int majorDim = 0, long style = wxHORIZONTAL, char \*name =  
"radioBox")****void wxRadioBox(wxPanel \*parent, wxFunction func, char \*label,**

```
int x = -1, int y = -1, int width = -1, int height = -1,  
int n, wxBitmap *choices[], int majorDim = 0, long style = wxHORIZONTAL, char *name =  
"radioBox")
```

Constructor, creating and showing a radiobox.

*func* may be NULL; otherwise it is used as the callback for the radiobox. Note that the cast (*wxFunction*) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration.

If *label* is non-NULL, it will be used to label the radiobox.

The parameters *x* and *y* are used to specify an absolute position, or a position after the previous panel item if omitted or default.

If *width* or *height* are omitted (or are less than zero), an appropriate size will be used for the radiobox.

*n* is the number of possible choices, and *choices* is an array of strings or bitmaps of size *n*. *wxWindows* allocates its own memory for these strings so the calling program must deallocate the array itself.

*majorDim* specifies the number of rows (if style is *wxVERTICAL*) or columns (if style is *wxHORIZONTAL*) for a two-dimensional radiobox.

*style* specifies a bitwise-or list of styles. Specify *wxVERTICAL* to lay out a two-dimensional radiobox in columns of specified *majorDim* height, or *wxHORIZONTAL* to lay it out in rows.

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual radioboxes.

## **wxRadioBox::~~wxRadioBox**

```
void ~wxRadioBox(void)
```

Destructor, destroying the radiobox item.

## **wxRadioBox::Create**

```
Bool Create(wxPanel *parent, wxFunction func, char *label,  
    int x = -1, int y = -1, int width = -1, int height = -1,  
    int n, char *choices[], int majorDim = 0, long style = wxHORIZONTAL, char *name =  
    "radioBox")
```

```
Bool Create(wxPanel *parent, wxFunction func, char *label,  
    int x = -1, int y = -1, int width = -1, int height = -1,  
    int n, wxBitmap *choices[], int majorDim = 0, long style = wxHORIZONTAL, char *name =  
    "radioBox")
```

Creates the radiobox for two-step construction. Derived classes should call or replace this function. See *wxRadioBox::wxRadioBox* (page 260) for further details.

**wxRadioBox::Enable****void Enable**(Bool *enable*)

Enables or disables the entire radiobox.

**void Enable**(int *n*, Bool *enable*)

Enables or disables an individual button in the radiobox (does nothing in XView).

**wxRadioBox::FindString****int FindString**(char \**s*)

Finds a choice matching the given string, returning the position if found, or -1 if not found.

**wxRadioBox::GetSelection****int GetSelection**(void)

Gets the id (position) of the selected string.

**wxRadioBox::GetStringSelection****char \* GetStringSelection**(void)

Gets the selected string. This must be copied by the calling program if long term use is to be made of it.

**wxRadioBox::Number****int Number**(void)

Returns the number of choices in the radiobox.

**wxRadioBox::SetSelection****void SetSelection**(int *n*)

Sets the choice by passing the desired string position.

**wxRadioBox::SetStringSelection****void SetStringSelection**(char \* *s*)

Sets the choice by passing the desired string.

**wxRadioBox::Show****void Show**(int *item*, **Bool** *show*)

Shows or hides individual radio box controls.

**wxRadioBox::GetString****char \*** GetString(int *n*)

Returns a temporary pointer to the string at position *n*.

**9.93. wxRadioButton: wxItem**

A radio button item is a button which usually denotes one of several mutually exclusive options. It can be created as a standard button with a label, or as a bitmap button.

Please note that this is an experimental panel item, and is implemented for Windows and Motif only. Compilation of this functionality is controlled by the `USE_RADIOBUTTON` symbol.

**wxRadioButton::wxRadioButton****void wxRadioButton**(void)

Constructor, for use by derived classes.

**void wxRadioButton**(wxPanel \**parent*, wxFunction *func*, char \**label*, **Bool** *value*,  
int *x* = -1, int *y* = -1, int *width* = -1, int *height* = -1,  
long *style* = 0, char \**name* = "radioButton")**void wxRadioButton**(wxPanel \**parent*, wxFunction *func*, wxBitmap \**bitmap*, **Bool** *value*,  
int *x* = -1, int *y* = -1, int *width* = -1, int *height* = -1,  
long *style* = 0, char \**name* = "radioButton")

Constructor, creating and showing a radio button.

*func* may be NULL; otherwise it is used as the callback for the radio box. Note that the cast (wxFunction) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration.

If *label* is non-NULL, it will be used to label the radio button.

*bitmap* can be used to give the radio button a custom bitmap instead of a standard appearance and label.

*value* determines the initial value of the radio button.

The parameters *x* and *y* are used to specify an absolute position, or a position after the previous panel item if omitted or default.

If *width* or *height* are omitted (or are less than zero), an appropriate size will be used for the radio button.



*style* specifies a bitwise-or list of styles. The `wxB_GROUP` style can be used to start or end a group of buttons in Windows.

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual radio buttons.

### **wxRadioButton::~~wxRadioButton**

**void ~wxRadioButton(void)**

Destructor, destroying the radio button item.

### **wxRadioButton::Create**

**Bool Create(wxPanel \*parent, wxFunction func, char \*label, Bool value,  
int x = -1, int y = -1, int width = -1, int height = -1,  
long style = 0, char \*name = "radioButton")**

**Bool Create(wxPanel \*parent, wxFunction func, wxBitmap \*bitmap, Bool value,  
int x = -1, int y = -1, int width = -1, int height = -1,  
long style = 0, char \*name = "radioButton")**

Creates the choice for two-step construction. Derived classes should call or replace this function. See `wxRadioButton::wxRadioButton` (page 263) for further details.

### **wxRadioButton::GetValue**

**Bool GetValue(void)**

Returns TRUE if the radio button is depressed, FALSE otherwise.

### **wxRadioButton::SetValue**

**void SetValue(Bool value)**

Sets the radio button to selected or unselected status.

## **9.94. wxRecordSet: wxObject**

See also *Overview* (page 395)

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`.

### **wxRecordSet::wxRecordSet**

**void 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.

### **wxRecordSet::~~wxRecordSet**

**void ~wxRecordSet(void)**

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.

### **wxRecordSet::AddNew**

**void AddNew(void)**

Not implemented.

### **wxRecordSet::BeginQuery**

**Bool BeginQuery(int openType, char \*sql = NULL, int options = wxOPTION\_DEFAULT)**

Not implemented.

### **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 char \*col, void \*buf, long size)**

The same as above, but uses the column name as the identifier.

### **wxRecordSet::CanAppend**

**Bool CanAppend(void)**

Not implemented.

### **wxRecordSet::Cancel**

**void Cancel(void)**

Not implemented.

**wxRecordSet::CanRestart**

**Bool CanRestart(void)**

Not implemented.

**wxRecordSet::CanScroll**

**Bool CanScroll(void)**

Not implemented.

**wxRecordSet::CanTransact**

**Bool CanTransact(void)**

Not implemented.

**wxRecordSet::CanUpdate**

**Bool CanUpdate(void)**

Not implemented.

**wxRecordSet::ConstructDefaultSQL**

**Bool ConstructDefaultSQL(void)**

Not implemented.

**wxRecordSet::Delete**

**Bool Delete(void)**

Deletes the current record. Not implemented.

**wxRecordSet::Edit**

**void Edit(void)**

Not implemented.

**wxRecordSet::EndQuery****Bool EndQuery(void)**

Not implemented.

**wxRecordSet::ExecuteSQL****Bool ExecuteSQL(char \*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* (page 396).

**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.

**wxRecordSet::GetColName****char \* GetColName(int col)**

Returns the name of the column at position *col*. Returns NULL if *col* does not exist.

**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 char \* name)**

The same as above, but uses the column name as the identifier.

See *ODBC SQL data types* (page 395) for a list of possible data types.

**wxRecordSet::GetColumns****Bool GetColumns(char \*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

### **wxRecordSet::GetCurrentRecord**

**long GetCurrentRecord(void)**

Not implemented.

### **wxRecordSet::GetDatabase**

**wxDatabase \* GetDatabase(void)**

Returns the wxDatabase object bound to a wxRecordSet.

### **wxRecordSet::GetDataSources**

**Bool GetDataSources(void)**

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());
}
```

```
}
```

**wxRecordSet::GetDefaultConnect****char \* GetDefaultConnect(void)**

Not implemented.

**wxRecordSet::GetDefaultSQL****char \* GetDefaultSQL(void)**

Not implemented.

**wxRecordSet::GetErrorCode****wxRETCODE GetErrorCode(void)**

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.

**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 char \*name, int dataType, void \*dataPtr)**

The same as above, but uses the column name as the identifier.

See *ODBC SQL data types* (page 395) for a list of possible data types.

**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 char \*name, int dataType)**

The same as above, but uses the column name as the identifier.

See *ODBC SQL data types* (page 395) for a list of possible data types.

### **wxRecordSet::GetFilter**

**char \* GetFilter(void)**

Returns the current filter.

### **wxRecordSet::GetForeignKeys**

**Bool GetPrimaryKeys(char \*ptable = NULL, char \*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

**wxRecordSet::GetNumberCols****long GetNumberCols(void)**

Returns the number of columns in the result set.

**wxRecordSet::GetNumberFields****int GetNumberFields(void)**

Not implemented.

**wxRecordSet::GetNumberParams****int GetNumberParams(void)**

Not implemented.

**wxRecordSet::GetNumberRecords****long GetNumberRecords(void)**

Returns the number of records in the result set.

**wxRecordSet::GetPrimaryKeys****Bool GetPrimaryKeys(char \*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

**wxRecordSet::GetOptions****int GetOptions(void)**

Returns the options of the wxRecordSet. Options are not supported yet.

**wxRecordSet::GetResultSet****Bool GetResultSet(void)**

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.

**wxRecordSet::GetSortString**

**char \* GetSortString(void)**

Not implemented.

**wxRecordSet::GetSQL**

**char \* GetSQL(void)**

Not implemented.

**wxRecordSet::GetTableName**

**char \* GetTableName(void)**

Returns the name of the current table.

**wxRecordSet::GetTables**

**Bool GetTables(void)**

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

**wxRecordSet::GetType**

**int GetType(void)**

Returns the type of the wxRecordSet: wxOPEN\_TYPE\_DYNASET or wxOPEN\_TYPE\_SNAPSHOT. See the wxRecordSet description for details.

**wxRecordSet::GoTo**

**Bool GoTo(long n)**

Moves the cursor to the record with the number n, where the first record has the number 0.

**wxRecordSet::IsBOF**

**Bool IsBOF(void)**

Returns TRUE if the user tried to move the cursor before the first record in the set.

**wxRecordSet::IsFieldDirty****Bool IsFieldDirty(int *field*)**

Returns TRUE if the given field has been changed but not saved yet.

**Bool IsFieldDirty(const char \**name*)**

Same as above, but uses the column name as the identifier.

**wxRecordSet::IsFieldNull****Bool IsFieldNull(int *field*)**

Returns TRUE if the given field has no data.

**Bool IsFieldNull(const char \* *name*)**

Same as above, but uses the column name as the identifier.

**wxRecordSet::IsColNullable****Bool IsColNullable(int *col*)**

Returns TRUE if the given column may contain no data.

**Bool IsColNullable(const char \**name*)**

Same as above, but uses the column name as the identifier.

**wxRecordSet::IsEOF****Bool IsEOF(void)**

Returns TRUE if the user tried to move the cursor behind the last record in the set.

**wxRecordSet::IsDeleted****Bool IsDeleted(void)**

Not implemented.

**wxRecordSet::IsOpen****Bool IsOpen(void)**

Returns TRUE if the parent database is open.

**wxRecordSet::Move**

**Bool Move(long rows)**

Moves the cursor a given number of rows. Negative values are allowed.

**wxRecordSet::MoveFirst**

**Bool MoveFirst(void)**

Moves the cursor to the first record.

**wxRecordSet::MoveLast**

**Bool MoveLast(void)**

Moves the cursor to the last record.

**wxRecordSet::MoveNext**

**Bool MoveNext(void)**

Moves the cursor to the next record.

**wxRecordSet::MovePrev**

**Bool MovePrev(void)**

Moves the cursor to the previous record.

**wxRecordSet::Query**

**Bool Query(char \*columns, char \*table, char \*filter = NULL)**

Start a query. An SQL string of the following type will automatically be generated and executed: "SELECT columns FROM table WHERE filter".

**wxRecordSet::RecordCountFinal**

**Bool RecordCountFinal(void)**

Not implemented.

**wxRecordSet::Requery**

**Bool Requery(void)**

Re-executes the last query. Not implemented.

**wxRecordSet::SetFieldDirty**

**void SetFieldDirty(int field, Bool dirty = TRUE)**

Sets the dirty tag of the field field. Not implemented.

**void SetFieldDirty(const char \*name, Bool dirty = TRUE)**

Same as above, but uses the column name as the identifier.

**wxRecordSet::SetDefaultSQL****void SetDefaultSQL(char \*s)**

Not implemented.

**wxRecordSet::SetFieldNull****void SetFieldNull(void \*p, Bool isNull = TRUE)**

Not implemented.

**wxRecordSet::SetOptions****void SetOptions(int opt)**

Sets the options of the wxRecordSet. Not implemented.

**wxRecordSet::SetTableName****void SetTableName(char \*tablename)**

Specify the name of the table you want to use.

**wxRecordSet::SetType****void SetType(int type)**

Sets the type of the wxRecordSet. See the wxRecordSet class description for details.

**wxRecordSet::Update****Bool Update(void)**

Writes back the current record. Not implemented.

**9.95. wxScreenDC: wxCanvasDC**

An instance of this class may be created to access the whole screen. Free the instance as soon as it has been used, since there are a limited number of device contexts in some environments.

Note that this hasn't been tested yet.

See *wxDC* (page 108) for further information on device contexts.

**wxScreenDC::wxScreenDC****void wxScreenDC(void)**

Constructor.

### 9.96. wxScrollBar: wxItem

A `wxScrollBar` is a *panel item* that represents a horizontal or vertical scroll control. It may be used on panels to give similar functionality to a scrollable `wxCanvas`, or it may be used as a kind of slider.

*Note* that the constructor arguments have changed in version 1.65.

*Note also* that from 1.66, `SetObjectLength` (page 277) is now consistent under Motif and Windows, so your code under Windows may need to change. You must call `SetViewLength` before calling `SetObjectLength`. See `wxGenericGrid` for an example of usage.

#### `wxScrollBar::wxScrollBar`

```
void wxScrollBar(wxPanel *parent, wxFunction func,  
    int x = -1, int y = -1, int width = -1, int height = -1,  
    long style = wxHORIZONTAL, char *name = "scrollBar")
```

Constructor, creating and showing a scrollbar. The parent must be a valid panel or dialog box pointer.

*Note* that the constructor arguments have changed in version 1.65: the old *direction* parameter is now passed in the window style.

*func* may be NULL; otherwise it is used as the callback for the scrollbar.

The parameters *x* and *y* are used to specify an absolute position, or a position after the previous panel item if omitted or default.

If *width* or *height* are omitted (or are less than zero), an appropriate size will be used for the scrollbar.

*style* may be either `wxHORIZONTAL` or `wxVERTICAL`.

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual scrollbars.

#### `wxScrollBar::~~wxScrollBar`

```
void ~wxScrollBar(void)
```

Destructor, destroying the scrollbar.

#### `wxScrollBar::Create`

```
void Create(wxPanel *parent, wxFunction func,  
    int x = -1, int y = -1, int width = -1, int height = -1,  
    long style = wxHORIZONTAL, char *name = "scrollBar")
```

Scrollbar creation function called by the scrollbar constructor. Call it when a derived scrollbar class uses the zero-argument `wxScrollBar` constructor, but can reuse the existing scrollbar creation code. See `wxScrollBar::wxScrollBar` (page 276) for details.

**wxScrollBar::GetValue****int GetValue(void)**

Returns the current position of the scrollbar.

**wxScrollBar::GetValues**

**void GetValues(int \*viewStart, int \*viewLength, int \*objectLength, int \*pageLength)** Returns scrollbar settings information.

**wxScrollBar::SetObjectLength****void SetObjectLength(int objectLength)**

Sets the object length for the scrollbar. This is the total object size (virtual size). You must call *SetViewLength* (page 277) before calling *SetObjectLength*.

Example: you are implementing scrollbars on a text window, where text lines have a maximum width of 100 characters. Your text window has a current width of 60 characters. So the view length is 60, and the object length is 100. The scrollbar will then enable you to scroll to see the other 40 characters.

You will need to call *SetViewLength* and *SetObjectLength* whenever there is a change in the size of the window (the view size) or the size of the contents (the object length).

**wxScrollBar::SetPageLength****void SetPageLength(int pageLength)**

Sets the page length for the scrollbar. This is the number of scroll units which are scrolled when the user pages down (clicks on the scrollbar outside the thumbtrack area).

**wxScrollBar::SetViewLength****void SetViewLength(int viewLength)**

Sets the view length for the scrollbar.

**wxScrollBar::SetValue****void SetValue(int viewStart)**

Sets the position of the scrollbar.

**9.97. wxServer: wxIPCObject**

See also *IPC overview* (page 378)

A `wxServer` object represents the server part of a client-server DDE (Dynamic Data Exchange) conversation (available under both Windows and UNIX).

### **wxServer::wxServer**

**void wxServer(void)**

Constructs a server object.

### **wxServer::Create**

**Bool Create(char \*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).

### **wxServer::OnAcceptConnection**

**wxConnection \* OnAcceptConnection(char \*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 `wxConnection` 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.

## **9.98. wxSlider: wxItem**

A slider is, as its name suggests, an item with a handle which can be pulled back and forth to change a value. It is currently horizontal only. In MS Windows, a scrollbar is used to simulate the slider.

### **wxSlider::wxSlider**

**void wxSlider(wxPanel \*parent, wxFunction func, char \*label,  
int value, int min\_value, int max\_value, int width,  
int x = -1, int y = -1, long style = wxHORIZONTAL, char \*name = "slider")**

Constructor, creating and showing a horizontal slider.

*func* may be NULL; otherwise it is used as the callback for the slider. Note that the cast (`wxFunction`) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration.

If *label* is non-NULL, it will be used to label the slider.

The parameters *x* and *y* are used to specify an absolute position, or a position after the previous

panel item if omitted or default.

The *width* is in pixels, and the scroll increment will be adjusted to a suitable value given the minimum and maximum integer values.

The *style* parameter may be `wxHORIZONTAL` to denote a horizontal slider, or `wxVERTICAL` for a vertical slider.

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual sliders.

### **wxSlider::~~wxSlider**

**void ~wxSlider(void)**

Destructor, destroying the slider.

### **wxSlider::Create**

**void Create(wxPanel \*parent, wxFunction func, char \*label,  
int value, int min\_value, int max\_value, int width,  
int x = -1, int y = -1, long style = 0, char \*name = "slider")**

Used for two-step slider construction. See `wxSlider::wxSlider` (page 278) for further details.

### **wxSlider::GetMax**

**int GetMax(void)**

Gets the maximum slider value.

### **wxSlider::GetMin**

**int GetMin(void)**

Gets the minimum slider value.

### **wxSlider::GetValue**

**int GetValue(void)**

Gets the current slider value.

### **wxSlider::SetRange**

**void SetRange(int min\_value, int max\_value)**

Sets the minimum and maximum slider values.



**wxSlider::SetValue****void SetValue(int value)**

Sets the value (and displayed position) of the slider).

**9.99. wxSplitterWindow: wxCanvas**

See also *wxSplitterWindow overview* (page 420)

This class manages either one or 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.

**wxSplitterWindow::wxSplitterWindow****wxSplitterWindow(void)**

Default constructor.

**wxSplitterWindow(wxWindow \*parent, int x, int y, int width, int height, long style=0, char \*name)**

Constructor for creating the window.

**Parameters***parent*

The parent of the splitter window.

*width*

The window width.

*height*

The window height.

*style*

The window style. May be a bit list of:

**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.

*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 *Initialize* (page 282), *SplitVertically* (page 285) and *SplitHorizontally* (page 284) 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**

*Initialize* (page 282), *SplitVertically* (page 285), *SplitHorizontally* (page 284)

## **wxSplitterWindow::~~wxSplitterWindow**

**~wxSplitterWindow(void)**

Destroys the wxSplitterWindow and its children.

## **wxSplitterWindow::GetMinimumPaneSize**

**int GetMinimumPaneSize(void)**

Returns the current minimum pane size (defaults to zero).

**See also**

*SetMinimumPaneSize* (page 283)

## **wxSplitterWindow::GetSashPosition**

**int GetSashPosition(void)**

Returns the current sash position.

**See also**

*SetSashPosition* (page 283)

## **wxSplitterWindow::GetSplitMode**

**int GetSplitMode(void)**

Gets the split mode.

**See also**

*SetSplitMode* (page 284), *SplitVertically* (page 285), *SplitHorizontally* (page 284).

## **wxSplitterWindow::GetWindow1**

**wxWindow\* GetWindow1(void)**

Returns the left/top or only pane.

**wxSplitterWindow::GetWindow2****wxWindow\* GetWindow2(void)**

Returns the right/bottom pane.

**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**

*SplitVertically* (page 285), *SplitHorizontally* (page 284).

**wxSplitterWindow::IsSplit****Bool IsSplit(void)**

Returns TRUE if the window is split, FALSE otherwise.

**wxSplitterWindow::OnDoubleClickSash****virtual void OnDoubleClickSash(int x, int y)**

Application-overridable 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* (page 285) if the minimum pane size is zero.

**See also**

*Unsplit* (page 285)

## **wxSplitterWindow::OnUnsplit**

**virtual void OnUnsplit(wxWindow\* *removed*)**

Application-overridable 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.

### **See also**

*Unsplit* (page 285)

## **wxSplitterWindow::SetSashPosition**

**void SetSashPosition(int *position*, 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**

*GetSashPosition* (page 281)

## **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**

*GetMinimumPaneSize* (page 281)

**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**

*GetSplitMode* (page 281), *SplitVertically* (page 285), *SplitHorizontally* (page 284).

**wxSplitterWindow::SplitHorizontally**

**Bool SplitHorizontally(wxWindow\* window1, wxWindow\* window2, int sashPosition = -1)**

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 the value is -1, a default position is chosen.

**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* (page 282).

#### See also

*SplitVertically* (page 285), *IsSplit* (page 282), *Unsplit* (page 285).

### **wxSplitterWindow::SplitVertically**

**Bool SplitVertically(wxWindow\* window1, wxWindow\* window2, int sashPosition = -1)**

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 the value is -1, a default position is chosen.

#### 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* (page 282).

#### See also

*SplitHorizontally* (page 284), *IsSplit* (page 282), *Unsplit* (page 285).

### **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* (page 283) which can be overridden for the desired behaviour. By default, the pane being removed is hidden.

**See also**

*SplitHorizontally* (page 284), *SplitVertically* (page 285), *IsSplit* (page 282), *OnUnsplit* (page 283).

**9.100. wxString: wxObject**

See also *Overview* (page 399)

*Member functions by category* (page 363)

**CAVE:** The description of the memberfunctions is very sparse in the moment. It will be extended in the next version of the help file. The list of memberfunctions is complete.

**wxString::wxString**

```
void wxString(void)
void wxString(const wxString& x)
void wxString(const wxSubString& x)
void wxString(const char* t)
void wxString(const char* t, int len)
void wxString(char c)
```

Constructors.

**wxString::~~wxString**

```
void ~wxString(void)
```

String destructor.

**wxString::Alloc**

```
void Alloc(int newsize)
```

Preallocate some space for wxString.

**wxString::Allocation**

```
int Allocation(void) const
```

Report current allocation (not length!).

## **wxString::Append**

```
wxString& Append(const char* cs)
wxString& Append(const wxString& s)
```

Concatenation.

```
wxString& Append(char c, int rep = 1)
```

Append *c*, *rep* times

## **wxString::After**

```
wxSubString After(int pos)
wxSubString After(const wxString& x, int startpos = 0)
wxSubString After(const wxSubString& x, int startpos = 0)
wxSubString After(const char* t, int startpos = 0)
wxSubString After(char c, int startpos = 0)
wxSubString After(const wxRegex& r, int startpos = 0)
```

## **wxString::At**

```
wxSubString At(int pos, int len)
wxSubString operator()(int pos, int len)
wxSubString At(const wxString& x, int startpos = 0)
wxSubString At(const wxSubString& x, int startpos = 0)
wxSubString At(const char* t, int startpos = 0)
wxSubString At(char c, int startpos = 0)
wxSubString At(const wxRegex& r, int startpos = 0)
```

wxSubString extraction.

Note that you can't take a substring of a const wxString, since this leaves open the possibility of indirectly modifying the wxString through the wxSubString.

## **wxString::Before**

```
wxSubString Before(int pos)
wxSubString Before(const wxString& x, int startpos = 0)
wxSubString Before(const wxSubString& x, int startpos = 0)
wxSubString Before(const char* t, int startpos = 0)
wxSubString Before(char c, int startpos = 0)
wxSubString Before(const wxRegex& r, int startpos = 0)
```

## **wxString::Capitalize**

```
void Capitalize(void)
friend wxString Capitalize(wxString& x)
```



**wxString::Cat**

```

friend void Cat(const wxString& a, const wxString& b, wxString& c)
friend void Cat(const wxString& a, const wxSubString& b, wxString& c)
friend void Cat(const wxString& a, const char* b, wxString& c)
friend void Cat(const wxString& a, char b, wxString& c)
friend void Cat(const wxSubString& a, const wxString& b, wxString& c)
friend void Cat(const wxSubString& a, const wxSubString& b, wxString& c)
friend void Cat(const wxSubString& a, const char* b, wxString& c)
friend void Cat(const wxSubString& a, char b, wxString& c)
friend void Cat(const char* a, const wxString& b, wxString& c)
friend void Cat(const char* a, const wxSubString& b, wxString& c)
friend void Cat(const char* a, const char* b, wxString& c)
friend void Cat(const char* a, char b, wxString& c)

```

Concatenate first two arguments, store the result in the last argument.

```

friend void Cat(const wxString& a, const wxString& b, const wxString& c, wxString& d)
friend void Cat(const wxString& a, const wxString& b, const wxSubString& c, wxString& d)
friend void Cat(const wxString& a, const wxString& b, const char* c, wxString& d)
friend void Cat(const wxString& a, const wxSubString& b, char c, wxString& d)
friend void Cat(const wxString& a, const wxSubString& b, const wxString& c, wxString& d)
friend void Cat(const wxString& a, const wxSubString& b, const wxSubString& c,
wxString& d)
friend void Cat(const wxString& a, const wxSubString& b, const char* c, wxString& d)
friend void Cat(const wxString& a, const wxSubString& b, char c, wxString& d)
friend void Cat(const wxString& a, const char* b, const wxString& c, wxString& d)
friend void Cat(const wxString& a, const char* b, const wxSubString& c, wxString& d)
friend void Cat(const wxString& a, const char* b, const char* c, wxString& d)
friend void Cat(const wxString& a, const char* b, char c, wxString& d)

friend void Cat(const char* a, const wxString& b, const wxString& c, wxString& d)
friend void Cat(const char* a, const wxString& b, const wxSubString& c, wxString& d)
friend void Cat(const char* a, const wxString& b, const char* c, wxString& d)
friend void Cat(const char* a, const wxString& b, char c, wxString& d)
friend void Cat(const char* a, const wxSubString& b, const wxString& c, wxString& d)
friend void Cat(const char* a, const wxSubString& b, const wxSubString& c, wxString& d)
friend void Cat(const char* a, const wxSubString& b, const char* c, wxString& d)
friend void Cat(const char* a, const wxSubString& b, char c, wxString& d)
friend void Cat(const char* a, const char* b, const wxString& c, wxString& d)
friend void Cat(const char* a, const char* b, const wxSubString& c, wxString& d)
friend void Cat(const char* a, const char* b, const char* c, wxString& d)
friend void Cat(const char* a, const char* b, char c, wxString& d)

```

Double concatenation, by request. (Yes, there are too many versions, but if one is supported, then the others should be too). Concatenate the first 3 args, store the result in the last argument.

**wxString::Chars**

```
const char* Chars(void) const
```

Conversion.

## **wxString::CompareTo**

```
#define NO_POS ((int)(-1)) // undefined position
enum CaseCompare {exact, ignoreCase};
```

```
    int CompareTo(const char* cs, CaseCompare cmp = exact) const
int CompareTo(const wxString& cs, CaseCompare cmp = exact) const
```

## **wxString::Contains**

```
Bool Contains(char c) const
Bool Contains(const wxString& y) const
Bool Contains(const wxSubString& y) const
Bool Contains(const char* t) const
Bool Contains(const wxRegex& r) const
```

Return 1 if target appears anywhere in wxString; else 0.

```
Bool Contains(const char* pat, CaseCompare cmp) const
Bool Contains(const wxString& pat, CaseCompare cmp) const
```

Case dependent/independent variation .

```
Bool Contains(char c, int pos) const
Bool Contains(const wxString& y, int pos) const
Bool Contains(const wxSubString& y, int pos) const
Bool Contains(const char* t, int pos) const
Bool Contains(const wxRegex& r, int pos) const
```

Return 1 if the target appears anywhere after position *pos* (or before, if *pos* is negative) in wxString; else 0.

## **wxString::Copy**

```
wxString Copy(void) const
```

Duplication.

## **wxString::Del**

```
wxString& Del(int pos, int len)
```

Delete *len* characters starting at *pos*.

```
wxString& Del(const wxString& y, int startpos = 0)
wxString& Del(const wxSubString& y, int startpos = 0)
wxString& Del(const char* t, int startpos = 0)
wxString& Del(char c, int startpos = 0)
wxString& Del(const wxRegex& r, int startpos = 0)
```

Delete the first occurrence of target after *startpos*.

**wxString::DownCase**

```
void Downcase(void)
friend wxString Downcase(wxString& x)
```

**wxString::Elem**

```
char Elem(int i) const
```

Element extraction.

**wxString::Empty**

```
int Empty(void) const
```

**wxString::Error**

```
void Error(const char* msg) const
```

**wxString::First**

```
int First(char c) const
int First(const char* cs) const
int First(const wxString& cs) const
```

Return first or last occurrence of item.

**wxString::Firstchar**

```
char Firstchar(void) const
```

Element extraction.

**wxString::Freq**

```
int Freq(char c) const
int Freq(const wxString& y) const
int Freq(const wxSubString& y) const
int Freq(const char* t) const
```

Return number of occurrences of target in wxString.

**wxString::From**

```
wxSubString From(int pos)
wxSubString From(const wxString& x, int startpos = 0)
wxSubString From(const wxSubString& x, int startpos = 0)
```

```
wxSubString From(const char* t, int startpos = 0)  
wxSubString From(char c, int startpos = 0)  
wxSubString From(const wxRegex& r, int startpos = 0)
```

### **wxString::GetData**

```
char* GetData(void)
```

wxWindows compatibility conversion.

### **wxString::GSub**

```
int GSub(const wxString& pat, const wxString& repl)  
int GSub(const wxSubString& pat, const wxString& repl)  
int GSub(const char* pat, const wxString& repl)  
int GSub(const char* pat, const char* repl)  
int GSub(const wxRegex& pat, const wxString& repl)
```

Global substitution: substitute all occurrences of *pat* with *repl*, returning the number of matches.

### **wxString::Index**

```
int Index(char c, int startpos = 0) const  
int Index(const wxString& y, int startpos = 0) const  
int Index(const wxString& y, int startpos, CaseCompare cmp) const  
int Index(const wxSubString& y, int startpos = 0) const  
int Index(const char* t, int startpos = 0) const  
int Index(const char* t, int startpos, CaseCompare cmp) const  
int Index(const wxRegex& r, int startpos = 0) const
```

Return the position of target in string, or -1 for failure.

### **wxString::Insert**

```
wxString& Insert(int pos, const char* s)  
wxString& Insert(int pos, const wxString& s)
```

Insertion.

### **wxString::IsAscii**

```
int IsAscii(void) const
```

Classification (should be capital, because of ctype.h macros).

### **wxString::IsDefined**

```
int IsDefined(void) const
```

Classification (should be capital, because of ctype.h macros).

**wxString::IsNull****int IsNull(void) const**

Classification (should be capital, because of ctype.h macros).

**wxString::IsNumber****int IsNumber(void) const**

Classification (should be capital, because of ctype.h macros).

**wxString::IsWord****int IsWord(void) const**

Classification (should be capital, because of ctype.h macros).

**wxString::Last****int Last(char c) const**  
**int Last(const char\* cs) const**  
**int Last(const wxString& cs) const**

First or last occurrence of item.

**wxString::Lastchar****char Lastchar(void) const**

Element extraction.

**wxString::Length****unsigned int Length(void) const****wxString::LowerCase****void LowerCase(void)****wxString::Matches****Bool Matches(char c, int pos = 0) const**  
**Bool Matches(const wxString& y, int pos = 0) const**  
**Bool Matches(const wxSubString& y, int pos = 0) const**  
**Bool Matches(const char\* t, int pos = 0) const**

**Bool Matches**(const wxRegex& *r*, int *pos* = 0) const

Return 1 if target appears at position *pos* in wxString; else 0.

## **wxString::OK**

int OK(void) const

## **wxString::Prepend**

wxString& Prepend(const wxString& *y*)  
wxString& Prepend(const wxSubString& *y*)  
wxString& Prepend(const char\* *t*)  
wxString& Prepend(char *c*)

Prepend.

wxString& Prepend(char *c*, int *rep*=1)

Prepend *c*, *rep* times.

## **wxString::Readline**

friend int Readline(istream& *s*, wxString& *x*, char *terminator* = '\n', int *discard\_terminator* = 1)  
friend int Readline(FILE \* *f*, wxString& *x*, char *terminator* = '\n', int *discard\_terminator* = 1)

## **wxString::Remove**

wxString& RemoveLast(void)  
wxString& Remove(int *pos*)  
wxString& Remove(int *pos*, int *len*)

Remove *pos* to end of string.

## **wxString::Replace**

wxString& Replace(int *pos*, int *n*, const char\* *s*)  
wxString& Replace(int *pos*, int *n*, const wxString& *s*)

## **wxString::Replicate**

friend wxString Replicate(char *c*, int *n*)  
friend wxString Replicate(const wxString& *y*, int *n*)

Replication.

## **wxString::Reverse**

void Reverse(void)

**friend wxString Reverse(wxString& x)**

### **wxString::sprintf**

**void sprintf(const char \* *fmt*)**

Formatted assignment. We do not use the 'sprintf' constructor anymore, because with that constructor, every initialisation with a string would go through sprintf and this is not desirable, because sprintf interprets some characters. With the above function we can write:

```
wxString msg; msg.sprintf("Processing item %d\n", count);
```

### **wxString::Strip**

```
enumStripType {leading = 0x1, trailing = 0x2, both = 0x3};
```

**wxSubString Strip(StripType *s* = *trailing*, char *c* = ' ')**

Strip characters *s* at the front and/or end. StripType is defined for bitwise ORing.

### **wxString::SubString**

**wxString SubString(int *from*, int *to*)**

Edward Zimmermann's additions.

### **wxString::Through**

```
wxSubString Through(int pos)  
wxSubString Through(const wxString& x, int startpos = 0)  
wxSubString Through(const wxSubString& x, int startpos = 0)  
wxSubString Through(const char* t, int startpos = 0)  
wxSubString Through(char c, int startpos = 0)  
wxSubString Through(const wxRegex& r, int startpos = 0)
```

### **wxString::Uppcase**

```
void Uppcase(void)  
friend wxString Uppcase(wxString& x)
```

### **wxString::UpperCase**

```
void UpperCase(void)
```

### **wxString::operator =**

```
wxString& operator =(const wxString& y)
wxString& operator =(const char* y)
wxString& operator =(char c)
wxString& operator =(const wxSubString& y)
```

Assignment.

**wxString::operator +=**

```
wxString& operator +=(const wxString& y)
wxString& operator +=(const wxSubString& y)
wxString& operator +=(const char* t)
wxString& operator +=(char c)
```

Concatenation.

**wxString::operator []**

```
char& operator [] (int i)
```

Element extraction.

**wxString::operator ()**

```
char& operator () (int i)
```

**wxString::operator <<**

```
friend ostream& operator <<(ostream& s, const wxString& x)
friend ostream& operator <<(ostream& s, const wxSubString& x)
```

**wxString::operator >>**

```
friend istream& operator >>(istream& s, wxString& x)
```

**wxString::operator const char \***

```
operator const char*(void) const
```

Conversion.

**wxCHARARG**

```
#define wxCHARARG(s) ((char *) (s).Chars())
```

Here is a very, very, very ugly macro, but it makes things more transparent in cases, where a library function requires a (char \*) argument. This is especially the case in wxWindows, where all char-arguments are (char \*) and not (const char \*). This macro should only be used in such cases and NOT to modify the internal data. The conventional way would be 'function((char



`*)string.Chars())'. With the wxCHARARG macro, this can be achieved by 'function(wxCHARARG(string))'. This makes it clearer that the usage should be confined to arguments.`

### **CommonPrefix**

```
friend wxString CommonPrefix(const wxString& x, const wxString& y,  
    int startpos = 0)
```

### **CommonSuffix**

```
friend wxString CommonSuffix(const wxString& x, const wxString& y,  
    int startpos = -1)
```

### **Compare**

```
int Compare(const wxString& x, const wxString& y)  
int Compare(const wxString& x, const wxSubString& y)  
int Compare(const wxString& x, const char* y)  
int Compare(const wxSubString& x, const wxString& y)  
int Compare(const wxSubString& x, const wxSubString& y)  
int Compare(const wxSubString& x, const char* y)
```

Case dependent comparison. Returns 0 if the match succeeded.

### **FCompare**

```
int FCompare(const wxString& x, const wxString& y)
```

Case independent comparison. Returns 0 if the match succeeded.

### **Comparison operators**

```
int operator ==(const wxString& x, const wxString& y)  
int operator !=(const wxString& x, const wxString& y)  
int operator >(const wxString& x, const wxString& y)  
int operator >=(const wxString& x, const wxString& y)  
int operator <(const wxString& x, const wxString& y)  
int operator <=(const wxString& x, const wxString& y)  
int operator ==(const wxString& x, const wxSubString& y)  
int operator !=(const wxString& x, const wxSubString& y)  
int operator >(const wxString& x, const wxSubString& y)  
int operator >=(const wxString& x, const wxSubString& y)  
int operator <(const wxString& x, const wxSubString& y)  
int operator <=(const wxString& x, const wxSubString& y)  
int operator ==(const wxString& x, const char* t)  
int operator !=(const wxString& x, const char* t)  
int operator >(const wxString& x, const char* t)  
int operator >=(const wxString& x, const char* t)
```

```
int operator <(const wxString& x, const char* t)
int operator <=(const wxString& x, const char* t)
int operator ==(const wxSubString& x, const wxString& y)
int operator !=(const wxSubString& x, const wxString& y)
int operator >(const wxSubString& x, const wxString& y)
int operator >=(const wxSubString& x, const wxString& y)
int operator <(const wxSubString& x, const wxString& y)
int operator <=(const wxSubString& x, const wxString& y)
int operator ==(const wxSubString& x, const wxSubString& y)
int operator !=(const wxSubString& x, const wxSubString& y)
int operator >(const wxSubString& x, const wxSubString& y)
int operator >=(const wxSubString& x, const wxSubString& y)
int operator <(const wxSubString& x, const wxSubString& y)
int operator <=(const wxSubString& x, const wxSubString& y)
int operator ==(const wxSubString& x, const char* t)
int operator !=(const wxSubString& x, const char* t)
int operator >(const wxSubString& x, const char* t)
int operator >=(const wxSubString& x, const char* t)
int operator <(const wxSubString& x, const char* t)
int operator <=(const wxSubString& x, const char* t)
```

### **operator +**

```
wxString operator +(const wxString& x, const wxString& y)
wxString operator +(const wxString& x, const wxSubString& y)
wxString operator +(const wxString& x, const char* y)
wxString operator +(const wxString& x, char y)
wxString operator +(const wxSubString& x, const wxString& y)
wxString operator +(const wxSubString& x, const wxSubString& y)
wxString operator +(const wxSubString& x, const char* y)
wxString operator +(const wxSubString& x, char y)
wxString operator +(const char* x, const wxString& y)
wxString operator +(const char* x, const wxSubString& y)
```

### **Join**

```
friend wxString Join(wxString src[], int n, const wxString& sep)
```

### **Split**

```
friend int Split(const wxString& x, wxString res[], int maxn,
const wxString& sep)
friend int Split(const wxString& x, wxString res[], int maxn,
const wxRegex& sep)
```

Split string into array res at separators; return number of elements

## **9.101. wxStringList: wxList**

A string list is a list which is assumed to contain strings, with a specific member functions. Memory is allocated when strings are added to the list, and deallocated by the destructor or by

the **Delete** member.

### **wxStringList::wxStringList**

**void wxStringList(void)**

Constructor.

**void wxStringList(char \*first, ...)**

Constructor, taking NULL-terminated string argument list. wxStringList allocates memory for the strings.

### **wxStringList::~~wxStringList**

**void ~wxStringList(void)**

Deletes string list, deallocating strings.

### **wxStringList::Add**

**wxNode \* Add(char \*s)**

Adds string to list, allocating memory.

### **wxStringList::Delete**

**void Delete(char \*s)**

Searches for string and deletes from list, deallocating memory.

### **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.

### **wxStringList::Member**

**Bool Member(char \*s)**

Returns TRUE if **s** is a member of the list (tested using **strcmp**).

### **wxStringList::Sort**

**void Sort(void)**

Sorts the strings in ascending alphabetical order. Note that all nodes (but not strings) get deallocated and new ones allocated.

### 9.102. **wxText: wxItem**

A text item is an area of editable text, with an optional label displayed in front of it.

The callback function specified for the text item will be called for the following events:

- `wxEVENT_TYPE_TEXT_COMMAND` (text has changed)
- `wxEVENT_TYPE_TEXT_ENTER_COMMAND` (enter has been pressed: if the `wxPROCESS_ENTER` style is used)
- under Windows, `wxEVENT_TYPE_SET_FOCUS`, `wxEVENT_TYPE_KILL_FOCUS` when the focus changes.

#### **wxText::wxText**

##### **void wxText(void)**

Constructor, for deriving classes.

```
void wxText(wxPanel *parent, wxFunction func, char *label,  
char *value = "", int x = -1, int y = -1, int width = -1, int height = -1,  
long style = 0, char *name = "text")
```

Constructor, creating and showing a text item with the given string value.

*func* may be `NULL`; otherwise it is used as the callback for the list box. Note that the cast (`wxFunction`) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration.

If *label* is non-`NULL`, it will be used to label the text item.

The parameters *x* and *y* are used to specify an absolute position, or a position after the previous panel item if omitted or default.

If *width* or *height* are omitted (or are less than zero), an appropriate size will be used for the item.

The *style* parameter can be a bit list of the following:

<code>wxTE_PROCESS_ENTER</code>	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.
<code>wxTE_PASSWORD</code>	The text will be echoed as asterisks.
<code>wxTE_READONLY</code>	The text will not be user-editable.
<code>wxFIXED_LENGTH</code>	Allows the values of a column of items to be left-aligned. Create an item with this style, and pad out your labels with spaces to the same length. The item labels will initially be created with a string of identical characters, positioning all the values at the same x-position. Then the real label is restored.

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual text items.

**wxText::~~wxText****void ~wxText(void)**

Destructor, destroying the text item.

**wxText::Copy****void Copy(void)**

Copies the selected text to the clipboard under Motif and MS Windows.

**wxText::Create****Bool Create(wxPanel \*parent, wxFunction func, char \*label,  
char \*value = "", int x = -1, int y = -1, int width = -1, int height = -1,  
long style = 0, char \*name = "text")**

Creates the text item for two-step construction. Derived classes should call or replace this function. See *wxText::wxText* (page 299) for further details.

**wxText::Cut****void Cut(void)**

Copies the selected text to the clipboard and removes the selection. Windows and Motif only.

**wxText::GetInsertionPoint****long GetInsertionPoint(void)**

Returns the insertion point. Windows and Motif only.

**wxText::GetLastPosition****long GetLastPosition(void)**

Returns the last position in the text item. Windows and Motif only.

**wxText::GetValue****char \* GetValue(void)**

Gets a pointer to the current value. Copy this for long-term use.

**wxText::Paste**

**void Paste(void)**

Pastes text from the clipboard to the text item. Windows and Motif only.

**wxText::Remove****void Remove(long from, long to)**

Removes the text between the two positions. Windows and Motif only.

**wxText::Replace****void Replace(long from, long to, char \*value)**

Replaces the text between two positions with the given text. Windows and Motif only.

**wxText::SetEditable****void SetEditable(Bool editable)**

Makes the text item editable (TRUE) or read-only (FALSE).

**wxText::SetInsertionPoint****void SetInsertionPoint(long pos)**

Sets the insertion point. Windows only.

**wxText::SetInsertionPointEnd****void SetInsertionPointEnd(void)**

Sets the insertion point at the end of the text item. Windows and Motif only.

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

Selects the text between the two positions. Windows and Motif only.

**wxText::SetValue****void SetValue(char \* value)**

Sets the text. *value* must be deallocated by the calling program.

### 9.103. wxTextWindow: wxWindow

A text window is a subwindow of a frame (or a panel, on some platforms), offering some basic ability to display scrolling text. Editing is possible under all platforms, but if editing is required under Windows, the **wxNATIVE\_IMPL** style should be included. This is because the default implementation is a read-only text window that can display more than the 64K or so of text allowed using the standard edit control.

Some manipulation functions take integer positions, starting from zero.

Many of these functions are currently platform-specific, and have yet to be fully implemented or tested.

For compilers other than Borland C++, this class also derives from `streambuf`. You can then use instances of `wxTextWindow` for the usual ostream operations, for example:

```
wxTextWindow *textwin = new wxTextWindow(...);

ostream stream(textwin);
stream << "Hello! C++ streams are neat." << endl;
```

#### **wxTextWindow::wxTextWindow**

##### **void wxTextWindow(void)**

Constructor, for deriving classes.

**void wxTextWindow(wxWindow \*parent, int x = -1, int y = -1,  
int width = -1, int height = -1, long style = 0, char \*name = "textWindow")**

Constructor.

Under Windows and Motif, the parent can be either a frame or panel. Under XView, the parent must be a frame.

The parameters *x*, *y*, *width* and *height* can be omitted on construction if the position and size will later be set (for example by a application frame's **OnSize** callback, or if there is only one subwindow for the frame, in which case the subwindow fills the frame).

*style* is a bit list of some of the following:

<b>wxBORDER</b>	Use this style to draw a thin border in MS Windows (non-native implementation only).
<b>wxNATIVE_IMPL</b>	Use this style to allow editing under MS Windows, albeit with a 64K limitation.
<b>wxREADONLY</b>	Use this style to disable editing.
<b>wxHSCROLL</b>	Use this style to enable a horizontal scrollbar, or leave it out to allow line wrapping. Windows and Motif only.

The *name* parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual text windows.

**wxTextWindow::~~wxTextWindow****void ~wxTextWindow(void)**

Destructor. Deletes any stored text before deleting the physical window.

**wxTextWindow::Clear****void Clear(void)**

Clears the window and deletes the stored text.

**wxTextWindow::Create****Bool Create(wxWindow \*parent, int x = -1, int y = -1,  
int width = -1, int height = -1, long style = 0, char \*name = "textWindow")**

Creates the text item for two-step construction. Derived classes should call or replace this function. See *wxTextWindow::wxTextWindow* (page 302) for further details.

**wxTextWindow::Copy****Bool Copy(void)**

Copies the selected text onto the clipboard.

Motif and Windows only.

**wxTextWindow::Cut****Bool Cut(void)**

Copies the selected text onto the clipboard, and then deletes the text from the window.

Motif and Windows only.

**wxTextWindow::DiscardEdits****void DiscardEdits(void)**

Resets the internal 'modified' flag as if the current edits had been saved.

**wxTextWindow::GetContents****char \* GetContents(void)**

Gets a pointer to a newly allocated buffer containing the text window contents. Free the buffer with the C++ delete operator.



**wxTextWindow::GetInsertionPoint****long GetInsertionPoint(void)**

Gets the current insertion point.

Motif and XView only.

**wxTextWindow::GetLastPosition****long GetLastPosition(void)**

Gets the position representing the end of the text window contents.

Motif and XView only.

**wxTextWindow::GetLineLength****int GetLineLength(long *lineNo*)**

Gets the length of the specified line (starting from zero).

**wxTextWindow::GetLineText****int GetLineText(long *lineNo*, char \**buffer*)**

Puts the contents of the specified line (starting from zero) into the given buffer, returning the number of characters copied.

**wxTextWindow::GetNumberOfLines****int GetNumberOfLines(void)**

Gets the number of lines in the text window buffer.

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

Selects the text between *from* and *to*.

**wxTextWindow::LoadFile****Bool LoadFile(char \* *file*)**

Loads and displays the named file, if it exists. Success is indicated by a return value of TRUE.

**wxTextWindow::Modified****Bool Modified(void)**

Returns TRUE if the text has been modified.

**wxTextWindow::OnChar****void OnChar(wxKeyEvent& *event*)**

In Motif and Windows, 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 `wxTextWindow::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.

Under Windows, the `wxNATIVE_IMPL` flag must be passed to the `wxTextWindow` constructor if overriding `OnChar` is to have any effect.

See `wxEvtHandler::OnChar` (page 151) and `wxKeyEvent` (page 193) for more details of the keystroke event.

**wxTextWindow::Paste****Bool Paste(void)**

Pastes text from the clipboard into the text window.

Motif and Windows only.

**wxTextWindow::PositionToXY****long PositionToXY(long *pos*, long \**x*, long \**y*)**

Converts given character and line position to a position.

Motif and Windows only.

**wxTextWindow::Remove****void Remove(long *from*, long *to*)**

Removes the text between *from* and *to*.

**wxTextWindow::Replace**

**void Replace(long from, long to, char \*value)**

Replaces the text between *from* and *to* with *value*.

### **wxTextWindow::SaveFile**

**Bool SaveFile(char \* file)**

Saves the text in the named file. Success is indicated by a return value of TRUE.

### **wxTextWindow::SetFont**

**void SetFont(wxFont \*font)**

Sets the font for the text window.

Windows and Motif only.

### **wxTextWindow::SetEditable**

**void SetEditable(Bool editable)**

Determines whether the text window is user-editable

### **wxTextWindow::SetInsertionPoint**

**void SetInsertionPoint(long pos)**

Sets the current insertion point to *pos*.

### **wxTextWindow::SetInsertionPointEnd**

**void SetInsertionPointEnd(void)**

Sets the current insertion point to the end of the text.

Motif and XView only.

### **wxTextWindow::ShowPosition**

**void ShowPosition(long pos)**

Makes the line containing the given position visible.

Motif and XView only.

**wxTextWindow::WriteText****void WriteText(char \* text)**

Writes the text into the text window. Presently there is no means of writing text to other than the end of the existing text. Newlines in the text string are the only control characters allowed, and they will cause appropriate line breaks. See << (page 307) for more convenient ways of writing to the window.

**wxTextWindow::XYToPosition****long XYToPosition(long x, long y)**

Converts given character and line position to a position.

**wxTextWindow::operator <<****wxTextWindow& operator <<(char \*s)****wxTextWindow& operator <<(int i)****wxTextWindow& operator <<(long l)****wxTextWindow& operator <<(float f)****wxTextWindow& operator <<(double d)****wxTextWindow& operator <<(char c)**

Operator definitions for writing to a text window, for example:

```
wxTextWindow *wnd = new wxTextWindow(my_frame);  
  
(*wnd) << "Welcome to text window number " << 1 << "...\n";
```

**9.104. wxTimer: wxObject**

The wxTimer object is an abstraction of MS Windows, XView and X toolkit timers. 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.

See also `::wxStartTimer` (page 350) and `::wxGetElapsedTime` (page 346) for stopwatch functions.

**wxTimer::wxTimer****void wxTimer(void)**

Constructor.

**wxTimer::~~wxTimer**

**void ~wxTimer(void)**

Destructor. Stops the timer if activated.

**wxTimer::Interval**

**int Interval(void)**

Returns the current interval for the timer.

**wxTimer::Notify**

**void Notify(void)**

This member should be overridden by the user. It is called on timeout.

**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.

**wxTimer::Stop**

**void Stop(void)**

Stops the timer.

### 9.105. wxToolBar: wxPanel

See also *Overview* (page 391)

A wxToolBar is a canvas containing mouse-sensitive bitmaps. Include the file `wx_tbar.h` to use this class.

**Note:** under XView, wxToolBar inherits from wxCanvas, not wxPanel, due to limitations in the XView toolkit.

**wxToolBar::wxToolBar**

**void wxToolBar(wxWindow \*parent, int x = 0, int y = 0,  
int width = -1, int height = -1, long style = 0,  
int orientation = wxVERTICAL, int nRowsOrColumns = 1, char \*name = "toolBar")**

Constructs a toolbar canvas.

*parent* is a parent window, usually a `wxFrame`.

*x*, *y* set the position of the window.

*width*, *height* set the size of the window.

*style* is a bitlist, which may contain the following flags:

`wxB_3DBUTTONS` Gives a 3D look to the buttons, but not to the same extent as `wxButtonBar`.

*orientation* specifies a `wxVERTICAL` or `wxHORIZONTAL` orientation for laying out the toolbar.

*nRowsOrColumns* specifies the number of rows or columns, whose meaning depends on *orientation*. If laid out vertically, *nRowsOrColumns* specifies the number of rows to draw before the next column is started; if horizontal, it refers to the number of columns to draw before the next row is started.

*name* specifies a window name for the toolbar.

Under Windows 95, the `wxButtonBar` constructor only accepts `wxVERTICAL` plus the number of rows.

## **wxToolBar::~~wxToolBar**

**void ~wxToolBar(void)**

Toolbar destructor.

## **wxToolBar::AddSeparator**

**void AddSeparator()**

Adds a separator for spacing groups of tools.

## **wxToolBar::AddTool**

**wxToolBarTool \* AddTool(int toolIndex, wxBitmap \* bitmap1, wxBitmap \* bitmap2 = NULL, Bool isToggle = FALSE, float xPos = -1, float yPos = -1, wxObject \* clientData = NULL, char \* shortHelpString = NULL, char \* longHelpString = NULL)**

Adds a tool to the toolbar. The *toolIndex* is 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.

The first bitmap is the primary tool bitmap for toggle and button tools. 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).

The arguments *xPos* and *yPos* allow the programmer to specify the position of the tool if automatic layout is not suitable. For example, a toolbar along the top of a window may have groups of tools, with spacing between groups. In this case, specifying *xPos* is all that is required, and *yPos* will take its value from the current vertical spacing value.

*clientData* is an optional pointer to client data which can be retrieved later using **GetToolClientData**.

*shortHelpString* is used for displaying a tooltip for the tool in the Windows 95 implementation of `wxButtonBar`.

*longHelpString* can be used to display longer help, such as status line help.

## **wxToolBar::CreateTools**

### **Bool CreateTools()**

Required for the Windows 95 version of `wxButtonBar`: call this function after all tools have been added to the toolbar. It is harmless to call `CreateTools` for `wxToolBar`.

## **wxToolBar::DrawTool**

### **void DrawTool(wxMemoryDC& memDC, wxToolBarTool \*tool)**

Draws the specified tool onto the canvas using the given memory device context. Used internally, and should not need to be used by the programmer.

## **wxToolBar::EnableTool**

### **void EnableTool(int toolIndex, Bool enable)**

Enables or disables the tool. Not currently implemented, but should give some indication of whether the tool is active.

## **wxToolBar::FindToolForPosition**

### **wxToolBarTool \* FindToolForPosition(float x, float y)**

Find a tool for the given mouse position, or return NULL. Used internally, and should not need to be used by the programmer.

## **wxToolBar::GetMaxSize**

### **void GetMaxSize(float \*w, float \*h)**

Gets the maximum size taken up by the tools after layout, including margins. This can be used to size a frame around the toolbar canvas.

## **wxToolBar::GetToolClientData**

**wxObject \* GetToolClientData(int toolIndex)**

Get any client data associated with the tool.

### **wxToolBar::GetToolEnabled**

**Bool GetToolEnabled(int toolIndex)**

Returns TRUE if the tool is enabled, FALSE otherwise.

### **wxToolBar::GetToolLongHelp**

**char \* GetToolLongHelp(int toolIndex)**

Returns the long help for the given tool.

### **wxToolBar::GetToolShortHelp**

**char \* GetToolShortHelp(int toolIndex)**

Returns the short help for the given tool.

### **wxToolBar::GetToolState**

**Bool GetToolState(int toolIndex)**

Gets the on/off state of a toggle tool.

### **wxToolBar::Layout**

**void Layout(void)**

Called by the application after the tools have been added to automatically lay the tools out on the canvas. If you have given absolute positions when adding the tools, do not call this.

### **wxToolBar::OnLeftClick**

**Bool OnLeftClick(int toolIndex, Bool toggleDown)**

Called when the user clicks on a tool with the left mouse button. The programmer should override this function to detect left tool clicks. *toolIndex* is the identifier passed to **AddTool**, and *toggleDown* is TRUE if the tool is a toggle and the toggle is down, otherwise is FALSE.

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.



**wxToolBar::OnMouseEnter****void OnMouseEnter**(int *toolIndex*)

This is called when the mouse moves into a tool (*toolIndex* is greater than -1) or out of the toolbox (*toolIndex* is -1). The programmer can override this to provide extra information about the tool, such as a short description on the status line. Icons are not always as intuitive as they're cracked up to be!

**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. *toolIndex* is the identifier passed to **AddTool**, and *x* and *y* give the mouse cursor position.

A typical use of this member might be to pop up a menu.

**wxToolBar::SetMargins****void SetMargins**(int *x*, int *y*)

Set the values to be used as margins for the toolbar. 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.

**wxToolBar::SetToolLongHelp****void SetToolLongHelp**(int *toolIndex*, char \**helpString*)

Sets the long help for the given tool.

**wxToolBar::SetToolPacking****void SetToolPacking**(int *packing*)

Sets the value used for spacing tools. The default value is 1.

**wxToolBar::SetToolShortHelp****void SetToolShortHelp**(int *toolIndex*, char \**helpString*)

Sets the short help for the given tool.

**wxToolBar::SetToolSeparation****void SetToolSeparation**(int *separation*)

Sets the value used by tool separators. The default value is 5.

### **wxToolBar::ToggleTool**

**void ToggleTool**(int *toolIndex*, **Bool** *toggle*)

Toggles a tool on or off.

### **9.106. wxTypeTree: wxList**

*OBSOLETE CLASS.* Please see the *run time class information* (page 370) for an alternative type system.

wxTypeTree implements an explicit type hierarchy which can be useful for querying C++ types at run-time, usually by calling *wxSubType* (page 350) using the *wxObject::\_\_type* (page 222) member.

A type is added to the global variable **wxAllTypes**; wxWindows adds its own standard types on initialization, in **wxInitStandardTypes**, but the application can add its own.

The standard wxWindows types, grouped by functionality, are:

- wxTYPE\_ANY
- wxTYPE\_OBJECT (an alias for wxTYPE\_ANY)
- wxTYPE\_WINDOW
- wxTYPE\_DIALOG\_BOX
- wxTYPE\_ITEM
- wxTYPE\_PANEL
- wxTYPE\_CANVAS
- wxTYPE\_TEXT\_WINDOW
- wxTYPE\_FRAME
- wxTYPE\_BUTTON
- wxTYPE\_TEXT
- wxTYPE\_MESSAGE
- wxTYPE\_CHOICE
- wxTYPE\_LIST\_BOX
- wxTYPE\_SLIDER
- wxTYPE\_CHECK\_BOX
- wxTYPE\_MENU
- wxTYPE\_MENU\_BAR
- wxTYPE\_MULTI\_TEXT
- wxTYPE\_RADIO\_BOX
- wxTYPE\_EVENT
- wxTYPE\_DC
- wxTYPE\_DC\_CANVAS
- wxTYPE\_DC\_POSTSCRIPT
- wxTYPE\_DC\_PRINTER
- wxTYPE\_DC\_METAFILE
- wxTYPE\_DC\_MEMORY
- wxTYPE\_MOUSE\_EVENT
- wxTYPE\_KEY\_EVENT

- wxTYPE\_COMMAND\_EVENT
- wxTYPE\_PEN
- wxTYPE\_BRUSH
- wxTYPE\_FONT
- wxTYPE\_ICON
- wxTYPE\_BITMAP
- wxTYPE\_METAFILE
- wxTYPE\_TIMER
- wxTYPE\_COLOUR
- wxTYPE\_COLOURMAP
- wxTYPE\_CURSOR
- wxTYPE\_DDE\_CLIENT
- wxTYPE\_DDE\_SERVER
- wxTYPE\_DDE\_CONNECTION
- wxTYPE\_HELP\_INSTANCE
- wxTYPE\_LIST
- wxTYPE\_STRING\_LIST
- wxTYPE\_HASH\_TABLE
- wxTYPE\_NODE
- wxTYPE\_APP

### **wxTypeTree::wxTypeTree**

#### **void wxTypeTree(void)**

Constructor. Used by wxWindows only, since there only one instance of this class.

### **wxTypeTree::AddType**

#### **void AddType(WXTYPE *newType*, WXTYPE *parentType*, char \**name*)**

Adds a type to the hierarchy. *newType* is the type being registered, *parentType* is the parent type, and *name* is an identifier (which can be used in error messages). The top (root) type is wxTYPE\_ANY.

Example:

```
wxAllTypes.AddType(wxTYPE_WINDOW, wxTYPE_ANY, "window");
wxAllTypes.AddType(wxTYPE_PANEL, wxTYPE_WINDOW, "panel");
wxAllTypes.AddType(wxTYPE_CANVAS, wxTYPE_WINDOW, "canvas");
```

### **wxTypeTree::GetName**

#### **char \* GetName(WXTYPE *typ*)**

Gets a temporary pointer to the name of the given type, or NULL if the type is not found.

## **9.107. wxUpdateIterator: wxObject**

This class is used to iterate through all damaged regions of a canvas, panel or dialog box, 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 *wxCanvas::OnPaint* (page 63) for an example of use.

### **wxUpdateliterator::wxUpdateliterator**

**void wxUpdateliterator(wxCanvas \*canvas)**

Creates an iterator object.

### **wxUpdateliterator::GetX**

**int GetX(void)**

Returns the x value for the current region.

### **wxUpdateliterator::GetY**

**int GetY(void)**

Returns the y value for the current region.

### **wxUpdateliterator::GetWidth**

**int GetWidth(void)**

Returns the width value for the current region.

### **wxUpdateliterator::GetHeight**

**int GetWidth(void)**

Returns the width value for the current region.

### **wxUpdateliterator::operator ++**

**void operator ++(void)**

Increments the iterator to the next region.

## **9.108. wxView: wxEvtHandler**

See also *Overview* (page 374)

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* (page 140), *wxDocTemplate* (page 135) and *wxDocManager* (page 128) classes.

### **wxView::viewDocument**

**wxDocument \* 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.

### **wxView::viewFrame**

**wxFrame \* viewFrame**

Frame associated with the view, if any.

### **wxView::viewTypeName**

**char \* 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.

### **wxView::wxView**

**void wxView(void)**

Constructor. Define your own default constructor to initialize application-specific data.

### **wxView::~~wxView**

**void ~wxView(void)**

Destructor. Removes itself from the document's list of views.

### **wxView::Activate**

**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*.

**wxView::Close****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.

**wxView::GetDocument****wxDocument \*** **GetDocument**(*void*)

Gets a pointer to the document associated with the view.

**wxView::GetDocumentManager****wxDocumentManager \*** **GetDocumentManager**(*void*)

Returns a pointer to the document manager instance associated with this view.

**wxView::GetFrame****wxFrame \*** **GetFrame**(*void*)

Gets the frame associated with the view (if any).

**wxView::GetViewName****char \*** **GetViewName**(*void*)

Gets the name associated with the view (passed to the `wxDocTemplate` constructor). Not currently used by the framework.

**wxView::OnActivateView****void OnActivateView**(*Bool activate, wxView \*activeView, wxView \*deactiveView*)

Called when a view is activated by means of `wxView::Activate`. The default implementation does nothing.

**wxView::OnChangeFilename****void OnChangeFilename**(*void*)

Called when the filename has changed. The default implementation constructs a suitable title and sets the title of the view frame (if any).

## **wxView::OnClose**

**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 canvas, you need to disassociate the canvas from the view and perhaps clear the canvas. If *deleteWindow* is TRUE, delete the frame associated with the view.

## **wxView::OnCreate**

**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.

## **wxView::OnCreatePrintout**

**wxPrintout \* OnCreatePrintout**(void)

If the printing framework is enabled in the library, this function returns a *wxPrintout* (page 251) 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.

## **wxView::OnUpdate**

**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.

## **wxView::SetDocument**

**void SetDocument**(wxDocument \**doc*)

Associates the given document with the view. Normally called by the framework.

## **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.

**wxView::SetViewName**

**void SetViewName(char \*name)**

Sets the view type name. Should only be called by the framework.

### **9.109. wxWindow: wxObject**

See also *Event handling overview* (page 390)

wxWindow is the base class for all windows and panel items. Any children of the window will be deleted automatically by the destructor before the window itself is deleted.

**wxWindow::wxWindow**

**void wxWindow(void)**

Constructor.

**wxWindow::~~wxWindow**

**void ~wxWindow(void)**

Destructor. Deletes all subwindows, then deletes itself.

**wxWindow::AddChild**

**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.

**wxWindow::CaptureMouse**

**void CaptureMouse(void)**

Directs all mouse input to this window. Call **ReleaseMouse** to release the capture.

**wxWindow::Center**

**void Center(int direction)**

See *Centre* (page 320).



**wxWindow::Centre****void Centre**(int *direction*)

Centres the window. The parameter may be `wxHORIZONTAL`, `wxVERTICAL` or `wxBOTH`.

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.

**wxWindow::ClientToScreen****void ClientToScreen**(int \*x, int \*y)

Converts the values pointed to by *x* and *y* to screen coordinates from coordinates relative to this window.

**wxWindow::Close****Bool Close**(Bool *force*)

Applies to managed windows (`wxFrames` and `wxDialogBoxes`) only. The purpose of this call is to provide a more elegant way of destroying a window than using the *delete* operator.

Close calls `OnClose` for the window, providing an opportunity for the window to veto the close.

If `OnClose` returns `FALSE` and *force* is `FALSE`, then `FALSE` is returned and no deletion occurs.

If *force* is `TRUE`, then regardless of the return value of `OnClose`, the window will be added to a list for deletion during application idle time. The window will therefore not be deleted immediately, and the object can be used just after calling `Close`.

This has an important benefit under X, where immediate deletion of a window can cause problems if events for that window and its children are still pending. Using `Close` should eliminate these problems since the window will not be deleted until there are no more X events pending.

**wxWindow::DestroyChildren****void DestroyChildren**(void)

Destroys all children of a window. Called automatically by the destructor.

**wxWindow::DragAcceptFiles****void DragAcceptFiles**(Bool *accept*)

Under Windows, if *accept* is `TRUE`, makes the window eligible for a **OnDropFiles** event.

**wxWindow::Enable**

**void Enable**(**Bool** *enable*)

Enable or disable the window, so input has no effect.

### **wxWindow::GetCharHeight**

**float GetCharHeight**(**void**)

Get the character height for this window.

### **wxWindow::GetCharWidth**

**float GetCharWidth**(**void**)

Get the average character width for this window.

### **wxWindow::GetChildren**

**wxList \* GetChildren**(**void**)

Returns a temporary pointer to a list of the window's children.

### **wxWindow::GetClientSize**

**void GetClientSize**(**int** \**width*, **int** \**height*)

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.

### **wxWindow::GetConstraints**

**wxLayoutConstraints \* GetConstraints**(**void**)

Gets a pointer to the window's layout constraints (if any).

### **wxWindow::GetEventHandler**

**wxEvtHandler \* GetEventHandler**(**void**)

Gets the event handler for this window. By default, the window is its own event handler. Use this function if you wish to manually call an event handler function (such as `OnPaint`).

See *wxEvtHandler* (page 150).

### **wxWindow::GetGrandParent**

**wxWindow \* GetGrandParent(void)**

Returns the grandparent of a window, if any.

**wxWindow::GetHandle****char \* GetHandle(void)**

Gets the platform-specific handle of the physical window.

**wxWindow::GetPosition****void GetPosition(int \*x, int \*y)**

This gets the position of the window in pixels, relative to the parent window or if no parent, relative to the whole display.

**wxWindow::GetLabel****char \* GetLabel(void)**

Generic way of getting a label from any window, perhaps for identification purposes. Some windows do not have labels (such as subwindows), but frames, dialogs and panel items all have labels, where the interpretation of what constitutes a label differs from class to class. This can be useful for meta-programs (such as testing tools or special-needs access programs) which need to identify windows by name rather than visually.

**wxWindow::GetName****char \* GetName(void)**

Returns a pointer to internal data containing the window's name. This name is not guaranteed to be unique; it is up to the programmer to supply an appropriate name.

**wxWindow::GetParent****wxWindow \* GetParent(void)**

Returns the parent of a window, if any.

**wxWindow::GetSize****void GetSize(int \*width, int \*height)**

This gets the size of the entire window in pixels.

**wxWindow::GetTextExtent**

**void GetTextExtent(char \*string, float \*x, float \*y)**

Gets the width and height of the string as it would be drawn on the window with the currently selected font.

### **wxWindow::GetUserEditMode**

**Bool GetUserEditMode(void)**

Returns TRUE if the window is in user interface edit mode.

See *wxWindow::SetUserEditMode* (page 327) for further details.

### **wxWindow::GetWindowStyleFlag**

**long GetWindowStyleFlag(void)**

Gets the window style that was passed to the constructor or **Create** member.

### **wxWindow::IsShown**

**Bool IsShown(void)**

Returns TRUE if the window is shown, FALSE if it has been hidden.

### **wxWindow::Layout**

**void Layout(void)**

Invokes the constraint-based layout algorithm for this window. It is called automatically by the default **wxWindow::OnSize** member.

### **wxWindow::Lower**

**void Lower(void)**

Lowers the window to the bottom of the window hierarchy if it is a managed window (dialog or frame). Motif and Windows only.

### **wxWindow::MakeModal**

**void MakeModal(Bool flag)**

If *flag* is TRUE, this call disables all other windows in the application so that the user can only interact with this window. If *flag* is FALSE, the effect is reversed.

**wxWindow::Move****void Move(int x, int y)**

Moves the window to the given position.

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);
```

**wxWindow::PopupMenu****Bool PopupMenu(wxMenu \*menu, float x, float 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 `commandId` member of the event to check the selected menu identifier.

Valid only for subwindows (panels, canvases and text windows).

See also *wxMenu* (page 206).

This function should now function correctly under Motif, removing the need to use *FakePopupMenu*.

**wxWindow::Raise****void Raise(void)**

Raises the window to the top of the window hierarchy if it is a managed window (dialog or frame). Motif and Windows only.

**wxWindow::Refresh****void Refresh(Bool eraseBackground = TRUE, wxRectangle \*rect = NULL)**

Causes a message or event to be generated to repaint the window. If *eraseBackground* is `TRUE`, the background will be erased. If *rect* is non-NULL, only the given rectangle will be treated as damaged.

**wxWindow::ReleaseMouse****void ReleaseMouse(void)**

Releases mouse input captured with **CaptureMouse**.

**wxWindow::ScreenToClient**

**void ScreenToClient(int \*x, int \*y)**

Converts the values pointed to by *x* and *y* to window coordinates from screen coordinates.

### **wxWindow::SetAutoLayout**

**void SetAutoLayout(Bool autoLayout)**

Set this to TRUE if you wish the Layout function to be called from within wxWindow::OnSize functions (when the window is resized).

See also *SetConstraints* (page 325).

### **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.

Pass NULL to this function to disassociate and delete the window's constraints.

You must call *wxWindow::SetAutoLayout* (page 325) to tell a window to use the constraints automatically in OnSize; otherwise, you must override OnSize and call Layout explicitly.

### **wxWindow::SetDoubleClick**

**void SetDoubleClick(Bool allowDoubleClick)**

For canvases, allows double click if *allowDoubleClick* is TRUE. The default is FALSE.

### **wxWindow::SetFocus**

**void SetFocus(void)**

This sets the window to receive keyboard input. The only panel item that will respond to this under XView is the **wxText** item and derived items.

### **wxWindow::SetName**

**void SetName(char \*name)**

Sets the window's name.

### **wxWindow::SetSize**

**void SetSize(int x, int y, int width, int height, int sizeFlags = wxSIZE\_AUTO)**

**void SetSize(int width, int height)**

This sets the size and/or position of the entire window in pixels. 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. The behaviour is controlled by *sizeFlags*, which is a bit list of the following:

- `wxSIZE_AUTO_WIDTH` (1): a -1 width value is taken to indicate a wxWindows-supplied default width.
- `wxSIZE_AUTO_HEIGHT` (2): a -1 height value is taken to indicate a wxWindows-supplied default width.
- `wxSIZE_AUTO` (3): -1 size values are taken to indicate a wxWindows-supplied default size.
- `wxSIZE_USE_EXISTING` (0): 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.

In versions of wxWindows prior to 1.61 (c), it was not always clear what interpretation was being used. Now implementations of `SetSize` can also implicitly implement the `wxWindow::Move` function, which is defined as the call:

```
SetSize(x, y, -1, -1, wxSIZE_USE_EXISTING);
```

**wxWindow::SetSizeHints****void SetSizeHints(int minW=-1, int minH=-1, int maxW=-1, int maxH=-1, int incW=-1, int incH=-1)**

Sets the minimum and maximum frame size under MS Windows, Motif and XView. Under Motif and XView the width and height resizing increments can also be set.

If a pair of values is not set (or set to -1), the default values will be used.

**wxWindow::SetClientSize****void SetClientSize(int width, int height)**

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 **SetSize**, 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.

**wxWindow::SetColourMap****void SetColourMap(wxColourMap \*colourMap)**

Assigns the given colourmap to the window.

See *wxColourMap* (page 81) for further details.

### **wxWindow::SetCursor**

**wxCursor \* SetCursor(wxCursor \*cursor)**

Sets the window's cursor, returning the previous cursor (if any). This function applies to all subwindows.

See also *::wxSetCursor* (page 336), *wxCursor* (page 94).

### **wxWindow::SetEventHandler**

**void SetEventHandler(wxEvtHandler \*handler)**

Sets the event handler for this window. By default, the window is its own event handler. See *wxEvtHandler* (page 150).

### **wxWindow::SetTitle**

**void SetTitle(char \*title)**

Sets the window's title, allocating its own string storage. Currently applicable only to frames.

### **wxWindow::SetUserEditMode**

**void SetUserEditMode(Bool editable)**

Sets a flag indicating that the user can 'edit' the interface. This is for the use of visual user interface building tools, and currently only works for *wxPanel* and *wxDialogBox*.

When this mode is on, the windows stop having their normal functionality and instead receives *OnEvent* calls that the application can intercept. In particular, the base *wxPanel* class implements panel item moving and sizing, and passes left and right click events from the items and panel to the application via functions such as *OnLeftClick*. The application might pop up a menu or select/deselect items in response to these calls.

See also *wxWindow::GetUserEditMode* (page 323).

### **wxWindow::Show**

**Bool Show(Bool show)**

If *show* is *TRUE*, displays the window and brings it to the front. Otherwise, hides the window.



## 10. Functions

The functions defined in `wxWindows` are described here.

### 10.1. File functions

See also `wxPathList` (page 235).

#### **::wxDirExists**

**Bool wxDirExists(char \*dirname)**

Returns TRUE if the directory exists.

#### **::Dos2UnixFilename**

**void Dos2UnixFilename(char \*s)**

Converts a DOS to a UNIX filename by replacing backslashes with forward slashes.

#### **::wxFileExists**

**Bool wxFileExists(char \*filename)**

Returns TRUE if the file exists.

#### **::wxFileNameFromPath**

**char \* wxFileNameFromPath(char \*path)**

Returns a temporary pointer to the filename for a full path. Copy this pointer for long-term use.

#### **::wxFindFirstFile**

**char \* wxFindFirstFile(const char \*spec, int flags = 0)**

This function does directory searching; returns the first file that matches the path *spec*, or NULL. Use `wxFindNextFile` (page 329) to get the next matching file.

*spec* may contain wildcards.

*flags* is reserved for future use.

The returned filename is a pointer to static memory so should not be freed.

For example:

```
char *f = wxFindFirstFile("/home/project/*.");
while (f)
{
    ...
    f = wxFindNextFile();
}
```

### **::wxFindNextFile**

**char \* wxFindFirstFile(void)**

Returns the next file that matches the path passed to *wxFindFirstFile* (page 328).

### **::wxIsAbsolutePath**

**Bool wxIsAbsolutePath(char \*filename)**

Returns TRUE if the argument is an absolute filename, i.e. with a slash or drive name at the beginning.

### **::wxPathOnly**

**char \* wxPathOnly(char \*path)**

Returns a temporary pointer to the directory part of the filename. Copy this pointer for long-term use.

### **::wxUnix2DosFilename**

**void wxUnix2DosFilename(char \*s)**

Converts a UNIX to a DOS filename by replacing forward slashes with backslashes.

### **::wxConcatFiles**

**Bool wxConcatFiles(char \*file1, char \*file2, char \*file3)**

Concatenates *file1* and *file2* to *file3*, returning TRUE if successful.

### **::wxCopyFile**

**Bool wxCopyFile(char \*file1, char \*file2)**

Copies *file1* to *file2*, returning TRUE if successful.

### **::wxGetHostName**

**Bool wxGetHostName(char \*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.

### **::wxGetEmailAddress**

**Bool wxGetEmailAddress(char \*buf, int sz)**

Copies the user's email address into the supplied buffer, by concatenating the values returned by *wxGetHostName* (page 329) and *wxGetUserId* (page 330).

Returns TRUE if successful, FALSE otherwise.

### **::wxGetUserId**

**Bool wxGetUserId(char \*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.

### **::wxGetUserName**

**Bool wxGetUserName(char \*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.

### **::wxGetWorkingDirectory**

**char \* wxGetWorkingDirectory(char \*buf=NULL, int sz=1000)**

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.

**::wxGetTempFileName****char \* wxGetTempFileName(char \*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.

**::wxIsWild****Bool wxIsWild(char \*pattern)**

Returns TRUE if the pattern contains wildcards. See *wxMatchWild* (page 331).

**::wxMatchWild****Bool wxMatchWild(char \*pattern, char \*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* (page 331).

**::wxMkdir****Bool wxMkdir(char \*dir)**

Makes the directory *dir*, returning TRUE if successful.

**::wxRemoveFile****Bool wxRemoveFile(char \*file)**

Removes *file*, returning TRUE if successful.

**::wxRenameFile****Bool wxRenameFile(char \*file1, char \*file2)**

Renames *file1* to *file2*, returning TRUE if successful.

**::wxRmdir****Bool wxRmdir(char \*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.

### **::wxSetWorkingDirectory**

**Bool wxSetWorkingDirectory(char \*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.

## **10.2. String functions**

### **::copystring**

**char \* copystring(char \*s)**

Makes a copy of the string *s* using the C++ new operator, so it can be deleted with the *delete* operator.

### **::wxStringMatch**

**Bool wxStringMatch(char \*s1, char \*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.

### **::wxStringEq**

**Bool wxStringEq(char \*s1, char \*s2)**

A macro defined as:

```
#define wxStringEq(s1, s2) (s1 && s2 && (strcmp(s1, s2) == 0))
```

### **::wxTransferFileToStream**

**Bool wxTransferFileToStream(char \*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.

### **::wxTransferStreamToFile**

**Bool wxTransferStreamToFile(istream& stream char \*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.

### 10.3. 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.

#### **::wxFileSelector**

```
char * wxFileSelector(char *message, char *default_path = NULL,  
char *default_filename = NULL, char *default_extension = NULL,  
char *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:

```
char *s = wxFileSelector("Choose a file to open");  
if (s)  
{  
    ...  
}
```

Remember that the returned pointer is temporary and should be copied if other `wxWindows` calls will be made before the value is to be used.

#### **::wxGetTextFromUser**

```
char * wxGetTextFromUser(char *message, char *caption = "Input text",  
char *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 NULL.

If *centre* is TRUE, the message text (which may include new line characters) is centred; if FALSE, the message is left-justified.

### **::wxGetMultipleChoice**

```
int wxGetMultipleChoice(char *message, char *caption, int n, char *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.

### **::wxGetSingleChoice**

```
char * wxGetSingleChoice(char *message, char *caption, int n, char *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 NULL.

*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.

### **::wxGetSingleChoiceIndex**

```
int wxGetSingleChoiceIndex(char *message, char *caption, int n, char *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.

### **::wxGetSingleChoiceData**

```
char * wxGetSingleChoiceData(char *message, char *caption, int n, char *choices[],
    char *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.

### **::wxMessageBox**

```
int wxMessageBox(char *message, char *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.

## **10.4. GDI functions**

The following are relevant to the GDI (Graphics Device Interface).

### **::wxColourDisplay**



**Bool wxColourDisplay(void)**

Returns TRUE if the display is colour, FALSE otherwise.

**::wxDisplayDepth****int wxDisplayDepth(void)**

Returns the depth of the display (a value of 1 denotes a monochrome display).

**::wxMakeMetaFilePlaceable**

**Bool wxMakeMetaFilePlaceable(char \*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* (page 213)) 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 *MM\_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.

**::wxSetCursor**

**void wxSetCursor(wxCursor \*cursor)**

Globally sets the cursor; only has an effect in MS Windows. See also *wxCursor* (page 94), *wxWindow::SetCursor* (page 327).

**10.5. System event functions**

The *wxWindows* system event implementation is incomplete and experimental, but is intended to be a platform-independent way of intercepting and sending events, including defining application-specific events and handlers.

Ultimately it is intended to be used as a way of testing *wxWindows* applications using scripts, although there are currently problems with this (especially with modal dialogs).

All this is documented more to provoke comments and suggestions, and jog my own memory, rather than to be used, since it has not been tested. However **wxSendEvent** will probably work if you instantiate the event structure properly for a command event type (see the code in `wb_panel.cpp` for `wxPanel::OnDefaultAction` (page 231) which uses **wxSendEvent** to send a command to the default button).

## **::wxAddPrimaryEventHandler**

**Bool wxAddPrimaryEventHandler(wxEventHandler handlerFunc)**

Add a primary event handler---the normal event handler for this event. For built-in events, these would include moving and resizing windows. User-defined primary events might include the code to select an image in a diagram (which could of course be achieved by a series of external events for mouse-clicking, but would be more difficult to specify and less robust).

Returns TRUE if it succeeds.

An event handler takes a pointer to a `wxEvent` and a boolean flag which is TRUE if the event was externally generated, and returns a boolean which is TRUE if that event was handled.

## **::wxAddSecondaryEventHandler**

**Bool wxAddSecondaryEventHandler(wxEventHandler handlerFunc, Bool pre, Bool override, Bool append)**

Add a secondary event handler, `pre = TRUE` iff it should be called before the event is executed. `override = TRUE` iff the handler is allowed to override all subsequent events by returning TRUE. Returns TRUE if succeeds.

A secondary event handler is an application-defined handler that may intercept normal events, possibly overriding them. A primary event handler provides the normal behaviour for the event.

An event handler takes a pointer to a `wxEvent` and a boolean flag which is TRUE if the event was externally generated, and returns a boolean which is TRUE if that event was handled.

## **::wxNotifyEvent**

**Bool wxNotifyEvent(wxEvent& event, Bool pre)**

Notify the system of the event you are about to execute/have just executed. If TRUE is returned and `pre = TRUE`, the calling code should not execute the event (since it has been intercepted by a handler and vetoed).

These events are always internal, because they're generated from within the main application code.

## **::wxRegisterEventClass**

**void wxRegisterEventClass(WXTYPE eventId, WXTYPE superClassId, wxEventConstructor constructor, char \*description)**

Register a new event class (derived from `wxEvent`), giving the new event class type, its superclass, a function for creating a new event object of this class, and an optional description.

### **::wxRegisterEventName**

```
void wxRegisterEventName(WXTYPE eventTypeId, WXTYPE eventClassId,  
    char *eventName)
```

Register the name of the event. This will allow a simple command language where giving the event type name and some arguments will cause a new event of class *eventClassId* to be created, with given event type, and some arguments, allows an event to be dynamically constructed and sent.

### **::wxRegisterExternalEventHandlers**

```
void wxRegisterExternalEventHandlers(void)
```

Define this and link before `wxWindows` library to allow registering events from 'outside' the main application.

### **::wxRemoveSecondaryEventHandler**

```
Bool wxRemoveSecondaryEventHandler(wxEventHandler handlerFunc, Bool pre)
```

Remove a secondary event handler. Returns TRUE if it succeeds.

### **::wxSendEvent**

```
Bool wxSendEvent(wxEvent& event, Bool external)
```

Send an event to the system; usually it will be external, but set external to FALSE if calling from within the main application in response to other events.

Returns TRUE if the event was processed.

## **10.6. Printer settings**

The following functions are used to control PostScript printing. Under Windows, PostScript output can only be sent to a file.

### **::wxGetPrinterCommand**

```
char * wxGetPrinterCommand(void)
```

Gets the printer command used to print a file. The default is `lpr`.

### **::wxGetPrinterFile**

**char \* wxGetPrinterFile(void)**

Gets the PostScript output filename.

**::wxGetPrinterMode**

**int wxGetPrinterMode(void)**

Gets the printing mode controlling where output is sent (PS\_PREVIEW, PS\_FILE or PS\_PRINTER). The default is PS\_PREVIEW.

**::wxGetPrinterOptions**

**char \* wxGetPrinterOptions(void)**

Gets the additional options for the print command (e.g. specific printer). The default is nothing.

**::wxGetPrinterOrientation**

**int wxGetPrinterOrientation(void)**

Gets the orientation (PS\_PORTRAIT or PS\_LANDSCAPE). The default is PS\_PORTRAIT.

**::wxGetPrinterPreviewCommand**

**char \* wxGetPrinterPreviewCommand(void)**

Gets the command used to view a PostScript file. The default depends on the platform.

**::wxGetPrinterScaling**

**void wxGetPrinterScaling(float \*x, float \*y)**

Gets the scaling factor for PostScript output. The default is 1.0, 1.0.

**::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.

**::wxSetPrinterCommand**

**void wxSetPrinterCommand(char \*command)**

Sets the printer command used to print a file. The default is `lpr`.

**::wxSetPrinterFile****void wxSetPrinterFile(char \*filename)**

Sets the PostScript output filename.

**::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.

**::wxSetPrinterOptions****void wxSetPrinterOptions(char \*options)**

Sets the additional options for the print command (e.g. specific printer). The default is nothing.

**::wxSetPrinterOrientation****void wxSetPrinterOrientation(int orientation)**

Sets the orientation (PS\_PORTRAIT or PS\_LANDSCAPE). The default is PS\_PORTRAIT.

**::wxSetPrinterPreviewCommand****void wxSetPrinterPreviewCommand(char \*command)**

Sets the command used to view a PostScript file. The default depends on the platform.

**::wxSetPrinterScaling****void wxSetPrinterScaling(float x, float y)**

Sets the scaling factor for PostScript output. The default is 1.0, 1.0.

**::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.

**10.7. Clipboard functions**

These clipboard functions are implemented for Windows only.

**::wxClipboardOpen****Bool wxClipboardOpen(void)**

Returns TRUE if this application has already opened the clipboard.

**::wxCloseClipboard****Bool wxCloseClipboard(void)**

Closes the clipboard to allow other applications to use it.

**::wxEmptyClipboard****Bool wxEmptyClipboard(void)**

Empties the clipboard.

**::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.

**::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.

**::wxGetClipboardFormatName****Bool wxGetClipboardFormatName(int dataFormat, char \*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.

**::wxIsClipboardFormatAvailable****Bool wxIsClipboardFormatAvailable(int dataFormat)**

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

**::wxOpenClipboard****Bool wxOpenClipboard(void)**

Opens the clipboard for passing data to it or getting data from it.

**::wxRegisterClipboardFormat****int wxRegisterClipboardFormat(char \*formatName)**

Registers the clipboard data format name and returns an identifier.

**::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.

**10.8. Miscellaneous functions****::NewId****long NewId(void)**

Generates an integer identifier unique to this run of the program.

### **::RegisterId**

**void RegisterId(long id)**

Ensures that ids subsequently generated by **NewId** do not clash with the given **id**.

### **::wxBeginBusyCursor**

**void wxBeginBusyCursor(wxCursor \*cursor = wxHOURLASS\_CURSOR)**

Changes the cursor to the given cursor for all windows in the application. Use *wxEndBusyCursor* (page 344) 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* (page 348).

### **::wxBell**

**void wxBell(void)**

Ring the system bell.

### **::wxCleanUp**

**void wxCleanUp(void)**

Normally, *wxWindows* will call this cleanup function for you. However, if you call *wxEntry* (page 344) in order to initialize *wxWindows* manually, then you should also call *wxCleanUp* before terminating *wxWindows*, if *wxWindows* does not get a chance to do it.

### **::wxCreateDynamicObject**

**wxObject \* wxCreateDynamicObject(char \*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.

### **::wxDebugMsg**

**void wxDebugMsg(char \*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.

## **::wxDisplaySize**

**void wxDisplaySize(int \*width, int \*height)**

Gets the physical size of the display in pixels.

## **::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. See also `wxCleanUp` (page 343).

**void wxEntry(HANDLE hInstance, HANDLE hPrevInstance, char \*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, char \*commandLine)** `wxWindows` initialization under Windows (for applications constructed as a DLL).

**int wxEntry(int argc, char \*\*argv)**

`wxWindows` initialization under UNIX (XView or Motif).

## **::wxError**

**void wxError(char \*msg, char \*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` (page 345).

## **::wxEndBusyCursor**

**void wxEndBusyCursor(void)**

Changes the cursor back to the original cursor, for all windows in the application. Use with `wxBeginBusyCursor` (page 343).

See also `wxIsBusy` (page 348).

## **::wxExecute**

**long wxExecute(char \*command, Bool sync = FALSE)**

**long wxExecute(char \*\*argv, Bool sync = FALSE)**

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.

If execution is asynchronous, the return value is the process id, otherwise it is a status value. A zero value indicates that the command could not be executed.

See also *wxShell* (page 350).

## **::wxExit**

**void wxExit(void)**

Exits application after calling *wxApp::OnExit* (page 46). 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 *wxFrame::OnClose* (page 176) and *wxApp* (page 44).

## **::wxFatalError**

**void wxFatalError(char \*msg, char \*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* (page 344).

## **::wxFindMenuItemId**

**int wxFindMenuItemId(wxFrame \*frame, char \*menuString, char \*itemString)**

Find a menu item identifier associated with the given frame's menu bar.

## **::wxFindWindowByLabel**

**wxWindow \* wxFindWindowByLabel(char \*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.

## **::wxFindWindowByName**

**wxWindow \* wxFindWindowByName(char \*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.

### **::wxGetActiveWindow**

**wxWindow \* wxGetActiveWindow(void)**

Gets the currently active window (Windows only).

### **::wxGetDisplayName**

**char \* wxGetDisplayName(void)**

Under X only, returns the current display name. See also *wxSetDisplayName* (page 349).

### **::wxGetHomeDir**

**char \* wxGetHomeDir(char \*buf)**

Fills the buffer with a string representing the user's home directory (UNIX only).

### **::wxGetHostName**

**Bool wxGetHostName(char \*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".

### **::wxGetElapsedTime**

**long wxGetElapsedTime(Bool resetTimer = TRUE)**

Gets the time in milliseconds since the last *::wxStartTimer* (page 350).

If *resetTimer* is TRUE (the default), the timer is reset to zero by this call.

See also *wxTimer* (page 307).

### **::wxGetFreeMemory**

**long wxGetFreeMemory(void)**

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.

### **::wxGetMousePosition**

**void wxGetMousePosition(int\* x, int\* y)**

Returns the mouse position in screen coordinates.

### **::wxGetOsVersion**

**int wxGetOsVersion(int \*major = NULL, int \*minor = NULL)**

Gets operating system version information.

Platform	Return types
XView	Return value is wxXVIEW_X, <i>major</i> is X version, <i>minor</i> is X revision.
Macintosh	Return value is wxMACINTOSH.
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.

### **::wxGetResource**

**Bool wxGetResource(char \*section, char \*entry, char \*\*value, char \*file = NULL)**

**Bool wxGetResource(char \*section, char \*entry, float \*value, char \*file = NULL)**

**Bool wxGetResource(char \*section, char \*entry, long \*value, char \*file = NULL)**

**Bool wxGetResource(char \*section, char \*entry, int \*value, char \*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::wx\_class) 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* (page 351).

**::wxGetUserId****Bool wxGetUserId(char \*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".

**::wxGetUserName****Bool wxGetUserName(char \*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".

**::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.

**::wxInitClipboard****void wxInitClipboard(void)**

Initializes the generic clipboard system by creating an instance of the class *wxClipboard* (page 75).

**::wxIPCCleanUp****void wxIPCCleanUp(void)**

Call this when your application is terminating, if you have called *wxIPCInitialize* (page 348).

**::wxIPCInitialize****void wxIPCInitialize(void)**

Initializes for interprocess communication operation. May be called multiple times without harm.

See also *wxServer* (page 277), *wxClient* (page 74), *wxConnection* (page 91) and the relevant section of the user manual.

**::wxIsBusy**

**Bool wxIsBusy(void)**

Returns TRUE if between two *wxBeginBusyCursor* (page 343) and *wxEndBusyCursor* (page 344) calls.

**::wxLoadUserResource**

**char \* wxLoadUserResource(char \*resourceName, char \*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* (page 357).

This function is available under Windows only.

**::wxNow**

**char \* wxNow(void)**

Returns a string representing the current date and time.

**::wxPostDelete**

**void wxPostDelete(wxObject \*object)**

Under X, 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 X will still send events to the window.

Now obsolete: use *wxWindow::Close* (page 320) instead.

**::wxSetDisplayName**

**void wxSetDisplayName(char \*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* (page 346).

**::wxShell****Bool wxShell(const char \*command = NULL)**

Executes a command in an interactive shell window. If no command is specified, then just the shell is spawned.

See also *wxExecute* (page 344).

**::wxSleep****void wxSleep(int secs)**

Under X, sleeps for the specified number of seconds using the technique specified in the XView manual, not UNIX **sleep**.

**::wxStripMenuCodes****void wxStripMenuCodes(char \*in, char \*out)**

Strips any menu codes from *in* and places the result in *out*. Menu codes include & (mark the next character with an underline as a keyboard shortcut in Windows and Motif) and \t (tab in Windows).

**::wxStartTimer****void wxStartTimer(void)**

Starts a stopwatch; use *::wxGetElapsedTime* (page 346) to get the elapsed time.

See also *wxTimer* (page 307).

**::wxSubType****Bool wxSubType(WXTYPE type1, WXTYPE type2)**

*OBSOLETE FUNCTION*: please use *wxObject::IsKindOf* (page 222) instead.

TRUE if *type1* is a subtype of, or the same type as, *type2*. This can be useful when determining whether an object is an instance of a class derived from some other class, and its usage can make for generic code.

See also *wxTypeTree* (page 313) and *wxObject::\_\_type* (page 222).

Example:

```
wxNode *node = GetChildren()->First();
while (node)
{
    // Find a child that's a subwindow, but not a dialog box.
```

```
wxWindow *child = (wxWindow *)node->Data();
if ((wxSubType(child->__type, wxTYPE_PANEL) &&
    !wxSubType(child->__type, wxTYPE_DIALOG_BOX)) ||
    wxSubType(child->__type, wxTYPE_TEXT_WINDOW) ||
    wxSubType(child->__type, wxTYPE_CANVAS))
{
    child->SetFocus();
    return;
}
node = node->Next();
}
```

### **::wxToLower**

**char wxToLower(char ch)**

Converts the character to lower case. This is implemented as a macro for efficiency.

### **::wxToUpper**

**char wxToUpper(char ch)**

Converts the character to upper case. This is implemented as a macro for efficiency.

### **::wxTrace**

**void wxTrace(char \*fmt, ...)**

Takes printf-style variable argument syntax. Output is directed to the current output stream (see *wxDebugContext* (page 398)).

### **::wxTraceLevel**

**void wxTraceLevel(int level, char \*fmt, ...)**

Takes printf-style variable argument syntax. Output is directed to the current output stream (see *wxDebugContext* (page 398)). 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.

### **::wxWriteResource**

**Bool wxWriteResource(char \*section, char \*entry, char \*value, char \*file = NULL)**

**Bool wxWriteResource(char \*section, char \*entry, float value, char \*file = NULL)**

**Bool wxWriteResource(char \*section, char \*entry, long value, char \*file = NULL)**

**Bool wxWriteResource(char \*section, char \*entry, int value, char \*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* (page 347).

## **::wxYield**

### **Bool wxYield(void)**

Yields control to pending messages in the windowing system (has no effect under XView). 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.

## **10.9. Macros**

These macros are defined in wxWindows.

### **CLASSINFO**

#### **wxClassInfo \* CLASSINFO(className)**

Returns a pointer to the wxClassInfo object associated with this class.

### **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.

### **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:
        ...
};
```

## **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**.

## **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:
        char *frameTitle;
    public:
        ...
};
```

## **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)
```

```
{  
...  
}
```

## **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**.

## **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**.

## **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**.

## **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)  
{  
...  
}
```

## **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.

## WXTRACE

**WXTRACE**(formatString, ...)

Calls `wxTrace` with printf-style variable argument syntax. Output is directed to the current output stream (see *wxDebugContext* (page 398)).

## WXTRACELEVEL

**WXTRACELEVEL**(level, formatString, ...)

Calls `wxTraceLevel` with printf-style variable argument syntax. Output is directed to the current output stream (see *wxDebugContext* (page 398)). 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.

### 10.10. wxWindows resource functions

See also *wxWindows resource system* (page 414)

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* (page 414).

See also *wxPanel::LoadFromResource* (page 230) for panel and dialog loading from resource data.

#### **::wxResourceAddIdentifier**

**Bool wxResourceAddIdentifier**(char \*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.

#### **::wxResourceClear**

**void wxResourceClear**(void)

Clears the wxWindows resource table.

#### **::wxResourceCreateBitmap**

**wxBitmap \* wxResourceCreateBitmap**(char \*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 char *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");
```

### **::wxResourceCreateIcon**

**wxIcon \* wxResourceCreateIcon(char \*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 char *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");
```

### **::wxResourceCreateMenuBar**

**wxMenuBar \* wxResourceCreateMenuBar(char \*resource)**

Creates a new menu bar given a valid wxWindows menubar resource identifier. For example, if the .WXR file contains the following:

```
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']\
        ]\
    ]).\
    );
```

then this function can be called as follows:

```
wxMenuBar *menuBar = wxResourceCreateMenuBar("menuBar11");
```

### **::wxResourceGetIdentifier**

**int wxResourceGetIdentifier(char \*name)**

Used for retrieving the integer value associated with an identifier. A zero value indicates that the identifier was not found.

See *wxResourceAddIdentifier* (page 355).

**::wxResourceParseData****Bool wxResourceParseData(char \*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*.

**::wxResourceParseFile****Bool wxResourceParseFile(char \*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.

**::wxResourceParseString****Bool wxResourceParseString(char \*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 char *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* (page 349) to load an entire *.wxr* file into a string.

### **::wxResourceRegisterBitmapData**

**Bool wxResourceRegisterBitmapData(char \*name, char \*xbm\_data, int width, int height, wxResourceTable \*table = NULL)**

**Bool wxResourceRegisterBitmapData(char \*name, char \*\*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.

### **::wxResourceRegisterIconData**

Another name for *wxResourceRegisterBitmapData* (page 358).

## 11. Classes by category

A classification of wxWindows classes by category.

### 11.1. 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 *subwindows* (page 359), and dialog boxes have their own built-in subwindow similar to a panel.

- *wxFrame* (page 172)
- *wxDialogBox* (page 123)
- *wxEnhDialogBox* (page 146)

See also *Common dialogs* (page 359).

### 11.2. Subwindows

Subwindows should be created as children of frames. The panel subwindow may contain panel items (controls or widgets).

- *wxCanvas* (page 57)
- *wxPanel* (page 228)
- *wxTextWindow* (page 302)
- *wxToolBar* (page 308)
- *wxButtonBar* (page 55)
- *wxSplitterWindow* (page 280)

See also *wxWindow* (page 319).

### 11.3. Common dialogs

See also *Overview* (page 385)

Common dialogs are ready-made dialog classes or functions.

- *wxColourDialog* (page 80)
- *wxFileSelector* (page 333)
- *wxGetMultipleChoice* (page 334)
- *wxGetSingleChoice* (page 334)
- *wxGetSingleChoiceIndex* (page 334)
- *wxGetSingleChoiceData* (page 335)
- *wxGetTextFromUser* (page 333)
- *wxFontDialog* (page 162)
- *wxPrintDialog* (page 248)
- *wxPageSetupDialog* (page 227)
- *wxMessageBox* (page 335)

See also *wxDialogBox* (page 123).

### 11.4. Panel items

These are widgets (in Motif terminology) or controls (in MS Windows terminology) that can be placed on panels and dialog boxes, with the exception of *wxMenu* and *wxMenuBar*.



- *wxBUTTON* (page 54)
- *wxCHECKBOX* (page 68)
- *wxCHOICE* (page 69)
- *wxCOMBOBOX* (page 82)
- *wxGAUGE* (page 180)
- *wxGROUPBOX* (page 182)
- *wxITEM* (page 191)
- *wxLISTBOX* (page 201)
- *wxMULTITEXT* (page 219)
- *wxMENU* (page 206)
- *wxMENUBAR* (page 209)
- *wxMESSAGE* (page 211)
- *wxRADIOBOX* (page 260)
- *wxRADIOBUTTON* (page 263)
- *wxSLIDER* (page 278)
- *wxTEXT* (page 299)

See also *wxWindow* (page 319).

## 11.5. Window layout

See also *Overview* (page 388)

These are the classes and functions relevant to using automated window layout.

- *wxIndividualLayoutConstraint* (page 189)
- *wxLayoutConstraints* (page 196)
- *wxWindow::Layout* (page 323)
- *wxWindow::SetConstraints* (page 325)
- *wxWindow::GetConstraints* (page 321)

## 11.6. Device contexts

See also *Overview* (page 383)

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.

- *wxCanvasDC* (page 68)
- *wxDC* (page 108)
- *wxMemoryDC* (page 205)
- *wxMetaFileDC* (page 213)
- *wxPanelDC* (page 235)
- *wxPostScriptDC* (page 240)
- *wxPrinterDC* (page 250)

## 11.7. Graphics device interface

See also *Bitmaps overview* (page 384)

These classes are related to the Graphics Device Interface, in MS Windows terminology.

- *wxColour* (page 76)
- *wxBitmap* (page 48)
- *wxBrush* (page 51)
- *wxBrushList* (page 53)
- *wxCursor* (page 94)
- *wxFont* (page 158)
- *wxFontList* (page 163)
- *wxIcon* (page 183)
- *wxPen* (page 237)
- *wxPenList* (page 239)
- *wxColourMap* (page 81)

## 11.8. Events

See also *Overview* (page 390)

Some member functions that an application overrides are passed event objects containing information about the event.

- *wxCommandEvent* (page 88)
- *wxEvent* (page 149)
- *wxKeyEvent* (page 193)
- *wxMouseEvent* (page 214)

## 11.9. Data structures

These are the data structure classes offered by wxWindows.

- *wxDate* (page 101)
- *wxHashTable* (page 185)
- *wxList* (page 197)
- *wxNode* (page 221)
- *wxObject* (page 221)
- *wxString* (page 286)
- *wxStringList* (page 297)

## 11.10. Run-time class information system

See also *Overview* (page 370)

wxWindows supports run-time manipulation of class information, and dynamic creation of objects given class names.

- *wxClassInfo* (page 72)
- *wxObject* (page 221)
- *Macros* (page 352)

## 11.11. Debugging features

See also *Overview* (page 397)

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

- *wxDebugContext* (page 120)
- *wxDebugStreamBuf* (page 123)
- *wxObject* (page 221)
- *wxTrace* (page 351)
- *wxTraceLevel* (page 351)
- *WXDEBUG\_NEW* (page 352)
- *WXTRACE* (page 355)
- *WXTRACELEVEL* (page 355)

### 11.12. Interprocess communication

See also *Overview* (page 378)

*wxWindows* provides a simple interprocess communications facilities based on DDE.

- *wxClient* (page 74)
- *wxConnection* (page 91)
- *wxHelpInstance* (page 187)
- *wxServer* (page 277)

### 11.13. Document/view framework

See also *Overview* (page 372)

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

- *wxDocument* (page 140)
- *wxView* (page 315)
- *wxDocTemplate* (page 135)
- *wxDocManager* (page 128)
- *wxDocChildFrame* (page 126)
- *wxDocParentFrame* (page 134)
- *wxTransferFileToStream* (page 332)
- *wxTransferStreamToFile* (page 332)

### 11.14. Printing framework

See also *Overview* (page 377)

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

- *wxPreviewFrame* (page 244)
- *wxPreviewCanvas* (page 241)
- *wxPreviewControlBar* (page 241)
- *wxPageSetupData* (page 223)
- *wxPageSetupDialog* (page 227)
- *wxPrintData* (page 245)
- *wxPrintDialog* (page 248)
- *wxPrinter* (page 249)
- *wxPrinterDC* (page 250)

- *wxPrintout* (page 251)
- *wxPrintPreview* (page 253)

### 11.15. Database classes

See also *Database classes overview* (page 393)

*wxWindows* provides a set of classes for accessing Microsoft's ODBC (Open Database Connectivity) product.

- *wxDatabase* (page 96)
- *wxQueryCol* (page 256)
- *wxQueryField* (page 259)
- *wxRecordSet* (page 264)

### 11.16. Miscellaneous

- *wxApp* (page 44)
- *wxForm* (page 166)
- *wxIntPoint* (page 191)
- *wxPathList* (page 235)
- *wxPoint* (page 240)
- *wxTimer* (page 307)
- *wxTypeTree* (page 313)

### 11.17. wxString member functions

See also *Overview* (page 399)

This section describes categories of *wxString* (page 286) class member functions.

#### 11.17.1. Assignment

- *wxString::operator =* (page 294)

#### 11.17.2. Classification

- *wxString::IsAscii* (page 291)
- *wxString::IsWord* (page 292)
- *wxString::IsNumber* (page 292)
- *wxString::IsNull* (page 292)
- *wxString::IsDefined* (page 291)

#### 11.17.3. Comparisons (case sensitive and insensitive)

- *wxString::CompareTo* (page 289)
- *Compare* (page 296)
- *FCompare* (page 296)
- *Comparisons* (page 296)

#### 11.17.4. Composition and Concatenation

- *wxString::operator +=* (page 295)
- *wxString::Append* (page 287)
- *wxString::Prepend* (page 293)
- *wxString::Cat* (page 288)
- *operator +* (page 297)

#### 11.17.5. Constructors/Destructors

- *wxString::wxString* (page 286)
- *wxString:: wxString* (page 286)

#### 11.17.6. Conversions

- *wxString::operator const char \** (page 295)
- *wxString::Chars* (page 288)
- *wxString::GetData* (page 291)

#### 11.17.7. Deletion/Insertion

- *wxString::Del* (page 289)
- *wxString::Remove* (page 293)
- *wxString::Insert* (page 291)
- *Split* (page 297)
- *Join* (page 297)

#### 11.17.8. Duplication

- *wxString::Copy* (page 289)
- *wxString::Replicate* (page 293)

#### 11.17.9. Element access

- *wxString::operator[]* (page 295)
- *wxString::operator()* (page 295)
- *wxString::Elem* (page 290)
- *wxString::Firstchar* (page 290)
- *wxString::Lastchar* (page 292)

#### 11.17.10. Extraction of Substrings

- *wxString::At* (page 287)

- *wxString::Before* (page 287)
- *wxString::Through* (page 294)
- *wxString::From* (page 290)
- *wxString::After* (page 287)
- *wxString::SubString* (page 294)

#### **11.17.11. Input/Output**

- *wxString::sprintf* (page 294)
- *wxString::operator <<* (page 295)
- *wxString::operator >>* (page 295)
- *wxString::Readline* (page 293)

#### **11.17.12. Searching/Matching**

- *wxString::Index* (page 291)
- *wxString::Contains* (page 289)
- *wxString::Matches* (page 292)
- *wxString::Freq* (page 290)
- *wxString::First* (page 290)
- *wxString::Last* (page 292)

#### **11.17.13. Substitution**

- *wxString::GSub* (page 291)
- *wxString::Replace* (page 293)

#### **11.17.14. Status**

- *wxString::Length* (page 292)
- *wxString::Empty* (page 290)
- *wxString::Allocation* (page 286)
- *wxString::IsNull* (page 292)

#### **11.17.15. Transformations**

- *wxString::Reverse* (page 293)
- *wxString::Uppcase* (page 294)
- *wxString::Uppercase* (page 294)
- *wxString::Downcase* (page 290)
- *wxString::Lowercase* (page 292)
- *wxString::Capitalize* (page 287)

#### **11.17.16. Utilities**

- *wxString::Strip* (page 294)
- *wxString::Error* (page 290)

- *wxString::OK* (page 293)
- *wxString::Alloc* (page 286)
- *wxCHARARG* (page 295)
- *CommonPrefix* (page 296)
- *CommonSuffix* (page 296)

## 12. Topic overviews

This chapter contains a selection of topic overviews.

### 12.1. 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 | wxMINIMIZE_BOX | wxTHICK_FRAME
```

#### 12.1.1. wxFrame styles

The following styles apply to wxFrame windows.

**wxICONIZE** Display the frame iconized (minimized) (Windows only).  
**wxCAPTION** Puts a caption on the frame (Windows and XView only).  
**wxDEFAULT\_FRAME** Defined as **wxMINIMIZE\_BOX** | **wxMAXIMIZE\_BOX** | **wxTHICK\_FRAME** | **wxSYSTEM\_MENU** | **wxCAPTION**.  
**wxMDI\_CHILD** Specifies a Windows MDI (multiple document interface) child frame.  
**wxMDI\_PARENT** Specifies a Windows MDI (multiple document interface) parent frame.  
**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).  
**wxSDI** Specifies a normal SDI (single document interface) 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).  
**wxRESIZE\_BORDER** Displays a resizable border around the window (Motif only).  
**wxTINY\_CAPTION\_HORIZ** Under Windows 3.1, displays a small horizontal caption if **USE\_ITSY\_BITS** is set to 1 in **wx\_setup.h** and the Microsoft ItsyBitsy library has been compiled. Use instead of **wxCAPTION**.  
**wxTINY\_CAPTION\_VERT** Under Windows 3.1, displays a small vertical caption if **USE\_ITSY\_BITS** is set to 1 in **wx\_setup.h** and the Microsoft ItsyBitsy library has been compiled. Use instead of **wxCAPTION**.

#### 12.1.2. wxDialogBox styles

The following styles apply to wxDialogBox windows.

**wxCAPTION** Puts a caption on the dialog box (Motif only).  
**wxDEFAULT\_DIALOG\_STYLE** Equivalent to **wxCAPTION** | **wxSYSTEM\_MENU** | **wxTHICK\_FRAME**.  
**wxRESIZE\_BORDER** Display a resizable frame around the window (Motif only).  
**wxSYSTEM\_MENU** Display a system menu (Motif only).  
**wxTHICK\_FRAME** Display a thick frame around the window (Motif only).  
**wxUSER\_COLOURS** Under Windows, overrides standard control processing to allow setting of the dialog box background colour.  
**wxVSCROLL** Give the dialog box a vertical scrollbar (XView only).



### 12.1.3. wxItem styles

The following styles apply to all *wxItem* (page 191) derived windows.

**wxHORIZONTAL\_LABEL**      The item will be created with a horizontal label.  
**wxVERTICAL\_LABEL**      The item will be created with a vertical label.  
**wxFIXED\_LENGTH**      Allows the values of a column of items to be left-aligned. Create an item with this style, and pad out your labels with spaces to the same length. The item labels will initially be created with a string of identical characters, positioning all the values at the same x-position. Then the real label is restored.

### 12.1.4. wxButton styles

There are no styles specific to *wxButton* (page 54).

### 12.1.5. wxComboBox

The following styles apply to *wxComboBox* (page 82) items.

**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 (Windows only).

### 12.1.6. wxGauge styles

The following styles apply to *wxGauge* (page 180) items.

**wxGA\_HORIZONTAL**      The item will be created as a horizontal gauge.  
**wxGA\_VERTICAL**      The item will be created as a vertical gauge.  
**wxGA\_PROGRESSBAR**      Under Windows 95, the item will be created as a horizontal progress bar.

### 12.1.7. wxGroupBox styles

There are no styles specific to *wxGroupBox* (page 182).

### 12.1.8. wxListBox styles

The following styles apply to *wxListBox* (page 201) items.

**wxNEEDED\_SB**      Create scrollbars if needed.  
**wxLB\_NEEDED\_SB**      Same as **wxNEEDED\_SB**.  
**wxALWAYS\_SB**      Create scrollbars immediately.

`wxB_ALWAYS_SB` Same as `wxB_ALWAYS_LB`.  
`wxB_SINGLE` Single-selection list.  
`wxB_MULTIPLE` Multiple-selection list.  
`wxB_EXTENDED` Extended-selection list (Motif only).  
`wxHSCROLL` Create horizontal scrollbar if contents are too wide (Windows only).

### 12.1.9. `wxMessage` styles

There are no styles specific to `wxMessage` (page 211).

### 12.1.10. `wxRadioBox`

The following styles apply to `wxRadioBox` (page 260) items.

`wxVERTICAL` Lays the radiobox out in columns.  
`wxHORIZONTAL` Lays the radiobox out in rows.

### 12.1.11. `wxRadioButton`

The following styles apply to `wxRadioButton` (page 263) items.

`wxRB_GROUP` Specifies the start or end of a group of radio buttons under MS Windows.

### 12.1.12. `wxSlider` styles

The following styles apply to `wxSlider` (page 278) items.

`wxHORIZONTAL` The item will be created as a horizontal slider.  
`wxVERTICAL` The item will be created as a vertical slider.

### 12.1.13. `wxText`/`wxMultiText` styles

The following styles apply to `wxText` (page 299) and `wxMultiText` (page 219) items.

`wxTE_PROCESS_ENTER` The callback function will receive the event `wxEVENT_TYPE_TEXT_ENTER_COMMAND`. Note that this will break tab traversal for this panel item under Windows. Single-line text only.  
`wxTE_PASSWORD` The text will be echoed as asterisks. Single-line text only.  
`wxTE_READONLY` The text will not be user-editable.  
`wxHSCROLL` A horizontal scrollbar will be displayed. If `wxHSCROLL` is omitted, only a vertical scrollbar is displayed, and lines will be wrapped. This parameter is ignored under XView. Multi-line text only.

### 12.1.14. `wxTextWindow` styles

The following styles apply to *wxTextWindow* (page 302) objects.

- `wxBORDER`      Use this style to draw a thin border in Windows 3 (non-native implementation only).
- `wxNATIVE_IMPL`      Use this style to allow editing under Windows 3.1, albeit with a 64K limitation.
- `wxREADONLY`      Use this style to disable editing.
- `wxHSCROLL`      Use this style to enable a horizontal scrollbar, or leave it out to allow line wrapping. Windows and Motif only.

### 12.1.15. wxPanel styles

The following styles apply to *wxPanel* (page 228) windows.

- `wxBORDER`      Draws a thin border around the panel.
- `wxUSER_COLOURS`      Under Windows, overrides standard control processing to allow setting of the panel background colour.
- `wxVSCROLL`      Gives the dialog box a vertical scrollbar (XView only).

### 12.1.16. wxCanvas styles

The following styles apply to *wxCanvas* (page 57) windows.

- `wxBORDER`      Gives the canvas a thin border (Windows 3 and Motif only).
- `wxRETAINED`      Gives the canvas a *wxWindows*-implemented backing store, making repainting much faster but at a potentially costly memory premium (XView and Motif only).

### 12.1.17. wxToolBar styles

The following styles apply to *wxToolBar* (page 308) objects.

- `wxTB_3DBUTTONS`      Gives a 3D look to the buttons, but not to the same extent as `wxButtonBar`.

## 12.2. Run time class information overview

Classes: *wxObject* (page 221), *wxClassInfo* (page 72).

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* (from version 1.62) 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.

Variations on these *macros* (page 352) 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` (page 343) 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`.

### 12.2.1. wxClassInfo

See also *Run time class information overview* (page 370)

Class: `wxClassInfo` (page 72)

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* (page 221) and *wxCreateDynamicObject* (page 343).

### 12.2.2. 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(wxFFrame, wxWindow)

wxFFrame::wxFFrame(void)
{
    ...
}
```

### 12.3. Document/view overview

Classes: *wxDocument* (page 140), *wxView* (page 315), *wxDocTemplate* (page 135), *wxDocManager* (page 128), *wxDocParentFrame* (page 134), *wxDocChildFrame* (page 126), *wxCommand* (page 86), *wxCommandProcessor* (page 89)

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 canvas) 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` (page 251) 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 (page 377) in your menus.

### 12.3.1. wxDocument overview

See also *Document/view framework overview* (page 372)

Class: `wxDocument` (page 140)

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` (page 315), `wxDocTemplate` (page 135) and `wxDocManager` (page 128) 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` (page 135) 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.

### 12.3.2. `wxView` overview

See also *Document/view framework overview* (page 372)

Class: `wxView` (page 315)

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` (page 140), `wxDocTemplate` (page 135) and `wxDocManager` (page 128) 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` (page 135) 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.

### 12.3.3. `wxDocTemplate` overview

See also *Document/view framework overview* (page 372)

Class: `wxDocTemplate` (page 135)

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` (page 315), `wxDocument` (page 140) and `wxDocManager` (page 128) 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`.

#### 12.3.4. `wxDocManager` overview

See also *Document/view framework overview* (page 372)

Class: `wxDocManager` (page 128)

The `wxDocManager` class is part of the document/view framework supported by `wxWindows`, and cooperates with the `wxView` (page 315), `wxDocument` (page 140) and `wxDocTemplate` (page 135) 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`.

#### 12.3.5. `wxCommand` overview

See also *Document/view framework overview* (page 372)

Classes: `wxCommand` (page 86), `wxCommandProcessor` (page 89)

`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* (page 376) 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.

### 12.3.6. *wxCommandProcessor* overview

See also *Document/view framework overview* (page 372)

Classes: *wxCommandProcessor* (page 89), *wxCommand* (page 86)

*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.

### 12.3.7. *wxFileHistory* overview

See also *Document/view framework overview* (page 372)

Classes: *wxFileHistory* (page 156), *wxDocManager* (page 128)

*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 in your *wxFrame::OnMenuCommand*. Below is the code used by the default document/view parent frame.

```
void wxDocParentFrame::OnMenuCommand(int id)
{
    switch (id)
    {
        case wxID_EXIT:
        {
            if (GetEventHandler()->OnClose())
                delete this;
            break;
        }
        case wxID_FILE1:
        case wxID_FILE2:
        case wxID_FILE3:
        case wxID_FILE4:
        case wxID_FILE5:
```

```
case wxID_FILE6:
case wxID_FILE7:
case wxID_FILE8:
case wxID_FILE9:
{
    char *f = docManager->GetHistoryFile(id-wxID_FILE1);
    if (f)
        (void)docManager->CreateDocument(f, wxDOC_SILENT);
    break;
}
default:
{
    docManager->OnMenuCommand(id);
}
}
}
```

### 12.3.8. 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)

## 12.4. Printing overview

Classes: `wxPrintout` (page 251), `wxPrinter` (page 249), `wxPrintPreview` (page 253), `wxPrinterDC` (page 250), `wxPrintDialog` (page 248).

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* (page 372) 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 WXPRIINT_PRINT:
    {
        wxPrinter printer;
        MyPrintout printout("My printout");
        printer.Print(this, &printout, TRUE);
        break;
    }
    case WXPRIINT_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 WXPRIINT_PRINT_SETUP:
    {
        wxPrintDialog printerDialog(this);
        printerDialog.GetPrintData().SetSetupDialog(TRUE);
        printerDialog.Show(TRUE);
        break;
    }
}

```

## 12.5. Interprocess communication overview

Classes: `wxServer` (page 277), `wxConnection` (page 91), `wxClient` (page 74).

The following describes how `wxWindows` implements DDE. The following three classes are central.

1. `wxClient`. This represents the client application, and is used only within a client program.
2. `wxServer`. This represents the server application, and is used only within a server program.
3. `wxConnection`. 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 `wxServer`.
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 `wxConnection`.
5. Provide handlers for various messages that are sent to the server side of a `wxConnection`.

To create a working client, the programmer must:

1. Derive a class from `wxClient`.
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 `wxConnection`.
5. Provide handlers for various messages that are sent to the client side of a `wxConnection`.
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 `wxConnection` member functions to send messages to the server.

### 12.5.1. 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.

### 12.5.2. 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.

See also the source for *wxHelp*, which is a DDE server, and the files *wx\_help.h* and *wx\_help.cc* which implement the client interface to *wxHelp*.

### 12.5.3. Remote Procedure Call

DDE is quite a low level protocol, and all encoding and decoding of messages must be done by the client and server applications. The *wxWindows* extension *PrologIO* implements a remote procedure call protocol (RPC) so that a server can implement a library of functions for a client to call. *PrologIO* makes it easy for applications to pack and unpack the arguments and return value(s) of procedure calls, and provides a mechanism for the server to register its available calls and automatically handle the routing of calls to appropriate server callbacks, one to a procedure definition. All this makes calling or implementing server facilities childishly simple. Since *PrologIO* sits on top of the DDE wrapper, it is also platform independent. See the separate *PrologIO* manual and *PrologIO* (page 29).

### 12.5.4. More DDE details

A *wxClient* 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 *wxConnection* and another from *wxClient*. The custom *wxConnection* class will intercept communications in a 'conversation' with a server, and the custom *wxServer* is required so that a user-overridden *wxClient::OnMakeConnection* (page 74) member can return a *wxConnection* of the required class, when a connection is made.

For example:

```
class MyConnection: public wxConnection
{
public:
    MyConnection(void)::wxConnection(ipc_buffer, 3999) {}
    ~MyConnection(void) {}
    Bool OnAdvise(char *topic, char *item, char *data, int size, int
format)
    { wxMessageBox(topic, data); }
};

class MyClient: public wxClient
{
public:
    MyClient(void) {}
    wxConnection *OnMakeConnection(void) { return new MyConnection; }
};
```

Here, **MyConnection** will respond to *OnAdvise* (page 92) messages sent by the server.

When the client application starts, it must first call *wxIPCInitialize* (page 348) before creating an instance of the derived *wxClient*. 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 *wxClient::MakeConnection* (page 74) 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.

```
wxIPCInitialize();

char *server = "4242";
char hostName[256];
wxGetHostName(hostName, sizeof(hostName));

char *host = hostName;

if (argc > 1)
    server = argv[1];
if (argc > 2)
    host = argv[2];

// Create a new client
MyClient *client = new MyClient;
the_connection = (MyConnection *)client->MakeConnection(host, server,
"IPC TEST");

if (!the_connection)
{
    wxMessageBox("Failed to make connection to server", "Client Demo
Error");
    return NULL;
}
the_connection->StartAdvise("Item");
```

## 12.6. Font overview

Class: *wxFont* (page 158)

A font is an object which determines the appearance of text, primarily when drawing text to a canvas 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: <b>wxDEFAULT</b> , <b>wxDECORATIVE</b> , <b>wxROMAN</b> , <b>wxSCRIPT</b> , <b>wxSWISS</b> , <b>wxMODERN</b> . <b>wxMODERN</b> is a fixed pitch font; the others are either fixed or variable pitch.
Style	The value can be <b>wxNORMAL</b> , <b>wxSLANT</b> or <b>wxITALIC</b> .
Weight	The value can be <b>wxNORMAL</b> , <b>wxLIGHT</b> or <b>wxBOLD</b> .
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.

Under X, the situation is more complicated because X does not support a simple naming scheme that will allow consistent naming of screen and printer fonts. To address this, wxWindows implements a *font name directory* (page 382) with a naming convention to support screen and PostScript fonts. Under this scheme, the 'family' parameter can also be used as a font identifier. However, if you wish, you may still use the family parameter in exactly the same way as before without needing to understand the enhanced usage.

**Note:** There is currently a difference between the appearance of fonts on the two platforms, if the mapping mode is anything other than MM\_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 canvas will also scale fonts under both environments.

### 12.6.1. Font name directory overview

Class: *wxFontNameDirectory* (page 164)

The font name directory helps implement the portable font scheme used in the X versions of wxWindows, and optionally under Windows.

To draw text, you need a font id, weight, style, size, and underline flag. The combination font id x weight x style maps to a "real" platform-specific font.

Every font id is associated to one of a fixed number of family ids. Each of these family ids can be used as a font id, specifying a default font for that family. When font information is stored in a document that may cross platforms, the family id should be specified so that a reasonable default font can be selected on the new platform.

Each font id is associated to a name (corresponding to the "facename" in Windows). This name should be used to store information about a font on disk, since the font id used for a particular font name can change when a wxWindows application is restarted.

The font constructor:

```
wxFont(int pointSize, int familyOrFontId, int style, int weight,
       Bool underline = FALSE, char *faceName = NULL);
```

has changed if the new scheme is in operation. When faceName is NULL, familyOrFontId (formerly familyId) can be any font id. Recall that a family id can always be used as a font id. When faceName is not NULL, then familyOrFontId must be a family id. If the specified faceName cannot be found, then a default font for the family id (passed as familyOrFontId) will be used.

The mappings:

- font id, weight, style to real screen/PostScript font
- font id to font name
- font id to family
- font name to font id

are managed by a single instance of *wxFontNameDirectory*.

## 12.7. Device context overview

Classes: *wxDC* (page 108), *wxPostScriptDC* (page 240), *wxMetaFileDC* (page 213), *wxMemoryDC* (page 205), *wxPrinterDC* (page 250), *wxScreenDC* (page 275).

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 canvas has a device context 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.

To determine whether a device context is colour or monochrome, test the **Colour** Bool member variable. To override *wxWindows* monochrome graphics drawing behaviour, set this member to **TRUE**.

*wxDC* is abstract and cannot be used to create device context objects. Instead, use a derived class. *wxCanvasDC* (page 68) is a context that cannot be created by the user but can be retrieved from a *wxCanvas* (page 57) by using *wxCanvas::GetDC* (page 61).

When writing code to draw into a device context, use **wxDC** as a parameter whenever possible, to allow the most general use of your drawing code. You can then pass a device context object of any derived type. See the demo in `samples/hello` for code which uses this device-independent method of drawing.

## 12.8. wxApp overview

Classes: *wxApp* (page 44)

A *wxWindows* application does not have a *main* procedure; the equivalent is the *OnInit* (page 46) member defined for a class derived from *wxApp*. *OnInit* must create and return a main window frame as a bare minimum. If **NULL** is returned from *OnInit*, the application will exit. 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 *wxFrame::OnClose* (page 176) member.

In emergencies the *wxExit* (page 345) function can be called to kill the application.

An example of defining an application follows:

```
class DerivedApp: public wxApp
{
public:
    wxFrame *OnInit(void);
};

wxFrame *DerivedApp::OnInit(void)
{
    wxFrame *the_frame = new wxFrame(NULL, argv[0]);
    ...
    return the_frame;
}
```



```
MyApp DerivedApp;
```

## 12.9. Bitmaps overview

Classes: *wxBitmap* (page 48), *wxIcon* (page 183), *wxCursor* (page 94).

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 probably be required.

A bitmap created dynamically or loaded from a file can be selected into a memory device context (instance of *wxMemoryDC* (page 205)). This enables the bitmap to be copied to a canvas or memory device context using *wxDC::Blit* (page 108), or to be used as a drawing surface. The **wxToolBar** class was 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 canvas.

*wxWindows* contains code to 'grey out' a bitmap when used in an insensitive panel item. Under X, this code is contained in the *wxBitmap* class. Under Windows, the user-contributed Fafa library is responsible for this.

See *wxMemoryDC* (page 205) for an example of drawing onto a bitmap.

The following shows the conditional compilation required to load a bitmap in X and in Windows 3. The alternative is to use the string version of the bitmap constructor, which loads a file under X and a resource under Windows 3, but has the disadvantage of requiring the X icon file to be available at run-time.

```
#ifdef wx_x
#include "aiai.xbm"
#endif
#ifdef wx_msw
    wxIcon *icon = new wxBitmap("aiai");
#endif
#ifdef wx_x
    wxIcon *icon = new wxBitmap(aiai_bits, aiai_width, aiai_height);
#endif
```

### 12.9.1. Loading bitmaps: further information

See also the DIB and *wxImage* libraries distributed with *wxWindows*. DIB allows loading of .BMP files under Windows, and *wxImage* allows loading of a variety of bitmap formats under X. *wxBuilder* makes use of both of these packages: search for *wxLoadBitmap* for example of usage.

There is now (from version 1.61) extra provision for a number of bitmap formats via the standard *wxBitmap* class. These extra facilities can be enabled using settings in *wx\_setup.h*; by default they are switched off.

XPM colour pixmaps may be loaded and saved under Windows and X, with some restrictions imposed by the lack of colourmap facility when using XPM files. The user may elect to use XPM files as a cross-platform standard, or translate between XPM and BMP files using a suitable utility (one is under preparation for *wxWindows* users).

Also, under Windows, DIBs (device independent bitmaps with extension BMP) may be dynamically loaded and saved. Under X, GIF and BMP files may be loaded but not saved.

### 12.10. Dialog box overview

Classes: *wxDialogBox* (page 123), *wxEnhDialogBox* (page 146)

A dialog box is similar to a panel, in that it is a window which can be used for placing panel items, 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 XView, some panel items may display incorrectly in a modal dialog, and two modal dialogs may not be open simultaneously. This can be corrected using a patch (see *install/install.txt* and *install/xview.txt*).

Under implementations that permit it, *wxDialogBox* inherits from *wxCanvas* via *wxPanel*, and has a *wxPanelDC* that the application can draw on.

The panel device context associated with *wxDialogBox* behaves slightly differently than for a panel or canvas: drawing to it *requires* enclosing code in *BeginDrawing*, *EndDrawing* calls. This is because under Windows, dialog box device contexts are not 'retained' and settings would be lost if the device context were retrieved and released for each drawing operations.

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* (page 333).

See also *wxPanel* (page 228) and *wxWindow* (page 319) for inherited member functions.

### 12.11. Common dialogs overview

Classes and functions: *wxColourDialog* (page 80), *wxFontDialog* (page 162), *wxPrintDialog* (page 248), *dialog functions* (page 333)

Common dialog classes and functions encapsulate commonly-needed dialog box requirements. They are mostly '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.

Not all common dialogs have classes; some are still in functional form, awaiting an object-oriented make-over, such as the message box and file selector dialogs. A few familiar MS Windows-style common dialogs have yet to be implemented, such as the text search dialog and a directory selector.

### 12.11.1. wxColourDialog overview

Classes: *wxColourDialog* (page 80), *wxColourData* (page 78)

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*.

*wxColourDialog* is available under Motif and Windows. Under XView there seem to be some problems, probably related to modal dialogs.

#### 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 canvas.

```
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.Show(TRUE))
{
    wxColourData retData = dialog.GetColourData();
    wxColour col = retData.GetColour();
    wxBrush *brush = wxTheBrushList->FindOrCreateBrush(&col, wxSOLID);
```

```
myCanvas->SetBackground(brush);  
myCanvas->Clear();  
myCanvas->Refresh();  
}
```

### 12.11.2. wxFontDialog overview

Classes: *wxFontDialog* (page 162), *wxFontData* (page 160)

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, strikeout 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, strikeout 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.

*wxFontDialog* is available under Motif and Windows. Under XView there seem to be some problems, probably related to modal dialogs.

#### 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.Show(TRUE))  
{  
    wxFontData retData = dialog.GetFontData();  
    canvasFont = retData.GetChosenFont();  
    (*canvasTextColour) = retData.GetColour();  
    myCanvas->Refresh();  
}
```

### 12.11.3. wxPrintDialog overview

Classes: *wxPrintDialog* (page 248), *wxPrintData* (page 245)

This class represents the print and print setup common dialogs. You may obtain a *wxPrinterDC* (page 250) device context from a successfully dismissed print dialog.

The samples/printing example shows how to use it: see *Printing overview* (page 377) for an excerpt from this example.

## 12.12. Constraints overview

Classes: *wxLayoutConstraints* (page 196), *wxIndividualLayoutConstraint* (page 189).

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* (page 323) 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.

### 12.12.1. 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.

## 12.12.2. Window layout examples

### 12.12.2.1. Example 1: subwindow layout

This example specifies a panel and a canvas side by side, with a text subwindow below it.

```
frame->panel = new wxPanel(frame, 0, 0, 1000, 500, 0);
frame->canvas = new MyCanvas(frame, 0, 0, 400, 400, wxRETAINED);
frame->text_window = new MyTextWindow(frame, 0, 250, 400, 250,
wxNATIVE_IMPL);

// 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 canvas 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->canvas->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);
```

### 12.12.2.2. 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, (wxFunction)NULL, "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, (wxFunction)NULL, "A
list",
                                wxSINGLE, -1, -1, 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);

wxMultiText *mtext = new wxMultiText(frame->panel, (wxFunction)NULL,
"Multiline text", "Some text",
                                -1, -1, 150, 100);

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);
```

## 12.13. Event handling overview

Classes: *wxEvtHandler* (page 150), *wxWindow* (page 319)

To handle events that are generated by the windowing system (usually in response to a user's action), the programmer must derive a class from the window that is sent the event, and define appropriate behaviour. For example, to respond to *OnPaint* messages from a *wxCanvas*, you would derive a new class *MyCanvas* and define the function *MyCanvas::OnPaint*. Within this function, you paint the canvas as necessary for your application.

To respond to a user closing a window, define `OnClose`; to intercept character input from a canvas, define `OnChar`, and so on. These events are listed in the class description for `wxEvtHandler` (page 150), which is the base class for all window classes.

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, overriding the appropriate member function, and then call `wxWindow::SetEventHandler` (page 327) 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`.

You could use this technique to respond to panel item commands. Normally, you supply a function of type `wxFunction` to the panel item constructor, and it will be called with references to the panel item and command event. Instead, you could use a single `wxEvtHandler` object for handling one or more panel items, setting the event handler object for the panel item just after item construction. This handler could even be the panel or dialog box.

Another use of `SetEventHandler` 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 `Set/GetNextHandler` and `Set/GetPreviousHandler` to form a chain of event handlers, taking care to pass events along the chain.

## 12.14. Toolbar overview

Classes: `wxToolBar` (page 308), `wxButtonBar` (page 55)

The `wxToolBar` class gives `wxWindows` programs an extra, and increasingly popular, user interface component: 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. Besides which, a toolbar looks prettier than a purely menu-based interface.

`wxToolBar` uses a canvas subwindow for drawing bitmaps, and so bitmap images cannot be mixed with panel items, but in most cases this won't be important. 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).

Mouse click events for a given button are sent to a member called **`OnLeftClick`**, and so an application must derive from `wxToolBar` in order to use it. The application can also handle **`OnMouseEnter`** events for the tools, to give the user extra feedback about the tools as the mouse moves over them.



This toolbar class does not give as slick an appearance as, or the responsiveness of, conventional Windows toolbars. The buttons are not given a 3D appearance and do not depress like normal buttons. However, you can use the optimized `wxButtonBar` library for greatly improved Windows behaviour, and behaviour under X identical to `wxToolBar`. See *The wxButtonBar library* (page 392).

### 12.14.1. Using the toolbar library

Include the file `wx_tbar.h` to use this class.

An example of toolbar use is given in the sample program contained in `test.cc` and `test.h`. This creates a main window, and two toolbars: a floating toolbar with 24 tools, and a toolbar along the top of the main drawing canvas, divided into groups. The icons for this second toolbar would normally be quite small.

The test program defines a general-purpose derived frame called **`wxFrameWithToolBar`** which can manage a frame with one main subwindow and one horizontal toolbar.

Note that one of the bitmaps on the floating toolbar is a small version of the main graphic: this demonstrates how a memory device context can be used to draw into a bitmap. An application which allowed the user to build up a symbol library dynamically might create this kind of bitmap.

Left clicks and movements over the toolbars are intercepted and information is displayed on the status line.

The following fragment illustrates the essence of creating a toolbar.

```
toolBarBitmaps[0] = new wxBitmap("icon1");
toolBarBitmaps[1] = new wxBitmap("icon2");
toolBarBitmaps[2] = new wxBitmap("icon3");
...

toolBarFrame = new wxFrame(NULL, "Tools", 0, 0, 300, 200,
    wxSDI | wxDEFAULT_FRAME | wxSTAY_ON_TOP);

// 5 rows
toolBar = new TestToolBar(toolBarFrame, 10, 10, -1, -1, 0,
wxVERTICAL, 5);
toolBar->SetMargins(2, 2);
toolBar->GetDC()->SetBackground(wxGREY_BRUSH);

for (int i = 10; i < 25; i++)
    toolBar->AddTool(i, toolBarBitmaps[i], NULL, TRUE);

toolBar->Layout();
float maxWidth, maxHeight;
toolBar->GetMaxSize(&maxWidth, &maxHeight);
toolBarFrame->SetClientSize((int)maxWidth, (int)maxHeight);
toolBarFrame->Show(TRUE);
```

### 12.14.2. The `wxButtonBar` library

See also *wxToolBar* overview (page 391)

Class: *wxButtonBar* (page 55)

*wxToolBar* does the job, but it isn't as slick as it could be. The *wxButtonBar* library class presents an almost identical Application Programming Interface, but under Windows, the buttons are 3D and depress properly.

Under Windows, it expects 16-colour bitmaps that are 16 pixels wide and 15 pixels high. If you want to use a different size, call ***wxButtonBar::SetDefaultSize*** as the demo shows, before adding tools to the button bar. Don't supply more than one bitmap for each tool, because *wxButtonBar* generates all three images (normal, depressed and checked) from the single bitmap you give it.

Include the file `wx_bbar.h` to use this class.

X-optimized (or generic) button bar code may follow at a future date.

### 12.14.2.1. Windows 95 differences

Under Windows 95, *wxButtonBar* behaves slightly differently than under generic WIN32, since it uses the Windows 95 toolbar common control.

1. *CreateTools* must be called after the tools have been added.
2. No absolute positioning is supported but you can specify the number of rows, and add tool separators with *AddSeparator*. Layout does nothing.
3. Tooltips are supported.
4. *OnRightClick* is not supported.
5. The device context support is limited, though there is enough to support drawing a border from within *OnPaint*.
6. *OnEvent* and *OnChar* are not supported.
7. Scrollbars are not supported.

*Note:* under Windows 95, a *wxButtonBar* cannot be moved to any position other than the top-left of the frame. If this is a problem, you may wish to alter `wx_bbar.h` and `wx_bbar.cc` to compile the non-Windows 95 code instead.

## 12.15. Database classes overview

Classes: *wxDatabase* (page 96), *wxRecordSet* (page 264), *wxQueryCol* (page 256), *wxQueryField* (page 259)

**IMPORTANT NOTE:** The ODBC classes are a preliminary release and incomplete. Please take this into account when using them. Feedback and bug fixes are appreciated, as always. The classes are being developed by Olaf Klein ([oklein@smallo.ruhr.de](mailto:oklein@smallo.ruhr.de)) and Patrick Halke ([patrick@zaphod.ruhr.de](mailto:patrick@zaphod.ruhr.de)).

*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.

### 12.15.1. 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` (page 268)), then you do not need to call `wxDatabase::Open` (page 100). 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` (page 267), 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` (page 269) or `wxRecordSet::GetFieldDataPtr` (page 269) 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` (page 274), `wxRecordSet::MovePrev` (page 274) 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.

### 12.15.2. wxDatabase overview

See also *Database classes overview* (page 393)

Class: `wxDatabase` (page 96)

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).

### 12.15.3. `wxQueryCol` overview

See also *Database classes overview* (page 393)

Class: `wxQueryCol` (page 256)

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.

### 12.15.4. `wxQueryField` overview

See also *Database classes overview* (page 393)

Class: `wxQueryField` (page 259)

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.

### 12.15.5. `wxRecordSet` overview

See also *Database classes overview* (page 393)

Class: `wxRecordSet` (page 264)

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`.

### 12.15.6. ODBC SQL data types

See also *Database classes overview* (page 393)

These are the data types supported in ODBC SQL. Note that there are other, extended level conformance types, not currently supported in `wxWindows`.

`CHAR(n)`            A character string of fixed length *n*.  
`VARCHAR(n)`        A varying length character string of maximum length *n*.  
`LONG VARCHAR(n)`    A varying length character string: equivalent to `VARCHAR` for the purposes of ODBC.  
`DECIMAL(p, s)`    An exact numeric of precision *p* and scale *s*.

NUMERIC(p, s) 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.

### 12.15.7. A selection of SQL commands

See also *Database classes overview* (page 393)

The following is a very brief description of some common SQL commands, with examples.

#### 12.15.7.1. 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)
  )
```

#### 12.15.7.2. Insert

Inserts records into a table.

Example:

```
INSERT INTO Book
  (BookNumber, CategoryCode, Title)
VALUES(5, 'HR', 'The Lark Ascending')
```

### 12.15.7.3. 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.

### 12.15.7.4. 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'.

## 12.16. Debugging overview

Classes: *wxDebugContext* (page 120), *wxDebugStreamBuf* (page 123), *wxObject* (page 221)

**IMPORTANT NOTE:** The debugging facilities in wxWindows are new (June 1995) so please be careful when using them. Since they operate at a low level by redefining memory allocation operators, there may be unforeseen problems on specific platforms. Proceed with caution!

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 DEBUG flag set to 1 or more.

wxDebugContext is a class that never gets instantiated, but ties together various functions and variables. It allows you to set the debugging stream, dump all objects to that stream, write statistics about object allocation, and check memory for errors.

You can use the *WXTRACE* (page 355) macro to output debugging information in DEBUG mode; it will be defined to nothing for non-debugging code.

It is good practice to define a Dump member function for each class you derive from a wxWindows class, so that wxDebugContext::Dump can call it and give valuable information about the state of the application.

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 in theory 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 you have difficulty tracking down a memory leak, recompile in debugging mode and call wxDebugContext::Dump and wxDebugContext::Statistics at appropriate places. They will tell you what objects have not yet been deleted, and what kinds of object they are.

If you use the macro WXDEBUG\_NEW instead of the normal 'new', the debugging output (and error messages reporting memory problems) will also tell you what file and on what line you allocated the object.

To avoid the need for replacing existing new operators with WXDEBUG\_NEW, you can write this at the top of each application file:

```
#define new WXDEBUG\_NEW
```

In non-debugging mode, this will revert to the usual interpretation of new. Note that for this not to mess up new-based allocation of non-wxObject derived classes and built-in types, there are global definitions of new and delete which match the syntax required for storing filename and line numbers. These merely call malloc and free, and so do not do anything interesting. The definitions may possibly cause multiple symbol problems for some compilers and so might need to be omitted by setting the USE\_GLOBAL\_MEMORY\_OPERATORS to 0 in wx\_setup.h

### 12.16.1. wxDebugContext overview

See also *Debugging overview* (page 397)

Class: *wxDebugContext* (page 120)

wxDebugContext is a class for performing various debugging and memory tracing operations. wxDebugContext, and the related macros and function WXTRACE and wxTrace, are only present if USE\_DEBUG\_CONTEXT is used.

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. Unfortunately the define of `new` to `WXDEBUG_NEW` does not work for Borland C++ (and perhaps other compilers) because it fails to find the correct overloaded operator for non-object usage of `new`. Instead, you need to use `WXDEBUG_NEW` explicitly if there are any examples of non-object `new` usage in the file.

```
#define new WXDEBUG_NEW

wxDebugContext::SetCheckpoint();

wxDebugContext::SetFile("c:\\temp\\debug.log");

wxString *thing = new wxString;

// Proves that defining 'new' to be 'WXDEBUG_NEW' doesn't mess up
// non-object allocation. Doesn't work for Borland C++.
char *ordinaryNonObject = new char[1000];

wxDebugContext::Dump();
wxDebugContext::PrintStatistics();
```

You can use `wxDebugContext` if `DEBUG` is 1 or more, or you can use it at any other time (if `USE_DEBUG_CONTEXT` is 1). It is not disabled for `DEBUG = 1` (as in earlier versions of `wxWindows`) because you may not wish to recompile `wxWindows` and your entire application just to make use of the error logging facility. This is especially true in a Windows NT or Windows 95 environment, where you cannot easily output to a debug window: `wxDebugContext` can be used to write to log files instead.

## 12.17. wxString overview

Class: *wxString* (page 286)

Strings are used very frequently in most programs. There is no direct support in the C++ language for strings. A string class can be useful in many situations: it not only makes the code shorter and easier to read, it also provides more security, because we don't have to deal with pointer acrobatics.

`wxString` is available in two versions: a cut-down `wxWindows`, copyright-free version, and a much more powerful GNU-derived version. The default is the GNU-derived, fully-featured version, ported and revised by Stefan Hammes.

For backward compatibility most of the member functions of the original `wxWindows` `wxString` class have been included, except some 'dangerous' functions.

`wxString` can be compiled under MSW, UNIX and VMS (see below). The function names have been capitalized to be consistent with the `wxWindows` naming scheme.

The reasons for not using the GNU string class directly are:

- It is not available on all systems (generally speaking, it is available only on some UNIX systems).
- We can make changes and extensions to the string class as needed and are not forced to use 'only' the functionality of the GNU string class.



The GNU code comes with certain copyright restrictions. If you can't live with these, you will need to use the cut-down `wxString` class instead, by editing `wx_setup.h` and appropriate `wxWindows` makefiles.

### 12.17.1. Copyright of the original GNU code portion

Copyright (C) 1988, 1991, 1992 Free Software Foundation, Inc. written by Doug Lea  
(dl@rocky.oswego.edu)

This file is part of the GNU C++ Library. This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details. You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.

### 12.17.2. Features/Additions/Modifications

The `wxString` class offers many string handling functions and a support for regular expressions. This gives powerful, easy-to-use pattern-matching functionality. See below for a discussion of the GNU features of `wxString`. See also the header file `wxstrgnu.h` which shows all member functions.

As stated above, there are extensions to the `wxString` class. This includes the including of the 'old' `wxString` class member functions. Below is a list of the additional member functions:

- Access to the internal representation. Should be used with care:  
`char* GetData() const;`
- To make a copy of 'this' (only for compatibility):  
`wxString Copy() const;`
- For case sensitive and case insensitive comparisons:  
`enum caseCompare {exact, ignoreCase};`  
`int CompareTo(const char* cs, caseCompare cmp = exact)`  
`const;`  
`int CompareTo(const wxString& st, caseCompare cmp = exact)`  
`const;`
- For case sensitive and case insensitive containment check:  
`Bool Contains(const char* pat, caseCompare cmp = exact)`  
`const;`  
`Bool Contains(const wxString& pat, caseCompare cmp = exact)`  
`const;`
- For case sensitive and case insensitive index calculation:  
`int Index(const char* pat, int i=0, caseCompare cmp = exact)`

- ```
const;
    int Index(const wxString& s, int i=0, caseCompare cmp = exact)
const;
```
- For element access in addition to the [] operator:  
`char& operator()(int);` // Indexing with bounds checking
  - To put something in front of a string:  
`wxString& Prepend(const char*);` // Prepend a character string  
`wxString& Prepend(const wxString& s);`  
`wxString& Prepend(char c, int rep=1);` // Prepend c rep times
  - For concatenation:  
`wxString& Append(const char* cs);`  
`wxString& Append(const wxString& s);`  
`wxString& Append(char c, int rep=1);` // Append c rep times
  - To get the first and last occurrence of a char or string:  
`int First(char c) const;`  
`int First(const char* cs) const;`  
`int First(const wxString& cs) const;`  
`int Last(char c) const;`  
`int Last(const char* cs) const;`  
`int Last(const wxString& cs) const;`
  - To insert something into a string  
`wxString& Insert(int pos, const char*);`  
`wxString& Insert(int pos, const wxString&);`
  - To remove data (in addition to the 'Del' functions):  
`wxString& Remove(int pos);` // Remove pos to end of string  
`wxString& Remove(int pos, int n);` // Remove n chars starting at pos  
`wxString& RemoveLast(void);` // It removes the last char of a string
  - To replace data:  
`wxString& Replace(int pos, int n, const char*);`  
`wxString& Replace(int pos, int n, const wxString&);`
  - Alternative names for compatibility:  
`void LowerCase();` // Change self to lower-case  
`void UpperCase();` // Change self to upper-case
  - Edward Zimmermann's additions:

```
wxString SubString(int from, int to);
```

- Formatted assignment:

```
void sprintf(const char *fmt, ...);
```

We do not use the 'sprintf' constructor of the old wxString class anymore, because with that constructor, every initialisation with a string would go through sprintf and this is not desirable, because sprintf interprets some characters. With the above function we can write:

```
wxString msg; msg.sprintf("Processing item %d\n",count);
```

- Strip chars at the front and/or end. This can be useful for trimming strings:

```
enum StripType {leading = 0x1, trailing = 0x2, both = 0x3};
wxSubString Strip(StripType s=trailing, char c=' ');
```
- Line input: Besides the stream I/O functions this function can be used for non-standard formatted I/O with arbitrary line terminators.

```
friend int Readline(FILE *f, wxString& x,
                  char terminator = '\\n',
                  int discard_terminator = 1);
```

- The GNU wxString class lacks some classification functions:

```
int IsAscii() const;
int IsWord() const;
int IsNumber() const;
int IsNull() const;
int IsDefined() const;
```

- The meaning of nil has been changed. A wxString x is only nil, if it has been declared 'wxString x'. In all other cases it is NOT nil. This seems to me more logical than to let a 'wxString x=""' be nil as it was in the original GNU code.
- **IMPORTANT:**the following is a very, very, very ugly macro, but it makes things more transparent in cases, where a library function requires a (char\*) argument. This is especially the case in wxWindows, where most char-arguments are (char\*) and not (const char\*). this macro should only be used in such cases and NOT to modify the internal data. The standard type conversion function of wxString returns a '(const char\*)'. The conventional way would be 'function((char\*)string.Chars())'. With the macro this can be achieved by 'function(wxCHARARG(string))'. This makes it clearer that the usage should be confined to arguments. See below for examples.

```
#define wxCHARARG(s) ((char*)(s).Chars())
```

### 12.17.3. Function calls

When using `wxString` objects as parameters to other functions you should note the following:

```
void f1(const char *s){}
void f2(char *s){}

main(){
    wxString aString;
    f1(aString); // ok
    f2(aString); // error
    f2(wxCHARARG(aString)); // ok
    printf("%s",aString); // NO compilation error, but a runtime error.
    printf("%s",aString.Chars()); // ok
    printf("%s",wxCHARARG(aString)); // ok
}
```

#### 12.17.4. Header files

For DOS and UNIX we use a stub-headerfile `include/base/wxstring.h` which includes the two headerfiles in the `contrib/wxstring` directory, namely `contrib/wxstring/wxstrgnu.h` and `contrib/wxstring/wxregex.h`. If there is a headerfile `contrib/wxstring/wxstring.h`, please delete it. It will cause problems in the VMS compilation.

For VMS we have to do an addition due to the not very intelligent inclusion mechanism of the VMS C++ compiler: In the VMS-Makefile, the include-file search path is augmented with the `contrib/wxstring` directory, so that the correct headerfiles can be included.

So you have only to specify

```
#define USE_GNU_WXSTRING 1
```

in `include/base/wx_setup.h` to use the `wxString` class.

#### 12.17.5. Test program

Stefan Hammes has included a test program `test.cc` in the `contrib/wxstring` directory for many features of `wxString` and `wxRegex`. It also tests Stefan's extensions. When running the compiled program, there should be NO assert-errors if everything is OK. When compiling the test program, you can ignore warnings about unused variables. They occur because Stefan has used a special method of initializing all variables to the same start values before each test.

#### 12.17.6. Compilers

`wxString` and `wxRegex` have been compiled successfully with the following compilers (it should work on nearly every C++ compiler):

- PC MS-Visual C++ 1.0, 1.5
- UNIX gcc v2.6.3
- UNIX Sun SunPro compiler under Solaris 2.x
- VMS DEC C++ compiler (on VAX and AXP)

Warnings about type conversion or assignments can be ignored.

### 12.17.7. GNU Documentation

Below is the original GNU `wxString` and `wxRegex` documentation. It describes most functions of the classes. The function names have been capitalized to be consistent with the `wxWindows` naming scheme. The examples are integrated into the test program.

Copyright (C) 1988, 1991, 1992 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the section entitled "GNU Library General Public License" is included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that the section entitled "GNU Library General Public License" and this permission notice may be included in translations approved by the Free Software Foundation instead of in the original English.

#### 12.17.7.1. The `wxString` class

The '`wxString`' class is designed to extend GNU C++ to support string processing capabilities similar to those in languages like Awk. The class provides facilities that ought to be convenient and efficient enough to be useful replacements for '`char*`' based processing via the C string library (i.e., '`strcpy`', '`strcmp`', etc.) in many applications. Many details about `wxString` representations are described in the Representation section.

A separate '`wxSubString`' class supports substring extraction and modification operations. This is implemented in a way that user programs never directly construct or represent substrings, which are only used indirectly via `wxString` operations.

Another separate class, '`wxRegex`' is also used indirectly via `wxString` operations in support of regular expression searching, matching, and the like. The `wxRegex` class is based entirely on the GNU Emacs regex functions. See *Regular Expressions* (page 409) for a full explanation of regular expression syntax. (For implementation details, see the internal documentation in files `wxregex.h` and `wxregex.cc`).

#### 12.17.7.2. Constructor examples

Strings are initialized and assigned as in the following examples:

```
wxString x; Set x to the nil string. This is different from the original GNU code which sets a
strings also to nil when it is assign 0 or "".
```

```
wxString x = "Hello"; wxString y("Hello"); Set x and y to a copy of the string
"Hello".
```

```
wxString x = 'A'; wxString y('A'); Set x and y to the string value "A".
```

`wxString u = x; wxString v(x);` Set `u` and `v` to the same string as `wxString x`

`wxString u = x.At(1,4); wxString v(x.At(1,4));` Set `u` and `v` to the length 4 substring of `x` starting at position 1 (counting indexes from 0).

`wxString x("abc", 2);` Sets `x` to "ab", i.e., the first 2 characters of "abc".

There are no directly accessible forms for declaring `wxSubString` variables.

The declaration `wxRegex r("[a-zA-Z_][a-zA-Z0-9_]*");` creates compiled regular expression suitable for use in `wxString` operations described below. (In this case, one that matches any C++ identifier). The first argument may also be a `wxString`. Be careful in distinguishing the role of backslashes in quoted GNU C++ 'char\*' constants versus those in `Regexes`. For example, a `wxRegex` that matches either one or more tabs or all strings beginning with "ba" and ending with any number of occurrences of "na" could be declared as

```
wxRegex r = "\\(\\t+\\)|\\(ba\\(na\\)*\\)"
```

Note that only one backslash is needed to signify the tab, but two are needed for the parenthesization and virgule, since the GNU C++ lexical analyzer decodes and strips backslashes before they are seen by `wxRegex`.

There are three additional optional arguments to the `wxRegex` constructor that are less commonly useful:

`fast` (default 0) 'fast' may be set to true (1) if the `wxRegex` should be "fast-compiled". This causes an additional compilation step that is generally worthwhile if the `wxRegex` will be used many times.

`bufsize` (default `max(40, length of the string)`) This is an estimate of the size of the internal compiled expression. Set it to a larger value if you know that the expression will require a lot of space. If you do not know, do not worry: `realloc` is used if necessary.

`transtable` (default `none == 0`) The address of a byte translation table (a `char[256]`) that translates each character before matching.

As a convenience, several `Regexes` are predefined and usable in any program. Here are their declarations from `wxString.h`.

```
extern wxRegex RXwhite;      // = "[ \\n\\t]+"
extern wxRegex RXint;       // = "-?[0-9]+"
extern wxRegex RXdouble;    // = "-?\\(\\( [0-9]+\\. [0-9]*\\)\\)|
                             //   \\( [0-9]+\\)\\)|
                             //   \\(\\. [0-9]+\\)\\)"
                             // = "\\([eE][---+]?[0-9]+\\)?"
extern wxRegex RXalpha;     // = "[A-Za-z]+"
extern wxRegex RXlowercase; // = "[a-z]+"
extern wxRegex RXuppercase; // = "[A-Z]+"
extern wxRegex RXalphanum;  // = "[0-9A-Za-z]+"
extern wxRegex RXidentifier; // = "[A-Za-z_][A-Za-z0-9_]*"
```

### 12.17.7.3. Examples

Most `wxString` class capabilities are best shown via example. The examples below use the

following declarations.

```
wxString x = "Hello";
wxString y = "world";
wxString n = "123";
wxString z;
char *s = ", ";
wxString lft, mid, rgt;
wxRegex r = "e[a-z]*o";
wxRegex r2("/[a-z]*/");
char c;
int i, pos, len;
double f;
wxString words[10];
words[0] = "a";
words[1] = "b";
words[2] = "c";
```

#### 12.17.7.4. Comparing, Searching and Matching examples

The usual lexicographic relational operators ('==, !=, <, <=, >, >=') are defined. A functional form 'compare(wxString, wxString)' is also provided, as is 'fcompare(wxString, wxString)', which compares Strings without regard for upper vs. lower case.

All other matching and searching operations are based on some form of the (non-public) 'match' and 'search' functions. 'match' and 'search' differ in that 'match' attempts to match only at the given starting position, while 'search' starts at the position, and then proceeds left or right looking for a match. As seen in the following examples, the second optional 'startpos' argument to functions using 'match' and 'search' specifies the starting position of the search: If non-negative, it results in a left-to-right search starting at position 'startpos', and if negative, a right-to-left search starting at position 'x.Length() + startpos'. In all cases, the index returned is that of the beginning of the match, or -1 if there is no match.

Three wxString functions serve as front ends to 'search' and 'match'. 'index' performs a search, returning the index, 'matches' performs a match, returning nonzero (actually, the length of the match) on success, and 'contains' is a boolean function performing either a search or match, depending on whether an index argument is provided:

`x.Index("lo")` Returns the zero-based index of the leftmost occurrence of substring "lo" (3, in this case). The argument may be a wxString, wxSubString, char, char\*, or wxRegex.

`x.Index("l", 2)` Returns the index of the first of the leftmost occurrence of "l" found starting the search at position x[2], or 2 in this case.

`x.Index("l", -1)` Returns the index of the rightmost occurrence of "l", or 3 here.

`x.Index("l", -3)` Returns the index of the rightmost occurrence of "l" found by starting the search at the 3rd to the last position of x, returning 2 in this case.

`pos = r.Search("leo", 3, len, 0)` Returns the index of r in the char\* string of length 3, starting at position 0, also placing the length of the match in reference parameter len.

`x.Contains("He")` Returns nonzero if the wxString x contains the substring "He". The argument may be a wxString, wxSubString, char, char\*, or wxRegex.

`x.Contains("el", 1)` Returns nonzero if `x` contains the substring "el" at position 1. As in this example, the second argument to 'contains', if present, means to match the substring only at that position, and not to search elsewhere in the string.

`x.Contains(RXwhite)`; Returns nonzero if `x` contains any whitespace (space, tab, or newline). Recall that 'RXwhite' is a global whitespace `wxRegex`.

`x.Matches("lo", 3)` Returns nonzero if `x` starting at position 3 exactly matches "lo", with no trailing characters (as it does in this example).

`x.Matches(r)` Returns nonzero if `wxString` `x` as a whole matches `wxRegex` `r`.

`int f = x.Freq("l")` Returns the number of distinct, nonoverlapping matches to the argument (2 in this case).

### 12.17.7.5. Substring extraction examples

Substrings may be extracted via the 'at', 'before', 'through', 'from', and 'after' functions. These behave as either lvalues or rvalues.

`z = x.At(2, 3)` Sets `wxString` `z` to be equal to the length 3 substring of `wxString` `x` starting at zero-based position 2, setting `z` to "llo" in this case. A nil `wxString` is returned if the arguments don't make sense.

`x.At(2, 2) = "r"` Sets what was in positions 2 to 3 of `x` to "r", setting `x` to "Hero" in this case. As indicated here, `wxSubString` assignments may be of different lengths.

`x.At("He") = "je"`; `x("He")` is the substring of `x` that matches the first occurrence of its argument. The substitution sets `x` to "jello". If "He" did not occur, the substring would be nil, and the assignment would have no effect.

`x.At("l", -1) = "i"`; Replaces the rightmost occurrence of "l" with "i", setting `x` to "Helio".

`z = x.At(r)` Sets `wxString` `z` to the first match in `x` of `wxRegex` `r`, or "ello" in this case. A nil `wxString` is returned if there is no match.

`z = x.Before("o")` Sets `z` to the part of `x` to the left of the first occurrence of "o", or "Hell" in this case. The argument may also be a `wxString`, `wxSubString`, or `wxRegex`. (If there is no match, `z` is set to "").

`x.Before("ll") = "Bri"`; Sets the part of `x` to the left of "ll" to "Bri", setting `x` to "Brillo".

`z = x.Before(2)` Sets `z` to the part of `x` to the left of `x[2]`, or "He" in this case.

`z = x.After("Hel")` Sets `z` to the part of `x` to the right of "Hel", or "lo" in this case.

`z = x.Through("el")` Sets `z` to the part of `x` up and including "el", or "Hel" in this case.

`z = x.From("el")` Sets `z` to the part of `x` from "el" to the end, or "ello" in this case.

`x.After("Hel") = "p"`; Sets `x` to "Help";



`z = x.After(3)` Sets `z` to the part of `x` to the right of `x[3]` or "o" in this case.

`z = " ab c"; z = z.After(RXwhite)` Sets `z` to the part of its old string to the right of the first group of whitespace, setting `z` to "ab c"; Use `GSub`(below) to strip out multiple occurrences of whitespace or any pattern.

`x[0] = 'J'`; Sets the first element of `x` to 'J'. `x[i]` returns a reference to the *i*th element of `x`, or triggers an error if *i* is out of range.

`CommonPrefix(x, "Help")` Returns the `wxString` containing the common prefix of the two Strings or "Hel" in this case.

`CommonSuffix(x, "to")` Returns the `wxString` containing the common suffix of the two Strings or "o" in this case.

### 12.17.7.6. Concatenation examples

`z = x + s + ' ' + y.At("w") + y.After("w") + ".";` Sets `z` to "Hello, world."

`x += y;` Sets `x` to "Helloworld".

`Cat(x, y, z)` A faster way to say `z = x + y`.

`Cat(z, y, x, x)` Double concatenation; A faster way to say `x = z + y + x`.

`y.Prepend(x);` A faster way to say `y = x + y`.

`z = Replicate(x, 3);` Sets `z` to "HelloHelloHello".

`z = Join(words, 3, "/")` Sets `z` to the concatenation of the first 3 Strings in `wxString` array `words`, each separated by "/", setting `z` to "a/b/c" in this case. The last argument may be "" or 0, indicating no separation.

### 12.17.7.7. Other manipulation examples

`z = "this string has five words"; i = Split(z, words, 10, RXwhite);` Sets up to 10 elements of `wxString` array `words` to the parts of `z` separated by whitespace, and returns the number of parts actually encountered (5 in this case). Here, `words[0]` = "this", `words[1]` = "string", etc. The last argument may be any of the usual. If there is no match, all of `z` ends up in `words[0]`. The `words` array is *not* dynamically created by `split`.

`int nmatches x.GSub("l", "ll")` Substitutes all original occurrences of "l" with "ll", setting `x` to "HelliIo". The first argument may be any of the usual, including `wxRegex`. If the second argument is "" or 0, all occurrences are deleted. `gsub` returns the number of matches that were replaced.

`z = x + y; z.Del("loworl");` Deletes the leftmost occurrence of "loworl" in `z`, setting `z` to "Held".

`z = Reverse(x)` Sets `z` to the reverse of `x`, or "olleH".

`z = Uppcase(x)` Sets `z` to `x`, with all letters set to uppercase, setting `z` to "HELLO".

`z = Downcase(x)` Sets `z` to `x`, with all letters set to lowercase, setting `z` to "hello"

`z = Capitalize(x)` Sets `z` to `x`, with the first letter of each word set to uppercase, and all others to lowercase, setting `z` to "Hello"

`x.Reverse()`, `x.Uppcase()`, `x.Downcase()`, `x.Capitalize()` in-place, self-modifying versions of the above.

### 12.17.7.8. Reading, Writing and Conversion examples

`cout << x` Writes out `x`.

`cout << x.At(2, 3)` Writes out the substring "llo".

`cin >> x` Reads a whitespace-bounded string into `x`.

`x.Length()` Returns the length of `wxString x` (5, in this case).

`s = (const char*)x` Can be used to extract the 'char\*' char array. This coercion is useful for sending a `wxString` as an argument to any function expecting a 'const char\*' argument (like 'atoi', and 'File::open'). This operator must be used with care, since the conversion returns a pointer to 'wxString' internals without copying the characters: The resulting '(char\*)' is only valid until the next `wxString` operation, and you must not modify it. (The conversion is defined to return a const value so that GNU C++ will produce warning and/or error messages if changes are attempted.)

## 12.17.8. Regular Expressions

The following are extracts from GNU documentation.

### 12.17.8.1. Regular Expression Overview

Regular expression matching allows you to test whether a string fits into a specific syntactic shape. You can also search a string for a substring that fits a pattern.

A regular expression describes a set of strings. The simplest case is one that describes a particular string; for example, the string 'foo' when regarded as a regular expression matches 'foo' and nothing else. Nontrivial regular expressions use certain special constructs so that they can match more than one string. For example, the regular expression 'foo|bar' matches either the string 'foo' or the string 'bar'; the regular expression 'c[ad]\*r' matches any of the strings 'cr', 'car', 'cdr', 'caar', 'caddar' and all other such strings with any number of 'a's and 'd's.

The first step in matching a regular expression is to compile it. You must supply the pattern string and also a pattern buffer to hold the compiled result. That result contains the pattern in an internal format that is easier to use in matching.

Having compiled a pattern, you can match it against strings. You can match the compiled pattern any number of times against different strings.

### 12.17.8.2. Syntax of Regular Expressions

Regular expressions have a syntax in which a few characters are special constructs and the rest are "ordinary". An ordinary character is a simple regular expression which matches that character and nothing else. The special characters are '\\$', '^', '!', '\*', '+', '?', '[', ']' and '\\'. Any other character appearing in a regular expression is ordinary, unless a '\\' precedes it.

For example, 'f' is not a special character, so it is ordinary, and therefore 'f' is a regular expression that matches the string 'f' and no other string. (It does *not* match the string 'ff'.) Likewise, 'o' is a regular expression that matches only 'o'.

Any two regular expressions A and B can be concatenated. The result is a regular expression which matches a string if A matches some amount of the beginning of that string and B matches the rest of the string.

As a simple example, we can concatenate the regular expressions 'f' and 'o' to get the regular expression 'fo', which matches only the string 'fo'. Still trivial.

Note: for Unix compatibility, special characters are treated as ordinary ones if they are in contexts where their special meanings make no sense. For example, '\*foo' treats '\*' as ordinary since there is no preceding expression on which the '\*' can act. It is poor practice to depend on this behavior; better to quote the special character anyway, regardless of where it appears.

The following are the characters and character sequences which have special meaning within regular expressions. Any character not mentioned here is not special; it stands for exactly itself for the purposes of searching and matching.

- `.` is a special character that matches anything except a newline. Using concatenation, we can make regular expressions like `a.b` which matches any three-character string which begins with `a` and ends with `b`.
- `*` is not a construct by itself; it is a suffix, which means the preceding regular expression is to be repeated as many times as possible. In `f o *`, the `*` applies to the `o`, so `f o *` matches `f` followed by any number of `o`'s.

The case of zero `o`'s is allowed: `f o *` does match `f`.

`*` always applies to the *smallest* possible preceding expression. Thus, `f o *` has a repeating `o`, not a repeating `f o`.

The matcher processes a `*` construct by matching, immediately, as many repetitions as can be found. Then it continues with the rest of the pattern. If that fails, backtracking occurs, discarding some of the matches of the `*`'d construct in case that makes it possible to match the rest of the pattern. For example, matching `c[ad]*ar` against the string `caddaar`, the `[ad]*` first matches `addaa`, but this does not allow the next `a` in the pattern to match. So the last of the matches of `[ad]` is undone and the following `a` is tried again. Now it succeeds.

- `+` is like `*` except that at least one match for the preceding pattern is required for `+`. Thus, `c[ad]+r` does not match `cr` but does match anything else that `c[ad]*r` would match.

- `?` is like `*` except that it allows either zero or one match for the preceding pattern. Thus, `c[ad]?r` matches `cr` or `car` or `cd`, and nothing else.
- `[` begins a "character set", which is terminated by a `]`. In the simplest case, the characters between the two form the set. Thus, `[ad]` matches either `a` or `d`, and `[ad]*` matches any string of `a`'s and `d`'s (including the empty string), from which it follows that `c[ad]*r` matches `car`, etc.

Character ranges can also be included in a character set, by writing two characters with a `-` between them. Thus, `[a-z]` matches any lower-case letter. Ranges may be intermixed freely with individual characters, as in `[a-z$%.]`, which matches any lower case letter or `$`, `%` or period.

Note that the usual special characters are not special any more inside a character set. A completely different set of special characters exists inside character sets: `]`, `-` and `^`.

To include a `]` in a character set, you must make it the first character. For example, `[ ]a` matches `]` or `a`. To include a `-`, you must use it in a context where it cannot possibly indicate a range: that is, as the first character, or immediately after a range.

- `^` begins a "complement character set", which matches any character except the ones specified. Thus, `[^a-z0-9A-Z]` matches all characters *except* letters and digits.
- `^` is not special in a character set unless it is the first character. The character following the `^` is treated as if it were first (it may be a `-` or a `]`).
- `^` is a special character that matches the empty string -- but only if at the beginning of a line in the text being matched. Otherwise it fails to match anything. Thus, `^foo` matches `afoo` which occurs at the beginning of a line.
- `$` is similar to `^` but matches only at the end of a line. Thus, `xx*$` matches a string of one or more `x`'s at the end of a line.
- `\` has two functions: it quotes the above special characters (including `\`), and it introduces additional special constructs.

Because `\` quotes special characters, `\$` is a regular expression which matches only `$`, and `\[` is a regular expression which matches only `[`, and so on.

For the most part, `\` followed by any character matches only that character. However, there are several exceptions: characters which, when preceded by `\`, are special constructs. Such characters are always ordinary when encountered on their own.

No new special characters will ever be defined. All extensions to the regular expression syntax are made by defining new two-character constructs that begin with `\`.

- `\|` specifies an alternative. Two regular expressions A and B with `\|` in between form an expression that matches anything that either A or B will match.

Thus, `foo\|bar` matches either `foo` or `bar` but no other string.

`\|` applies to the largest possible surrounding expressions. Only a surrounding `\( ... \)` grouping can limit the grouping power of `\|`.

Full backtracking capability exists when multiple `\|`'s are used.

- `\( ... \)` is a grouping construct that serves three purposes:
  1. To enclose a set of `\|` alternatives for other operations. Thus, `\(foo\|bar\)x` matches either `foox` or `barx`.
  2. To enclose a complicated expression for the postfix `*` to operate on. Thus, `ba\(na\)*` matches `bananana`, etc., with any (zero or more) number of `na`'s.
  3. To mark a matched substring for future reference.

This last application is not a consequence of the idea of a parenthetical grouping; it is a separate feature which happens to be assigned as a second meaning to the same `\( ... \)` construct because there is no conflict in practice between the two meanings. Here is an explanation of this feature:

- `\DIGIT` After the end of a `\( ... \)` construct, the matcher remembers the beginning and end of the text matched by that construct. Then, later on in the regular expression, you can use `\` followed by `DIGIT` to mean "match the same text matched the `DIGIT`'th time by the `\( ... \)` construct." The `\( ... \)` constructs are numbered in order of commencement in the regexp.

The strings matching the first nine `\( ... \)` constructs appearing in a regular expression are assigned numbers 1 through 9 in order of their beginnings. `\1` through `\9` may be used to refer to the text matched by the corresponding `\( ... \)` construct.

For example, `\(.*\)\1` matches any string that is composed of two identical halves. The `\(.*)` matches the first half, which may be anything, but the `\1` that follows must match the same exact text.

- `\b` matches the empty string, but only if it is at the beginning or end of a word. Thus, `\bfoo\b` matches any occurrence of `foo` as a separate word. `\bball\(s\|\)\b` matches `ball` or `balls` as a separate word.
- `\B` matches the empty string, provided it is *\*not\** at the beginning or end of a word.
- `\<` matches the empty string, but only if it is at the beginning of a word.

- `\>` matches the empty string, but only if it is at the end of a word.
- `\w` matches any word-constituent character.
- `\W` matches any character that is not a word-constituent.

## 12.18. Writing a wxWindows application: a rough guide

To set a wxWindows application going, you'll need to derive a *wxApp* (page 44) class.

An application must have a top-level *wxFrame* (page 172) window (returned by *wxApp::OnInit* (page 46)), each frame containing one or more instances of *wxPanel* (page 228), *wxTextWindow* (page 302) or *wxCanvas* (page 57).

A frame can have a *wxMenuBar* (page 209), a status line, and a *wxIcon* (page 183) for when the frame is iconized.

A *wxPanel* (page 228) is used to place items (classes derived from *wxItem* (page 191)) which are used for user interaction. Examples of items are *wxButton* (page 54), *wxCheckBox* (page 68), *wxChoice* (page 69), *wxListBox* (page 201), *wxSlider* (page 278), *wxRadioBox* (page 260).

Instances of *wxDialogBox* (page 123) can also be used for panels, items 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 *dialog functions* (page 333), such as *wxMessageBox* (page 335) and *wxFileSelector* (page 333).

If you want to draw arbitrary graphics, you'll need a *wxCanvas* (page 57). In fact, you never draw directly onto a canvas---you use a *device context* (DC). *wxDC* (page 108) is the base for *wxCanvasDC* (page 68), *wxMemoryDC* (page 205), *wxPostScriptDC* (page 240), *wxMemoryDC* (page 205), *wxMetaFileDC* (page 213) and *wxPrinterDC* (page 250). 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* (page 110) and *wxDC::DrawText* (page 112). Control colour on a canvas (*wxColour* (page 76)) with brushes (*wxBrush* (page 51)) and pens (*wxPen* (page 237)).

On a canvas, you will probably need to intercept key events by overriding the *wxCanvas::OnChar* (page 62) member, and mouse events by overriding *wxCanvas::OnEvent* (page 63).

Most modern applications will have an on-line, hypertext help system; for this, you need *wxHelp* and the *wxHelpInstance* (page 187) class to control *wxHelp*. To add sparkle, you might use the *wxToolBar* class (documented separately) which makes heavy use of the *wxBitmap* (page 48).

GUI applications aren't all graphical wizardry. List and hash table needs are catered for by *wxList* (page 197), *wxStringList* (page 297) and *wxHashTable* (page 185). You will undoubtedly need some platform-independent *file functions* (page 328), and you may find it handy to maintain and search a list of paths using *wxPathList* (page 235). There's a *miscellany* (page 342) of operating

system and other functions.

If you have several communicating applications, you can try out the DDE-like functions, by using the three classes *wxClient* (page 74), *wxServer* (page 277) and *wxConnection* (page 91). These use DDE under Windows, and a simulation using sockets under UNIX.

## 12.19. 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* (page 231) 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* (page 322) function and a string comparison function such as *wxStringEq* (page 332). 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* (page 345).

For details of functions for manipulating resource files and loading user interface elements, see *wxWindows resource functions* (page 355).

### 12.19.1. 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 = ['aiai', wxBITMAP_TYPE_BMP_RESOURCE, 'WINDOWS'],\
    bitmap = ['aiai.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,\
'aiai_resource'],\
    control = [wxMessage, 'A Label', '', 'message7', 166, 61, -1, -1,\
'aiai_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(menuBar1);
    wxResourceParseData(aiai_resource);
    wxResourceParseData(project_resource);
    wxResourceParseData(panel3);
    ...
}
```

### 12.19.2. 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

### 12.19.3. 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.

### 12.19.4. 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).

### 12.19.5. 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).

#### 12.19.6. 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).

#### 12.19.7. Compiling the resource system

To enable the resource system, set **USE\_WX\_RESOURCES** to 1 in `wx_setup.h`. If your `wxWindows` makefile supports it, set the same name in the makefile to 1.

You will also need to compile the PrologIO utility (not always the easiest task): you will need YACC, and LEX (or FLEX). DOS versions of these are available on the AIAI ftp site under `/pub/wxwin/tools`.

#### 12.20. 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 `wxCanvas`, 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.

From version 1.50 beta (j), string return values 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.

## 12.21. `wxSplitterWindow` overview

Class: `wxSplitterWindow` (page 280)

A `wxSplitterWindow` manages one or two subwindows, allowing the user to change the position of a sash.

### 12.21.1. Example

The following fragment shows how to create a splitter window, creating two subwindows and hiding one of them.

```
splitter = new wxSplitterWindow(this, 0, 0, 400, 400, wxSP_3D);

leftCanvas = new MyCanvas(splitter);
leftCanvas->SetBackground(wxRED_BRUSH);
leftCanvas->SetScrollbars(20, 20, 50, 50, 4, 4);

rightCanvas = new MyCanvas(splitter);
rightCanvas->SetBackground(wxCYAN_BRUSH);
rightCanvas->SetScrollbars(20, 20, 50, 50, 4, 4);
rightCanvas->Show(FALSE);

splitter->Initialize(leftCanvas);

// Set this to prevent unsplitting
// splitter->SetMinimumPaneSize(20);
```

The next fragment shows how the splitter window can be manipulated after creation.

```
void MyFrame::OnMenuCommand(int id)
{
    switch (id)
    {
        case SPLIT_VERTICAL :
```

```
        if ( splitter->IsSplit() )
            splitter->Unsplit();
        leftCanvas->Show(TRUE);
        rightCanvas->Show(TRUE);
        splitter->SplitVertically( leftCanvas, rightCanvas );
        break;
    case SPLIT_HORIZONTAL :
        if ( splitter->IsSplit() )
            splitter->Unsplit();
        leftCanvas->Show(TRUE);
        rightCanvas->Show(TRUE);
        splitter->SplitHorizontally( leftCanvas, rightCanvas );
        break;
    case SPLIT_UNSPLOT :
        if ( splitter->IsSplit() )
            splitter->Unsplit();
        break;
    case SPLIT_QUIT:
        this->Close(TRUE);
        break;
    default:
        break;
    }
}
```



## References

- [1] **Boggan, Scott and Fakas, David and Welinske, Joe.** 1993. *Developing on-line help for Windows*. Sams Publishing. 11711 North College, Carmel, Indiana 46032, USA.
- [2] **Wong, William.** 1993. *Plug and play programming*. M and T Books. 115 West 18th Street, New York, New York 10011.
- [3] **Pree, Wolfgang.** 1994. *Design patterns for object-oriented software development*. Addison-Wesley. Reading, MA.
- [4] **Gamma, Erich and Helm, Richard and Johnson, Ralph and Vlissides, John.** 1994. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley. Reading, MA.
- [5] **Smart, Julian.** 1995. *wxCLIPS User Manual*. University of Edinburgh. Artificial Intelligence Applications Institute. 80 South Bridge, Edinburgh, EH1 1HN.
- [6] **Smart, Julian.** 1995. *Tex2RTF User Manual*. University of Edinburgh. Artificial Intelligence Applications Institute. 80 South Bridge, Edinburgh, EH1 1HN.





## Index



`::copystring`, 332  
`::Dos2UnixFilename`, 328  
`::NewId`, 342  
`::RegisterId`, 343  
`::wxAddPrimaryEventHandler`, 337  
`::wxAddSecondaryEventHandler`, 337  
`::wxBeginBusyCursor`, 343  
`::wxBell`, 343  
`::wxCleanUp`, 343  
`::wxClipboardOpen`, 341  
`::wxCloseClipboard`, 341  
`::wxColourDisplay`, 335  
`::wxConcatFiles`, 329  
`::wxCopyFile`, 329  
`::wxCreateDynamicObject`, 343  
`::wxDebugMsg`, 343  
`::wxDirExists`, 328  
`::wxDisplayDepth`, 336  
`::wxDisplaySize`, 344  
`::wxEmptyClipboard`, 341  
`::wxEndBusyCursor`, 344  
`::wxEntry`, 344  
`::wxEnumClipboardFormats`, 341  
`::wxError`, 344  
`::wxExecute`, 344  
`::wxExit`, 345  
`::wxFatalError`, 345  
`::wxFileExists`, 328  
`::wxFileNameFromPath`, 328  
`::wxFileSelector`, 333  
`::wxFindFirstFile`, 328  
`::wxFindMenuItemId`, 345  
`::wxFindNextFile`, 329  
`::wxFindWindowByLabel`, 345  
`::wxFindWindowByName`, 345  
`::wxGetActiveWindow`, 346  
`::wxGetClipboardData`, 341  
`::wxGetClipboardFormatName`, 342  
`::wxGetDisplayName`, 346  
`::wxGetElapsedTime`, 346  
`::wxGetEmailAddress`, 330  
`::wxGetFreeMemory`, 346  
`::wxGetHomeDir`, 346  
`::wxGetHostName`, 329, 346  
`::wxGetMousePosition`, 347  
`::wxGetMultipleChoice`, 334  
`::wxGetOsVersion`, 347  
`::wxGetPrinterCommand`, 338  
`::wxGetPrinterFile`, 338  
`::wxGetPrinterMode`, 339  
`::wxGetPrinterOptions`, 339  
`::wxGetPrinterOrientation`, 339  
`::wxGetPrinterPreviewCommand`, 339  
`::wxGetPrinterScaling`, 339  
`::wxGetPrinterTranslation`, 339  
`::wxGetResource`, 347  
`::wxGetSingleChoice`, 334  
`::wxGetSingleChoiceData`, 335  
`::wxGetSingleChoiceIndex`, 334  
`::wxGetTempFileName`, 331  
`::wxGetTextFromUser`, 333  
`::wxGetUserId`, 330, 348  
`::wxGetUserName`, 330, 348  
`::wxGetWorkingDirectory`, 330  
`::wxInitClipboard`, 348  
`::wxIPCCleanUp`, 348  
`::wxIPCInitialize`, 348  
`::wxIsAbsolutePath`, 329  
`::wxIsBusy`, 348  
`::wxIsClipboardFormatAvailable`, 342  
`::wxIsWild`, 331  
`::wxKill`, 348  
`::wxLoadUserResource`, 349  
`::wxMakeMetaFilePlaceable`, 336  
`::wxMatchWild`, 331  
`::wxMessageBox`, 335  
`::wxMkdir`, 331  
`::wxNotifyEvent`, 337  
`::wxNow`, 349  
`::wxOpenClipboard`, 342  
`::wxPathOnly`, 329  
`::wxPostDelete`, 349  
`::wxRegisterClipboardFormat`, 342  
`::wxRegisterEventClass`, 337  
`::wxRegisterEventName`, 338  
`::wxRegisterExternalEventHandlers`, 338  
`::wxRemoveFile`, 331  
`::wxRemoveSecondaryEventHandler`, 338  
`::wxRenameFile`, 331  
`::wxResourceAddIdentifier`, 355  
`::wxResourceClear`, 355  
`::wxResourceCreateBitmap`, 355  
`::wxResourceCreateIcon`, 356  
`::wxResourceCreateMenuBar`, 356  
`::wxResourceGetIdentifier`, 356  
`::wxResourceParseData`, 357  
`::wxResourceParseFile`, 357  
`::wxResourceParseString`, 357  
`::wxResourceRegisterBitmapData`, 358  
`::wxResourceRegisterIconData`, 358  
`::wxRmdir`, 331  
`::wxSendEvent`, 338  
`::wxSetClipboardData`, 342  
`::wxSetCursor`, 336  
`::wxSetDisplayName`, 349  
`::wxSetPrinterCommand`, 339  
`::wxSetPrinterFile`, 340  
`::wxSetPrinterMode`, 340  
`::wxSetPrinterOptions`, 340  
`::wxSetPrinterOrientation`, 340  
`::wxSetPrinterPreviewCommand`, 340  
`::wxSetPrinterScaling`, 340  
`::wxSetPrinterTranslation`, 340  
`::wxSetWorkingDirectory`, 332

::wxShell, 350  
 ::wxSleep, 350  
 ::wxStartTimer, 350  
 ::wxStringEq, 332  
 ::wxStringMatch, 332  
 ::wxStripMenuCodes, 350  
 ::wxSubType, 350  
 ::wxToLower, 351  
 ::wxToUpper, 351  
 ::wxTrace, 351  
 ::wxTraceLevel, 351  
 ::wxTransferFileToStream, 332  
 ::wxTransferStreamToFile, 332  
 ::wxUnix2DosFilename, 329  
 ::wxWriteResource, 351  
 ::wxYield, 352

— \_ —

\_\_type, 222

— ~ —

~wxApp, 44  
 ~wxBitmap, 49  
 ~wxBrush, 52  
 ~wxButton, 55  
 ~wxCanvas, 57  
 ~wxCheckBox, 69  
 ~wxChoice, 70  
 ~wxColourData, 78  
 ~wxColourDialog, 81  
 ~wxColourMap, 81  
 ~wxComboBox, 83  
 ~wxCommand, 87  
 ~wxCommandProcessor, 90  
 ~wxCursor, 96  
 ~wxDatabase, 96  
 ~wxDate, 102  
 ~wxDC, 108  
 ~wxDialogBox, 124  
 ~wxDocChildFrame, 127  
 ~wxDocManager, 129  
 ~wxDocParentFrame, 134  
 ~wxDocTemplate, 138  
 ~wxDocument, 141  
 ~wxEnhDialogBox, 147  
 ~wxEvent, 149  
 ~wxEvtHandler, 151  
 ~wxFileHistory, 157  
 ~wxFont, 159  
 ~wxFontData, 160  
 ~wxFontDialog, 163  
 ~wxFontNameDirectory, 164  
 ~wxForm, 168  
 ~wxFrame, 173  
 ~wxGauge, 181  
 ~wxGroupBox, 183  
 ~wxHashTable, 186  
 ~wxIcon, 184  
 ~wxList, 199  
 ~wxListBox, 203

~wxMenu, 207  
 ~wxMenuBar, 209  
 ~wxMessage, 212  
 ~wxMetaFile, 213  
 ~wxMetaFileDC, 214  
 ~wxPageSetupData, 223  
 ~wxPageSetupDialog, 227  
 ~wxPanel, 228  
 ~wxPen, 237  
 ~wxPreviewCanvas, 241  
 ~wxPreviewControlBar, 243  
 ~wxPreviewFrame, 244  
 ~wxPrintData, 245  
 ~wxPrintDialog, 248  
 ~wxPrinter, 249, 254  
 ~wxPrintout, 251  
 ~wxQueryCol, 257  
 ~wxQueryField, 259  
 ~wxRadioBox, 261  
 ~wxRadioButton, 264  
 ~wxRecordSet, 265  
 ~wxScrollBar, 276  
 ~wxSlider, 279  
 ~wxSplitterWindow, 281  
 ~wxString, 286  
 ~wxStringList, 298  
 ~wxText, 300  
 ~wxTextWindow, 303  
 ~wxTimer, 308  
 ~wxToolBar, 309  
 ~wxView, 316  
 ~wxWindow, 319

## —A—

A minimal wxWindows program, 36  
 A selection of SQL commands, 396  
 Abort, 249  
 Above, 190  
 Absolute, 190  
 Activate, 316  
 ActivateView, 129  
 Add, 168, 236, 298  
 AddBrush, 53  
 AddChild, 319  
 AddCmd, 148  
 AddDocument, 129  
 AddEnvList, 236  
 AddFileToHistory, 129, 157  
 AddFont, 164  
 AddMonths, 102  
 AddNew, 265  
 AddPen, 240  
 AddSeparator, 309  
 AddTool, 309  
 AddType, 314  
 AddView, 141  
 AddWeeks, 102  
 AddYears, 102  
 Advise, 92  
 After, 287  
 aligning items, 368

Alloc, 286  
 Allocating and deleting wxWindows objects, 22  
 Allocation, 286  
 AllocData, 259  
 AllowDoubleClick, 58  
 Append, 71, 83, 199, 203, 207, 209, 287  
 AppendField, 258  
 AppendSeparator, 207  
 argc, 44  
 argv, 44  
 AsIs, 190  
 Assignment, 363  
 AssociatePanel, 168  
 AssociateTemplate, 130  
 At, 287

## —B—

Before, 287  
 BeginDrawing, 58, 108  
 BeginFind, 186  
 BeginQuery, 265  
 BeginTrans, 96  
 BeingReplaced, 76  
 Below, 190  
 BindVar, 257, 265  
 Bitmap resource format, 418  
 Blit, 108  
 Blue, 77  
 bottom, 196  
 Break, 207  
 Button, 216  
 ButtonDClick, 216  
 ButtonDown, 216  
 buttonFlags, 242  
 buttonFont, 242  
 ButtonUp, 216

## —C—

CanAppend, 265  
 Cancel, 96, 266  
 CanRestart, 266  
 CanScroll, 266  
 CanTransact, 97, 266  
 CanUndo, 87, 90  
 CanUpdate, 97, 266  
 Capitalize, 287  
 CaptureMouse, 319  
 Cat, 288  
 Center, 319  
 Centre, 124, 173, 192, 320  
 centreX, 197  
 centreY, 197  
 Chars, 288  
 Check, 120, 207, 210  
 Check Windows debug messages, 43  
 Checked, 89, 208, 210  
 childDocument, 126  
 childView, 126  
 Classification, 363  
 CLASSINFO, 352

Clear, 58, 71, 83, 109, 186, 199, 203, 303  
 ClearCommands, 90  
 ClearData, 259  
 clientData, 88  
 ClientToScreen, 320  
 Close, 97, 141, 214, 317, 320  
 closeButton, 242  
 Command, 174, 192  
 commandInt, 88  
 commandString, 88  
 CommitTrans, 97  
 CommonPrefix, 296  
 CommonSuffix, 296  
 Compare, 296  
 CompareTo, 289  
 Comparing, Searching and Matching examples, 406  
 Comparison operators, 296  
 Comparisons (case sensitive and insensitive), 363  
 Compilers, 403  
 Compiling the resource system, 419  
 Composition and Concatenation, 364  
 Concatenation examples, 408  
 Constraint layout: more detail, 388  
 Constraints on form items, 166  
 ConstructDefaultSQL, 266  
 Constructor examples, 404  
 Constructors/Destructors, 364  
 Contains, 289  
 controlBar, 244  
 controlDown, 193, 214  
 ControlDown, 195, 216  
 Conversions, 364  
 Copy, 83, 289, 300, 303  
 Copyright of the original GNU code portion, 400  
 copystring, 332  
 Create, 49, 55, 58, 69, 71, 82, 83, 124, 174, 181, 183, 203, 212, 229, 261, 264, 276, 278, 279, 300, 303, 396  
 CreateAbortWindow, 249  
 CreateButtons, 243  
 CreateCanvas, 245  
 CreateControlBar, 245  
 CreateDocument, 130, 138  
 CreateItem, 229  
 CreateObject, 73  
 CreateStatusLine, 174  
 CreateTools, 310  
 CreateView, 130, 138  
 CrossHair, 58, 109  
 currentView, 128  
 Cut, 84, 300, 303

## —D—

Data, 221  
 Data transfer, 379  
 DECLARE\_ABSTRACT\_CLASS, 352  
 DECLARE\_CLASS, 353  
 DECLARE\_DYNAMIC\_CLASS, 353  
 defaultDocumentNameCounter, 128

Definition of constructors, 24  
 Del, 289  
 delete, 223  
 Delete, 84, 168, 186, 203, 266, 298  
 DeleteAllViews, 142  
 DeleteContents, 199  
 DeleteNode, 199  
 DeleteObject, 200  
 Deletion/Insertion, 364  
 Deselect, 84, 203  
 DestroyChildren, 320  
 DestroyClippingRegion, 58, 109  
 DeviceToLogicalX, 109  
 DeviceToLogicalXRel, 109  
 DeviceToLogicalY, 109  
 DeviceToLogicalYRel, 109  
 Dialog resource format, 416  
 DisassociateTemplate, 130  
 DiscardEdits, 303  
 Disconnect, 92  
 Dispatch, 45  
 DisplayBlock, 188  
 DisplayContents, 188  
 DisplaySection, 188  
 Do, 87, 90  
 documentFile, 140  
 documentModified, 140  
 documentTemplate, 140  
 documentTitle, 141  
 documentTypeName, 141  
 documentViews, 141  
 Dos2UnixFilename, 328  
 Downcase, 290  
 DragAcceptFiles, 320  
 Dragging, 217  
 DrawAllStaticItems, 229  
 DrawArc, 58, 110  
 DrawBlankPage, 254  
 DrawEllipse, 59, 110  
 DrawEllipticArc, 110  
 DrawIcon, 110  
 DrawLine, 59, 110  
 DrawLines, 59, 110  
 DrawPoint, 59, 111  
 DrawPolygon, 59, 111  
 DrawRectangle, 60, 111  
 DrawRoundedRectangle, 60, 111  
 DrawSpline, 60, 112  
 DrawText, 60, 112  
 DrawTool, 310  
 Dump, 120, 222  
 Duplication, 364

## —E—

Edges and relationships, 189  
 Edit, 266  
 Elem, 290  
 Element access, 364  
 Empty, 290  
 Enable, 208, 210, 262, 321  
 EnableEffects, 160

EnableHelp, 223, 245  
 EnableMargins, 224  
 EnableOrientation, 224  
 EnablePageNumbers, 246  
 EnablePaper, 224  
 EnablePrinter, 224  
 EnablePrintToFile, 246  
 EnableScrolling, 60  
 EnableSelection, 246  
 EnableTool, 310  
 EnableTop, 210  
 EndDoc, 112  
 EndDrawing, 61, 112  
 EndPage, 112  
 EndQuery, 267  
 EnsureFileAccessible, 236  
 Entering, 217  
 Error, 290  
 ErrorOccured, 97  
 ErrorSnapshot, 97  
 eventClass, 149  
 eventHandle, 149  
 eventObject, 149  
 eventType, 149  
 Example, 167, 371, 420  
 Example 1: subwindow layout, 389  
 Example 2: panel item layout, 390  
 Examples, 380, 405  
 Execute, 92  
 ExecuteSQL, 267  
 ExitMainLoop, 46  
 Extraction of Substrings, 364  
 extraLong, 88

## —F—

FCompare, 296  
 Features/Additions/Modifications, 400  
 fileHistory, 128, 156  
 FileHistoryLoad, 130, 157  
 fileHistoryN, 156  
 FileHistorySave, 130, 157  
 FileHistoryUseMenu, 131, 157  
 fileMaxFiles, 156  
 fileMenu, 157  
 FillVar, 257  
 FillVars, 267  
 Find, 200  
 FindClass, 73  
 FindColour, 80  
 FindItem, 168, 208  
 FindItemById, 210  
 FindItemForId, 208  
 FindMenuItem, 210  
 FindName, 80  
 FindOrCreateBrush, 54  
 FindOrCreateFont, 164  
 FindOrCreateFontId, 164  
 FindOrCreatePen, 240  
 FindString, 71, 84, 204, 262  
 FindTemplateForPath, 131  
 FindToolForPosition, 310

FindValidPath, 236  
First, 200, 290  
Firstchar, 290  
Fit, 148, 174, 229  
FloodFill, 61, 112  
Font name directory overview, 382  
Form appearance, 166  
FormatDate, 102  
formats, 76  
Freq, 290  
From, 290, 291  
Function calls, 402  
Functions for making form items and constraints, 170

## —G—

General features, 18  
Genetic mutation, 42  
Get, 77, 186  
GetAFMName, 165  
GetAllowSymbols, 160  
GetAllPages, 246  
GetAppName, 45  
GetBackground, 113  
GetBackgroundColour, 192, 230  
GetBaseClassName1, 73  
GetBaseClassName2, 73  
GetBezelFace, 181  
GetBrush, 113  
GetButtonColour, 192, 230  
GetButtonFont, 229  
GetCanvas, 254  
GetCap, 237  
GetCharHeight, 113, 321  
GetCharWidth, 113, 321  
GetCheckPrevious, 121  
GetChildren, 321  
GetChooseFull, 78  
GetChosenFont, 161  
GetClassInfo, 222  
GetClassName, 45, 73  
GetClientData, 84, 89, 151, 204  
GetClientSize, 321  
GetClipboardClient, 75  
GetClipboardData, 75  
GetClipboardString, 75  
GetClippingBox, 113  
GetCmd, 148  
GetCollate, 246  
GetColName, 267  
GetColour, 52, 78, 161, 237  
GetColourData, 81  
GetColourMap, 49  
GetColType, 267  
GetColumns, 71, 267  
GetCommandProcessor, 142  
GetCommands, 90  
GetConstraints, 321  
GetContents, 303  
GetCurrentDocument, 131  
GetCurrentPage, 254  
GetCurrentRecord, 268  
GetCurrentView, 131  
GetCursor, 229  
GetCustomColour, 78  
GetDashes, 238  
GetData, 76, 257, 259, 291  
GetDatabase, 268  
GetDatabaseName, 97  
GetDataSource, 97  
GetDataSources, 268  
GetDay, 103  
GetDayOfWeek, 103  
GetDayOfWeekName, 103  
GetDayOfYear, 103  
GetDaysInMonth, 103  
GetDC, 61, 251  
GetDebugMode, 121  
GetDefaultButtonHeight, 56  
GetDefaultButtonWidth, 56  
GetDefaultConnect, 269  
GetDefaultExtension, 138  
GetDefaultInfo, 226  
GetDefaultItem, 229  
GetDefaultMinMargins, 225  
GetDefaultSQL, 269  
GetDepth, 49  
GetDescription, 138  
GetDirectory, 138  
GetDocument, 127, 317  
GetDocumentManager, 138, 142, 317  
GetDocumentName, 139, 142  
GetDocuments, 131  
GetDocumentTemplate, 142  
GetDocumentWindow, 142  
GetEditMenu, 91  
GetEnableEffects, 161  
GetEnableHelp, 225  
GetEnableMargins, 225  
GetEnableOrientation, 225  
GetEnablePaper, 225  
GetEnablePrinter, 225  
GetErrorClass, 98  
GetErrorCode, 98, 269  
GetErrorMessage, 98  
GetErrorNumber, 98  
GetEventClass, 150  
GetEventHandler, 321  
GetEventObject, 150  
GetEventType, 150  
GetExitOnDelete, 45  
GetFaceName, 159  
GetFamily, 159, 165  
GetFieldData, 269  
GetFieldDataPtr, 269, 270  
GetFileFilter, 139  
GetFileHistory, 131  
GetFilename, 142  
GetFilter, 270  
GetFirstDayOfMonth, 103  
GetFirstView, 143  
GetFlags, 139  
GetFont, 113

GetFontData, 163  
 GetFontId, 159, 165  
 GetFontName, 165  
 GetFrame, 254, 317  
 GetFromPage, 246  
 GetGrandParent, 322  
 GetHandle, 322  
 GetHDBC, 98  
 GetHeight, 50, 185  
 GetHelpString, 208, 210  
 GetHENV, 98  
 GetHorizontalSpacing, 230  
 GetInfo, 98, 99  
 GetInitialFont, 161  
 GetInsertionPoint, 84, 300, 304  
 GetJoin, 238  
 GetJulianDate, 103  
 GetLabel, 192, 208, 211, 322  
 GetLabelColour, 192, 230  
 GetLabelFont, 230  
 GetLabelTop, 211  
 GetLastPosition, 84, 300, 304  
 GetLevel, 121  
 GetLineLength, 220, 304  
 GetLineText, 220, 304  
 GetLogicalFunction, 113  
 GetMapMode, 114  
 GetMarginBottomRight, 224  
 GetMarginTopLeft, 224, 226  
 GetMax, 279  
 GetMaxCommands, 90  
 GetMaxDocsOpen, 131  
 GetMaxFiles, 158  
 GetMaxPage, 246, 255  
 GetMaxSize, 310  
 GetMenuBar, 174  
 GetMin, 279  
 GetMinimumPaneSize, 281  
 GetMinMarginBottomRight, 225  
 GetMinMarginTopLeft, 224  
 GetMinPage, 247, 255  
 GetMonth, 103  
 GetMonthEnd, 104  
 GetMonthName, 104  
 GetMonthStart, 104  
 GetName, 87, 257, 314, 322  
 GetNewFontId, 165  
 GetNextHandler, 151  
 GetNoCopies, 247  
 GetNoHistoryFiles, 132, 158  
 GetNumberCols, 271  
 GetNumberFields, 271  
 GetNumberOfLines, 220, 304  
 GetNumberParams, 271  
 GetNumberRecords, 271  
 GetObjectType, 150  
 GetODBCVersionFloat, 99  
 GetODBCVersionString, 99  
 GetOptimization, 114  
 GetOptions, 271  
 GetOrientation, 225  
 GetPageInfo, 251  
 GetPageSetupData, 227  
 GetPageSizeMM, 251  
 GetPageSizePixels, 252  
 GetPanelDC, 230  
 GetPanelItem, 172  
 GetPaperSize, 224  
 GetParent, 322  
 GetPassword, 99  
 GetPen, 114  
 GetPixel, 114  
 GetPointSize, 159  
 GetPosition, 322  
 GetPostScriptName, 165  
 GetPPIPrinter, 252  
 GetPPIScreen, 252  
 GetPreviousHandler, 151  
 GetPrimaryKeys, 270, 271  
 GetPrintableName, 143  
 GetPrintData, 248, 250, 255  
 GetPrintDC, 249  
 GetPrintMode, 45  
 GetPrintout, 255  
 GetPrintoutForPrinting, 255  
 GetPrintPreview, 243  
 GetRange, 181  
 GetResultSet, 271  
 GetSashPosition, 281  
 GetScreenName, 165  
 GetScrollPage, 61  
 GetScrollPixelsPerUnit, 61  
 GetScrollPos, 61  
 GetScrollRange, 61  
 GetScrollUnitsPerPage, 62  
 GetSelection, 71, 85, 89, 204, 262  
 GetSelections, 204  
 GetShowHelp, 161  
 GetSize, 73, 114, 258, 259, 322  
 GetSortString, 272  
 GetSplitMode, 281  
 GetSQL, 272  
 GetStipple, 52, 238  
 GetStream, 121, 241  
 GetStreamBuf, 121  
 GetString, 71, 72, 85, 89, 204, 263  
 GetStringSelection, 72, 85, 204, 262  
 GetStyle, 52, 159, 238  
 GetTableName, 272  
 GetTables, 272  
 GetTextBackground, 114  
 GetTextExtent, 115, 323  
 GetTextForeground, 115  
 GetTitle, 125, 143, 174, 208  
 GetToolBar, 175  
 GetToolClientData, 311  
 GetToolEnabled, 311  
 GetToolLongHelp, 311  
 GetToolShortHelp, 311  
 GetToolState, 311  
 GetToPage, 247  
 GetTopWindow, 45  
 GetType, 257, 260, 272  
 GetUnderlined, 160

GetUserEditMode, 323  
 GetUsername, 99  
 GetValue, 69, 85, 181, 220, 264, 277, 279, 300  
 GetValues, 277  
 GetVerticalSpacing, 230  
 GetView, 127  
 GetViewName, 139, 317  
 GetVirtualSize, 62  
 GetWeekOfMonth, 104  
 GetWeekOfYear, 104  
 GetWeight, 160  
 GetWidth, 50, 185, 238, 315  
 GetWindow1, 281  
 GetWindow2, 282  
 GetWindowStyleFlag, 323  
 GetX, 315  
 GetY, 315  
 GetYear, 104  
 GetYearEnd, 104  
 GetYearStart, 104  
 GetZoomControl, 243  
 GNU Documentation, 404  
 GoTo, 272  
 Green, 77  
 GSub, 291

## —H—

HasPage, 252  
 HasStream, 121  
 Header files, 403  
 height, 197

## —I—

Icon resource format, 418  
 Iconize, 125, 175  
 Iconized, 125, 175  
 IMPLEMENT\_ABSTRACT\_CLASS, 353  
 IMPLEMENT\_ABSTRACT\_CLASS2, 354  
 IMPLEMENT\_CLASS, 354  
 IMPLEMENT\_CLASS2, 354  
 IMPLEMENT\_DYNAMIC\_CLASS, 354  
 IMPLEMENT\_DYNAMIC\_CLASS2, 354  
 Index, 291  
 Initialize, 80, 91, 132, 165, 187, 245, 282  
 InitializeClasses, 74  
 Initialized, 46  
 Input/Output, 365  
 Insert, 200, 291, 396  
 IntDrawLine, 62, 115  
 IntDrawLines, 62, 115  
 Interval, 308  
 InWaitForDataSource, 100  
 IsAscii, 291  
 IsBOF, 272  
 IsButton, 217  
 IsColNullable, 273  
 IsDefined, 291  
 IsDeleted, 273  
 IsDirty, 260  
 IsEditable, 169

IsEOF, 273  
 IsFieldDirty, 273  
 IsFieldNull, 273  
 IsKindOf, 74, 222  
 IsLeapYear, 105  
 IsModal, 125  
 IsModified, 143  
 IsModify, 143  
 IsNull, 292  
 IsNullable, 258  
 IsNumber, 292  
 IsOpen, 100, 273  
 IsPreview, 252  
 IsRetained, 62  
 IsRowDirty, 258  
 IsSelection, 89  
 IsShown, 323  
 IsSplit, 282  
 IsVisible, 139  
 IsWord, 292

## —J—

Join, 297

## —K—

keyCode, 194  
 KeyCode, 195  
 KeywordSearch, 188

## —L—

Last, 200, 292  
 Lastchar, 292  
 Layout, 311, 323  
 Leaving, 217  
 left, 197  
 LeftDClick, 217  
 leftDown, 214, 215  
 LeftDown, 217  
 LeftIsDown, 217  
 LeftOf, 190  
 LeftUp, 218  
 Length, 292  
 ListToArray, 298  
 LoadAccelerators, 175  
 LoadFile, 50, 188, 304  
 LoadFromResource, 231  
 Loading bitmaps: further information, 384  
 LoadObject, 143, 222  
 LogicalToDeviceX, 115  
 LogicalToDeviceXRel, 115  
 LogicalToDeviceY, 116  
 LogicalToDeviceYRel, 116  
 Lower, 323  
 LowerCase, 292

## —M—

MainLoop, 46



MakeConnection, 74  
 MakeDefaultName, 132  
 MakeKey, 186  
 MakeModal, 323  
 Matches, 292, 293  
 maxDocsOpen, 128  
 Maximize, 175  
 MaxX, 116  
 MaxY, 116  
 MDI versus SDI, 9  
 Member, 200, 236, 298  
 Menubar resource format, 418  
 MiddleDClick, 218  
 middleDown, 214  
 MiddleDown, 218  
 MiddleIsDown, 218  
 MiddleUp, 218  
 MinX, 116  
 MinY, 116  
 mnDocs, 128  
 mnFlags, 128  
 mnTemplates, 129  
 Modified, 305  
 Module definition file, 21  
 More advanced features: the *hello* demo, 38  
 More DDE details, 380  
 Move, 274, 324  
 MoveFirst, 274  
 MoveLast, 274  
 MoveNext, 274  
 MovePrev, 274  
 Moving, 218

—N—

new, 223  
 NewId, 342  
 NewLine, 231  
 Next, 186, 221  
 nextHandler, 150  
 nextPageButton, 242  
 Notify, 308  
 Nth, 200  
 Number, 72, 85, 201, 204, 262

—O—

ODBC SQL data types, 395  
 Ok, 50, 77, 116, 213, 255  
 OK, 293  
 OnAcceptConnection, 278  
 OnActivate, 127, 151, 175  
 OnActivateView, 317  
 OnAdvise, 92  
 OnBeginDocument, 252  
 OnBeginPrinting, 253  
 OnCancel, 169  
 OnChangedViewList, 144  
 OnChangeFilename, 317  
 OnChar, 62, 152, 305  
 OnCharHook, 46, 125, 152  
 OnClose, 127, 135, 152, 176, 245, 318

OnCloseDocument, 144  
 OnCommand, 152, 231  
 OnCreate, 144, 318  
 OnCreateCommandProcessor, 144  
 OnCreateFileHistory, 132  
 OnCreatePrintout, 318  
 OnDefaultAction, 152, 231  
 OnDisconnect, 93  
 OnDoubleClickSash, 282  
 OnDropFiles, 152  
 OnEndDocument, 253  
 OnEndPrinting, 253  
 OnEvent, 63, 153, 232  
 OnExecute, 93  
 OnExit, 46  
 OnFileClose, 132  
 OnFileNew, 132  
 OnFileOpen, 132  
 OnFileSave, 133  
 OnFileSaveAs, 133  
 OnHelp, 169  
 OnInit, 47  
 OnItemEvent, 153, 232  
 OnItemLeftClick, 153, 232  
 OnItemMove, 153, 232  
 OnItemRightClick, 153, 233  
 OnItemSelect, 154  
 OnItemSize, 154, 233  
 OnKillFocus, 154  
 OnLeftClick, 154, 233, 311  
 OnMakeConnection, 74  
 OnMenuCommand, 127, 133, 135, 154, 176  
 OnMenuSelect, 155, 177  
 OnMouseEnter, 312  
 OnMove, 155  
 OnNewDocument, 144  
 OnOk, 169  
 OnOpenDocument, 144  
 OnPaint, 63, 155, 233, 241, 243  
 OnPoke, 93  
 OnPreparePrinting, 253  
 OnPrintPage, 253  
 OnQuit, 189  
 OnRequest, 93  
 OnRevert, 169  
 OnRightClick, 154, 233, 312  
 OnSaveDocument, 145  
 OnSaveModified, 145  
 OnScroll, 64, 155  
 OnSelect, 155  
 OnSetFocus, 155  
 OnSetOptions, 100  
 OnSize, 156, 178  
 OnStartAdvise, 93  
 OnStopAdvise, 93  
 OnUnsplit, 283  
 OnUpdate, 169, 318  
 OnWaitForDataSource, 100  
 Open, 100  
 operator -, 106  
 operator --, 107  
 operator !=, 107, 296, 297

operator (), 287, 295  
operator [], 295  
operator +, 106, 297  
operator ++, 106, 315  
operator +=, 106, 295  
operator <, 107, 296, 297  
operator <<, 107, 295, 307  
operator <=, 107, 296, 297  
operator =, 77, 79, 162, 295  
operator -=, 106  
operator ==, 107, 296, 297  
operator >, 107, 296, 297  
operator >=, 107, 296, 297  
operator >>, 295  
operator char \*, 106  
operator const char\*, 295  
Other manipulation examples, 408

## —P—

PaintPage, 255  
PaintSelectionHandles, 234  
Paste, 85, 301, 305  
Pending, 47  
PercentOf, 191  
Play, 213  
Pointers to functions, 24  
Poke, 93  
PopupMenu, 324  
Position, 196, 218  
PositionToXY, 220, 305  
Positive thinking, 41  
Precompiled headers, 25  
Prepend, 293  
previewCanvas, 244  
Previous, 221  
previousHandler, 151  
previousPageButton, 242  
Print, 250, 256  
PrintClasses, 121  
PrintDialog, 250  
printPreview, 242, 244  
PrintStatistics, 122  
Procedures for writing an ODBC application, 394  
ProcessMessage, 47  
Put, 187

## —Q—

Query, 274  
Quit, 189

## —R—

Raise, 324  
ReadEvent, 150  
Reading, Writing and Conversion examples, 409  
Readline, 293  
RecordCountFinal, 274  
Red, 77  
Refresh, 324  
RegisterId, 343

Regular Expression Overview, 409  
Regular Expressions, 409  
ReleaseMouse, 324  
Remote Procedure Call, 380  
Remove, 85, 293, 301, 305  
RemoveBrush, 54  
RemoveDocument, 133  
RemoveFont, 164  
RemoveLast, 293  
RemovePen, 240  
RemoveView, 145  
RenderPage, 256  
Replace, 86, 293, 301, 306  
Replicate, 293  
ReportError, 250  
Requery, 274  
Request, 94  
Resource file, 21  
Resource format design issues, 419  
Reverse, 293, 294  
RevertValues, 170  
right, 197  
RightDClick, 219  
rightDown, 215  
RightDown, 219  
RightIsDown, 219  
RightOf, 191  
RightUp, 219  
RollbackTrans, 100

## —S—

SameAs, 191  
Save, 145  
SaveAs, 145  
SaveFile, 50, 306  
SaveObject, 145, 223  
ScreenToClient, 325  
Scroll, 64  
Searching/Matching, 365  
Select, 397  
SelectDocumentPath, 133  
SelectDocumentType, 134  
Selected, 205  
SelectObject, 206  
SelectViewType, 134  
Set, 78, 105, 170, 191, 205  
SetAllowSymbols, 161  
SetAppName, 47  
SetAutoLayout, 325  
SetBackground, 65, 117  
SetBackgroundColour, 193, 234  
SetBackgroundMode, 117  
SetBezelFace, 182  
SetBrush, 65, 117  
SetButtonColour, 193, 234  
SetButtonFont, 193, 235  
SetCanvas, 256  
SetCap, 238  
SetCheckpoint, 122  
SetCheckPrevious, 123  
SetChooseFull, 79

---

SetChosenFont, 162  
 SetClassName, 47  
 SetClientData, 85, 156, 205  
 SetClientSize, 326  
 SetClipboard, 213  
 SetClipboardClient, 75  
 SetClipboardString, 76  
 SetClippingRegion, 65, 117  
 SetCollate, 247  
 SetColour, 52, 53, 79, 162, 238  
 SetColourMap, 51, 117, 326  
 SetColumns, 72  
 SetCommandProcessor, 146  
 SetConstraints, 325  
 SetCurrentPage, 256  
 SetCursor, 327  
 SetDashes, 239  
 SetData, 221, 258, 260  
 SetDataSource, 100  
 SetDebugMode, 122  
 SetDefault, 55  
 SetDefaultExtension, 139  
 SetDefaultInfo, 227  
 SetDefaultMinMargins, 227  
 SetDefaultSize, 57  
 SetDefaultSQL, 275  
 SetDescription, 139  
 SetDeviceOrigin, 116  
 SetDirectory, 139  
 SetDirty, 260  
 SetDocument, 127, 318  
 SetDocumentManager, 140  
 SetDocumentName, 146  
 SetDocumentTemplate, 146  
 SetDoubleClick, 325  
 SetEditable, 170, 301, 306  
 SetEditMenu, 91  
 SetEventHandler, 327  
 SetExitOnDelete, 47  
 SetFieldDirty, 258, 274  
 SetFieldNull, 275  
 SetFile, 122  
 SetFileFilter, 140  
 SetFilename, 146  
 SetFirstItem, 205  
 SetFlags, 140  
 SetFocus, 325  
 SetFont, 65, 118, 306  
 SetFormat, 105  
 SetFrame, 256, 319  
 SetFromPage, 247  
 SetHelpString, 209, 211  
 SetHorizontalSpacing, 234, 235  
 SetIcon, 178  
 SetInitialFont, 162  
 SetInsertionPoint, 86, 301, 306  
 SetInsertionPointEnd, 86, 301, 306  
 SetJoin, 239  
 SetLabel, 55, 69, 193, 209, 211  
 SetLabelColour, 193, 235  
 SetLabelFont, 193, 235  
 SetLabelPosition, 234  
 SetLabelTop, 211  
 SetLevel, 122  
 SetLogicalFunction, 65, 118  
 SetLoginTimeout, 100  
 SetMapMode, 118  
 SetMarginBottomRight, 226  
 SetMargins, 312  
 SetMaxDocsOpen, 134  
 SetMaxPage, 247  
 SetMenuBar, 179  
 SetMinimumPaneSize, 283  
 SetMinMarginBottomRight, 226  
 SetMinMarginTopLeft, 226  
 SetMinPage, 247  
 SetModal, 126  
 SetName, 258, 325  
 SetNextHandler, 156  
 SetNoCopies, 247  
 SetNullable, 258  
 SetObjectLength, 277  
 SetOptimization, 119  
 SetOption, 105  
 SetOptions, 275  
 SetOrientation, 226  
 SetPageLength, 277  
 SetPaperSize, 226  
 SetPassword, 100  
 SetPen, 66, 119  
 SetPin, 148  
 SetPreviousHandler, 156  
 SetPrintMode, 48  
 SetPrintout, 256  
 SetPrintToFile, 248  
 SetQueryTimeout, 101  
 SetRange, 162, 182, 279  
 SetSashPosition, 283  
 SetScrollbars, 66  
 SetScrollPage, 67  
 SetScrollPos, 67  
 SetScrollRange, 67  
 SetSelection, 72, 86, 205, 262, 301, 304  
 SetSetupDialog, 248  
 SetShadowWidth, 182  
 SetShowHelp, 162  
 SetSize, 260, 325, 326  
 SetSizeHints, 326  
 SetSplitMode, 284  
 SetStandardError, 123  
 SetStatus, 148  
 SetStatusText, 179  
 SetStatusWidths, 179  
 SetStipple, 53, 239  
 SetStream, 123  
 SetString, 205  
 SetStringSelection, 72, 205, 262  
 SetStyle, 53, 239  
 SetSynchronousMode, 101  
 SetTableName, 275  
 SetTextBackground, 67, 119  
 SetTextForeground, 67, 120  
 SetTitle, 126, 146, 179, 209, 327  
 SetToolBar, 179

---

SetToolLongHelp, 312  
 SetToolPacking, 312  
 SetToolSeparation, 312  
 SetToolShortHelp, 312  
 SetToPage, 248  
 SetType, 259, 260, 275  
 Setup, 250  
 SetUserEditMode, 327  
 SetUsername, 101  
 SetUserScale, 120  
 SetValue, 69, 86, 182, 264, 277, 280, 301  
 SetVerticalSpacing, 235  
 SetView, 127  
 SetViewLength, 277  
 SetViewName, 319  
 SetWidth, 239  
 SetZoom, 256  
 SetZoomControl, 244  
 shiftDown, 195, 215  
 ShiftDown, 196, 219  
 Show, 81, 126, 148, 163, 228, 249, 263, 327  
 ShowPosition, 220, 306  
 Simplify the problem, 42  
 Sort, 201, 298  
 Split, 297  
 SplitHorizontally, 284  
 SplitVertically, 285  
 sprintf, 294  
 Start, 308  
 StartAdvise, 94  
 StartDoc, 120  
 StartPage, 120  
 Status, 365  
 StatusLineExists, 180  
 Stop, 308  
 StopAdvise, 94  
 Strip, 294  
 Submit, 91  
 Substitution, 365  
 SubString, 294  
 Substring extraction examples, 407  
 Syntax of Regular Expressions, 409

## —T—

Tab, 235  
 tDefaultExt, 136  
 tDescription, 136  
 tDirectory, 136  
 tDocClassInfo, 136  
 tDocTypeName, 136  
 tDocumentManager, 136  
 Templates, 24  
 Test program, 403  
 tFileFilter, 136  
 tFlags, 136  
 The format of a .WXR file, 414  
 The IPC demo, 40  
 The MDI demo, 39  
 The purpose of the form class, 166  
 The wxButtonBar library, 392  
 The wxString class, 404

The wxWindows event system, 11  
 Through, 294  
 ToggleTool, 313  
 top, 197  
 Transformations, 365  
 tViewClassInfo, 137  
 tViewTypeName, 137

## —U—

Unconstrained, 190  
 Undo, 87, 91  
 UNIX makefiles, 21  
 Unsplit, 285  
 Uppcase, 294  
 Update, 275, 397  
 UpdateValues, 170  
 UpperCase, 294  
 Use a debugger, 42  
 Use ASSERT, 41  
 Use relative positioning or constraints, 41  
 Use tracing code, 42  
 Use wxObject::Dump and the wxDebugContext class, 43  
 Use wxString in preference to character arrays, 41  
 Use wxWindows resource files, 41  
 userPanel, 147  
 Using the toolbar library, 392  
 Utilities, 365

## —V—

ValidHost, 75  
 viewDocument, 316  
 viewFrame, 316  
 ViewStart, 67  
 viewTypeName, 316

## —W—

WarpPointer, 68  
 What wxWindows has, 15  
 width, 197  
 Window layout examples, 389  
 Windows 95 differences, 393  
 Windows and NT features, 19  
 Windows makefiles, 20  
 work\_proc, 45  
 WriteEvent, 150  
 WriteText, 307  
 wx\_class, 44  
 wxAddPrimaryEventHandler, 337  
 wxAddSecondaryEventHandler, 337  
 wxALWAYS\_SB, 368  
 wxApp, 44  
 wxApp::~wxApp, 44  
 wxApp::argc, 44  
 wxApp::argv, 44  
 wxApp::Dispatch, 45  
 wxApp::ExitMainLoop, 46  
 wxApp::GetAppName, 45

wxApp::GetClassName, 45  
wxApp::GetExitOnDelete, 45  
wxApp::GetPrintMode, 45  
wxApp::GetTopWindow, 45  
wxApp::Initialized, 46  
wxApp::MainLoop, 46  
wxApp::OnCharHook, 46  
wxApp::OnExit, 46  
wxApp::OnInit, 46  
wxApp::Pending, 47  
wxApp::ProcessMessage, 47  
wxApp::SetAppName, 47  
wxApp::SetClassName, 47  
wxApp::SetExitOnDelete, 47  
wxApp::SetPrintMode, 48  
wxApp::work\_proc, 45  
wxApp::wx\_class, 44  
wxApp::wxApp, 44  
wxBeginBusyCursor, 343  
wxBell, 343  
wxBitmap, 48  
wxBitmap::~~wxBitmap, 49  
wxBitmap::Create, 49  
wxBitmap::GetColourMap, 49  
wxBitmap::GetDepth, 49  
wxBitmap::GetHeight, 49  
wxBitmap::GetWidth, 50  
wxBitmap::LoadFile, 50  
wxBitmap::Ok, 50  
wxBitmap::SaveFile, 50  
wxBitmap::SetColourMap, 51  
wxBitmap::wxBitmap, 48  
wxBITMAP\_TYPE\_BMP, 49, 50, 184  
wxBITMAP\_TYPE\_BMP\_RESOURCE, 49, 50  
wxBITMAP\_TYPE\_CUR, 95  
wxBITMAP\_TYPE\_CUR\_RESOURCE, 95  
wxBITMAP\_TYPE\_GIF, 49, 50, 184  
wxBITMAP\_TYPE\_ICO, 95, 184  
wxBITMAP\_TYPE\_ICO\_RESOURCE, 184  
wxBITMAP\_TYPE\_RESOURCE, 49, 50  
wxBITMAP\_TYPE\_XBM, 49, 50, 51, 95, 184  
wxBITMAP\_TYPE\_XPM, 49, 50, 51, 184  
wxBORDER, 370  
wxBrush, 51, 52  
wxBrush::~~wxBrush, 52  
wxBrush::GetColour, 52  
wxBrush::GetStipple, 52  
wxBrush::GetStyle, 52  
wxBrush::SetColour, 52  
wxBrush::SetStipple, 53  
wxBrush::SetStyle, 53  
wxBrush::wxBrush, 51  
wxBrushList, 53  
wxBrushList::AddBrush, 53  
wxBrushList::FindOrCreateBrush, 54  
wxBrushList::RemoveBrush, 54  
wxBrushList::wxBrushList, 53  
wxButton, 54  
wxButton styles, 368  
wxButton::~~wxButton, 55  
wxButton::Create, 55  
wxButton::SetDefault, 55  
wxButton::SetLabel, 55  
wxButton::wxButton, 54  
wxButtonBar, 56  
wxButtonBar::GetDefaultButtonHeight, 56  
wxButtonBar::GetDefaultButtonWidth, 56  
wxButtonBar::SetDefaultSize, 56  
wxButtonBar::wxButtonBar, 56  
wxCanvas, 57  
wxCanvas styles, 370  
wxCanvas::~~wxCanvas, 57  
wxCanvas::AllowDoubleClick, 57  
wxCanvas::BeginDrawing, 58  
wxCanvas::Clear, 58  
wxCanvas::Create, 58  
wxCanvas::CrossHair, 58  
wxCanvas::DestroyClippingRegion, 58  
wxCanvas::DrawArc, 58  
wxCanvas::DrawEllipse, 59  
wxCanvas::DrawLine, 59  
wxCanvas::DrawLines, 59  
wxCanvas::DrawPoint, 59  
wxCanvas::DrawPolygon, 59  
wxCanvas::DrawRectangle, 60  
wxCanvas::DrawRoundedRectangle, 60  
wxCanvas::DrawSpline, 60  
wxCanvas::DrawText, 60  
wxCanvas::EnableScrolling, 60  
wxCanvas::EndDrawing, 60  
wxCanvas::FloodFill, 61  
wxCanvas::GetClippingBox, 113  
wxCanvas::GetDC, 61  
wxCanvas::GetScrollPage, 61  
wxCanvas::GetScrollPixelsPerUnit, 61  
wxCanvas::GetScrollPos, 61  
wxCanvas::GetScrollRange, 61  
wxCanvas::GetScrollUnitsPerPage, 62  
wxCanvas::GetVirtualSize, 62  
wxCanvas::IntDrawLine, 62  
wxCanvas::IntDrawLines, 62  
wxCanvas::IsRetained, 62  
wxCanvas::OnChar, 62  
wxCanvas::OnEvent, 63  
wxCanvas::OnPaint, 63  
wxCanvas::OnScroll, 64  
wxCanvas::Scroll, 64  
wxCanvas::SetBackground, 64  
wxCanvas::SetBrush, 65  
wxCanvas::SetClippingRegion, 65  
wxCanvas::SetFont, 65  
wxCanvas::SetLogicalFunction, 65  
wxCanvas::SetPen, 66  
wxCanvas::SetScrollbars, 66  
wxCanvas::SetScrollPage, 67  
wxCanvas::SetScrollPos, 67  
wxCanvas::SetScrollRange, 67  
wxCanvas::SetTextBackground, 67  
wxCanvas::SetTextForeground, 67  
wxCanvas::ViewStart, 67  
wxCanvas::WarpPointer, 68  
wxCanvas::wxCanvas, 57  
wxCanvasDC, 68  
wxCanvasDC::GetClippingBox, 68

---

wxCanvasDC::wxCanvasDC, 68  
 wxCAPTION, 367  
 wxCB\_DROPDOWN, 368  
 wxCB\_READONLY, 368  
 wxCB\_SIMPLE, 368  
 wxCB\_SORT, 368  
 wxCHARARG, 295  
 wxCheckBox, 68  
 wxCheckBox::~~wxCheckBox, 69  
 wxCheckBox::Create, 69  
 wxCheckBox::GetValue, 69  
 wxCheckBox::SetLabel, 69  
 wxCheckBox::SetValue, 69  
 wxCheckBox::wxCheckBox, 68  
 wxChoice, 70  
 wxChoice::~~wxChoice, 70  
 wxChoice::Append, 71  
 wxChoice::Clear, 71  
 wxChoice::Create, 71  
 wxChoice::FindString, 71  
 wxChoice::GetColumns, 71  
 wxChoice::GetSelection, 71  
 wxChoice::GetString, 71, 72  
 wxChoice::GetStringSelection, 72  
 wxChoice::Number, 72  
 wxChoice::SetColumns, 72  
 wxChoice::SetSelection, 72  
 wxChoice::SetStringSelection, 72  
 wxChoice::wxChoice, 70  
 wxClassInfo, 73, 371  
 wxClassInfo::CreateObject, 73  
 wxClassInfo::FindClass, 73  
 wxClassInfo::GetBaseClassName1, 73  
 wxClassInfo::GetBaseClassName2, 73  
 wxClassInfo::GetClassName, 73  
 wxClassInfo::GetSize, 73  
 wxClassInfo::InitializeClasses, 74  
 wxClassInfo::IsKindOf, 74  
 wxClassInfo::wxClassInfo, 73  
 wxCleanup, 343  
 wxClient, 74  
 wxClient::MakeConnection, 74  
 wxClient::OnMakeConnection, 74  
 wxClient::ValidHost, 75  
 wxClient::wxClient, 74  
 wxClipboard::GetClipboardClient, 75  
 wxClipboard::GetClipboardData, 75  
 wxClipboard::GetClipboardString, 75  
 wxClipboard::SetClipboardClient, 75  
 wxClipboard::SetClipboardString, 76  
 wxClipboardClient::BeingReplaced, 76  
 wxClipboardClient::formats, 76  
 wxClipboardClient::GetData, 76  
 wxClipboardOpen, 341  
 wxCloseClipboard, 341  
 wxColour, 77  
 wxColour::Blue, 77  
 wxColour::Get, 77  
 wxColour::Green, 77  
 wxColour::Ok, 77  
 wxColour::operator =, 77  
 wxColour::Red, 77  
 wxColour::Set, 77  
 wxColour::wxColour, 77  
 wxColourData, 78  
 wxColourData::~~wxColourData, 78  
 wxColourData::GetChooseFull, 78  
 wxColourData::GetColour, 78  
 wxColourData::GetCustomColour, 78  
 wxColourData::operator =, 79  
 wxColourData::SetChooseFull, 79  
 wxColourData::SetColour, 79  
 wxColourData::SetCustomColour, 79  
 wxColourData::wxColourData, 78  
 wxColourDatabase, 80  
 wxColourDatabase::FindColour, 80  
 wxColourDatabase::FindName, 80  
 wxColourDatabase::Initialize, 80  
 wxColourDatabase::wxColourDatabase, 80  
 wxColourDialog, 80  
 wxColourDialog overview, 386  
 wxColourDialog::~~wxColourDialog, 80  
 wxColourDialog::GetColourData, 81  
 wxColourDialog::Show, 81  
 wxColourDialog::wxColourDialog, 80  
 wxColourDisplay, 336  
 wxColourMap, 81  
 wxColourMap::~~wxColourMap, 81  
 wxColourMap::Create, 81  
 wxColourMap::wxColourMap, 81  
 wxComboBox, 82, 368  
 wxComboBox::~~wxComboBox, 83  
 wxComboBox::Append, 83  
 wxComboBox::Clear, 83  
 wxComboBox::Copy, 83  
 wxComboBox::Create, 83  
 wxComboBox::Cut, 84  
 wxComboBox::Delete, 84  
 wxComboBox::Deselect, 84  
 wxComboBox::FindString, 84  
 wxComboBox::GetClientData, 84  
 wxComboBox::GetInsertionPoint, 84  
 wxComboBox::GetLastPosition, 84  
 wxComboBox::GetSelection, 84  
 wxComboBox::GetString, 85  
 wxComboBox::GetStringSelection, 85  
 wxComboBox::GetValue, 85  
 wxComboBox::Number, 85  
 wxComboBox::Paste, 85  
 wxComboBox::Remove, 85  
 wxComboBox::Replace, 86  
 wxComboBox::SetClientData, 85  
 wxComboBox::SetInsertionPoint, 86  
 wxComboBox::SetInsertionPointEnd, 86  
 wxComboBox::SetSelection, 86  
 wxComboBox::SetValue, 86  
 wxComboBox::wxComboBox, 82  
 wxCommand, 86  
 wxCommand overview, 375  
 wxCommand::~~wxCommand, 87  
 wxCommand::CanUndo, 87  
 wxCommand::Do, 87  
 wxCommand::GetName, 87  
 wxCommand::Undo, 87

---

- wxCommand::wxCommand, 86
- wxCommandEvent, 88
- wxCommandEvent::Checked, 89
- wxCommandEvent::clientData, 88
- wxCommandEvent::commandInt, 88
- wxCommandEvent::commandString, 88
- wxCommandEvent::extraLong, 88
- wxCommandEvent::GetClientData, 89
- wxCommandEvent::GetSelection, 89
- wxCommandEvent::GetString, 89
- wxCommandEvent::IsSelection, 89
- wxCommandEvent::wxCommandEvent, 88
- wxCommandProcessor, 90
- wxCommandProcessor overview, 376
- wxCommandProcessor::~~wxCommandProcessor, 90
- wxCommandProcessor::CanUndo, 90
- wxCommandProcessor::ClearCommands, 90
- wxCommandProcessor::Do, 90
- wxCommandProcessor::GetCommands, 90
- wxCommandProcessor::GetEditMenu, 90
- wxCommandProcessor::GetMaxCommands, 90
- wxCommandProcessor::Initialize, 91
- wxCommandProcessor::SetEditMenu, 91
- wxCommandProcessor::Submit, 91
- wxCommandProcessor::Undo, 91
- wxCommandProcessor::wxCommandProcessor, 90
- wxConcatFiles, 329
- wxConnection, 92
- wxConnection::Advise, 92
- wxConnection::Disconnect, 92
- wxConnection::Execute, 92
- wxConnection::OnAdvise, 92
- wxConnection::OnDisconnect, 93
- wxConnection::OnExecute, 93
- wxConnection::OnPoke, 93
- wxConnection::OnRequest, 93
- wxConnection::OnStartAdvise, 93
- wxConnection::OnStopAdvise, 93
- wxConnection::Poke, 93
- wxConnection::Request, 94
- wxConnection::StartAdvise, 94
- wxConnection::StopAdvise, 94
- wxConnection::wxConnection, 92
- wxConstraintFunction, 172
- wxCopyFile, 329
- wxCreateDynamicObject, 343
- wxCursor, 94, 95
- wxCursor::~~wxCursor, 96
- wxCursor::wxCursor, 94
- wxDatabase, 96
- wxDatabase overview, 394
- wxDatabase::~~wxDatabase, 96
- wxDatabase::BeginTrans, 96
- wxDatabase::Cancel, 96
- wxDatabase::CanTransact, 97
- wxDatabase::CanUpdate, 97
- wxDatabase::Close, 97
- wxDatabase::CommitTrans, 97
- wxDatabase::ErrorOccured, 97
- wxDatabase::ErrorSnapshot, 97
- wxDatabase::GetDatabaseName, 97
- wxDatabase::GetDataSource, 97
- wxDatabase::GetErrorClass, 98
- wxDatabase::GetErrorCode, 98
- wxDatabase::GetErrorMessage, 98
- wxDatabase::GetErrorNumber, 98
- wxDatabase::GetHDBC, 98
- wxDatabase::GetHENV, 98
- wxDatabase::GetInfo, 98
- wxDatabase::GetODBCVersionFloat, 99
- wxDatabase::GetODBCVersionString, 99
- wxDatabase::GetPassword, 99
- wxDatabase::GetUsername, 99
- wxDatabase::InWaitForDataSource, 99
- wxDatabase::IsOpen, 100
- wxDatabase::OnSetOptions, 100
- wxDatabase::OnWaitForDataSource, 100
- wxDatabase::Open, 100
- wxDatabase::RollbackTrans, 100
- wxDatabase::SetDataSource, 100
- wxDatabase::SetLoginTimeout, 100
- wxDatabase::SetPassword, 100
- wxDatabase::SetQueryTimeout, 101
- wxDatabase::SetSynchronousMode, 101
- wxDatabase::SetUsername, 101
- wxDatabase::wxDatabase, 96
- wxDate, 101
- wxDate::~~wxDate, 102
- wxDate::AddMonths, 102
- wxDate::AddWeeks, 102
- wxDate::AddYears, 102
- wxDate::FormatDate, 102
- wxDate::GetDay, 102
- wxDate::GetDayOfWeek, 103
- wxDate::GetDayOfWeekName, 103
- wxDate::GetDayOfYear, 103
- wxDate::GetDaysInMonth, 103
- wxDate::GetFirstDayOfMonth, 103
- wxDate::GetJulianDate, 103
- wxDate::GetMonth, 103
- wxDate::GetMonthEnd, 104
- wxDate::GetMonthName, 104
- wxDate::GetMonthStart, 104
- wxDate::GetWeekOfMonth, 104
- wxDate::GetWeekOfYear, 104
- wxDate::GetYear, 104
- wxDate::GetYearEnd, 104
- wxDate::GetYearStart, 104
- wxDate::IsLeapYear, 105
- wxDate::operator -, 106
- wxDate::operator --, 106
- wxDate::operator !=, 107
- wxDate::operator +, 106
- wxDate::operator ++, 106
- wxDate::operator +=, 106
- wxDate::operator <, 107
- wxDate::operator <<, 107
- wxDate::operator <=, 107
- wxDate::operator -=, 106
- wxDate::operator ==, 107
- wxDate::operator >, 107
- wxDate::operator >=, 107

wxDate::operator char \*, 106  
wxDate::Set, 105  
wxDate::SetFormat, 105  
wxDate::SetOption, 105  
wxDate::wxDate, 101  
wxDC, 108  
wxDC::~~wxDC, 108  
wxDC::BeginDrawing, 108  
wxDC::Blit, 108  
wxDC::Clear, 109  
wxDC::CrossHair, 109  
wxDC::DestroyClippingRegion, 109  
wxDC::DeviceToLogicalX, 109  
wxDC::DeviceToLogicalXRel, 109  
wxDC::DeviceToLogicalY, 109  
wxDC::DeviceToLogicalYRel, 109  
wxDC::DrawArc, 110  
wxDC::DrawEllipse, 110  
wxDC::DrawEllipticArc, 110  
wxDC::DrawIcon, 110  
wxDC::DrawLine, 110  
wxDC::DrawLines, 110  
wxDC::DrawPoint, 111  
wxDC::DrawPolygon, 111  
wxDC::DrawRectangle, 111  
wxDC::DrawRoundedRectangle, 111  
wxDC::DrawSpline, 112  
wxDC::DrawText, 112  
wxDC::EndDoc, 112  
wxDC::EndDrawing, 112  
wxDC::EndPage, 112  
wxDC::FloodFill, 112  
wxDC::GetBackground, 113  
wxDC::GetBrush, 113  
wxDC::GetCharHeight, 113  
wxDC::GetCharWidth, 113  
wxDC::GetFont, 113  
wxDC::GetLogicalFunction, 113  
wxDC::GetMapMode, 114  
wxDC::GetOptimization, 114  
wxDC::GetPen, 114  
wxDC::GetPixel, 114  
wxDC::GetSize, 114  
wxDC::GetTextBackground, 114  
wxDC::GetTextExtent, 115  
wxDC::GetTextForeground, 115  
wxDC::IntDrawLine, 115  
wxDC::IntDrawLines, 115  
wxDC::LogicalToDeviceX, 115  
wxDC::LogicalToDeviceXRel, 115  
wxDC::LogicalToDeviceY, 116  
wxDC::LogicalToDeviceYRel, 116  
wxDC::MaxX, 116  
wxDC::MaxY, 116  
wxDC::MinX, 116  
wxDC::MinY, 116  
wxDC::Ok, 116  
wxDC::SetBackground, 117  
wxDC::SetBackgroundMode, 117  
wxDC::SetBrush, 117  
wxDC::SetClippingRegion, 117  
wxDC::SetColourMap, 117  
wxDC::SetDeviceOrigin, 116  
wxDC::SetFont, 118  
wxDC::SetLogicalFunction, 118  
wxDC::SetMapMode, 118  
wxDC::SetOptimization, 119  
wxDC::SetPen, 119  
wxDC::SetTextBackground, 119  
wxDC::SetTextForeground, 119  
wxDC::SetUserScale, 120  
wxDC::StartDoc, 120  
wxDC::StartPage, 120  
wxDC::wxDC, 108  
WXDEBUG\_NEW, 352  
wxDebugContext overview, 398  
wxDebugContext::Check, 120  
wxDebugContext::Dump, 120  
wxDebugContext::GetCheckPrevious, 121  
wxDebugContext::GetDebugMode, 121  
wxDebugContext::GetLevel, 121  
wxDebugContext::GetStream, 121  
wxDebugContext::GetStreamBuf, 121  
wxDebugContext::HasStream, 121  
wxDebugContext::PrintClasses, 121  
wxDebugContext::PrintStatistics, 122  
wxDebugContext::SetCheckpoint, 122  
wxDebugContext::SetCheckPrevious, 122  
wxDebugContext::SetDebugMode, 122  
wxDebugContext::SetFile, 122  
wxDebugContext::SetLevel, 122  
wxDebugContext::SetStandardError, 123  
wxDebugContext::SetStream, 123  
wxDebugMsg, 343  
wxDEFAULT\_DIALOG\_STYLE, 367  
wxDEFAULT\_FRAME, 367  
wxDialogBox, 124  
wxDialogBox styles, 367  
wxDialogBox::~~wxDialogBox, 124  
wxDialogBox::Centre, 124  
wxDialogBox::Create, 124  
wxDialogBox::GetTitle, 125  
wxDialogBox::Iconize, 125  
wxDialogBox::Iconized, 125  
wxDialogBox::IsModal, 125  
wxDialogBox::OnCharHook, 125  
wxDialogBox::SetModal, 125  
wxDialogBox::SetTitle, 126  
wxDialogBox::Show, 126  
wxDialogBox::wxDialogBox, 123  
wxDirExists, 328  
wxDisplayDepth, 336  
wxDisplaySize, 344  
wxDocChildFrame, 126  
wxDocChildFrame::~~wxDocChildFrame, 127  
wxDocChildFrame::childDocument, 126  
wxDocChildFrame::childView, 126  
wxDocChildFrame::GetDocument, 127  
wxDocChildFrame::GetView, 127  
wxDocChildFrame::OnActivate, 127  
wxDocChildFrame::OnClose, 127  
wxDocChildFrame::OnMenuCommand, 127  
wxDocChildFrame::SetDocument, 127  
wxDocChildFrame::SetView, 127



- 
- wxDocChildFrame::wxDocChildFrame, 126
  - wxDocManager, 129
  - wxDocManager overview, 375
  - wxDocManager::~wxDocManager, 129
  - wxDocManager::ActivateView, 129
  - wxDocManager::AddDocument, 129
  - wxDocManager::AddFileToHistory, 129
  - wxDocManager::AssociateTemplate, 129
  - wxDocManager::CreateDocument, 130
  - wxDocManager::CreateView, 130
  - wxDocManager::currentView, 128
  - wxDocManager::defaultDocumentNameCounter, 128
  - wxDocManager::DisassociateTemplate, 130
  - wxDocManager::fileHistory, 128
  - wxDocManager::FileHistoryLoad, 130
  - wxDocManager::FileHistorySave, 130
  - wxDocManager::FileHistoryUseMenu, 131
  - wxDocManager::FindTemplateForPath, 131
  - wxDocManager::GetCurrentDocument, 131
  - wxDocManager::GetCurrentView, 131
  - wxDocManager::GetDocuments, 131
  - wxDocManager::GetFileHistory, 131
  - wxDocManager::GetMaxDocsOpen, 131
  - wxDocManager::GetNoHistoryFiles, 131
  - wxDocManager::Initialize, 132
  - wxDocManager::MakeDefaultName, 132
  - wxDocManager::maxDocsOpen, 128
  - wxDocManager::mnDocs, 128
  - wxDocManager::mnFlags, 128
  - wxDocManager::mnTemplates, 129
  - wxDocManager::OnCreateFileHistory, 132
  - wxDocManager::OnFileClose, 132
  - wxDocManager::OnFileNew, 132
  - wxDocManager::OnFileOpen, 132
  - wxDocManager::OnFileSave, 133
  - wxDocManager::OnFileSaveAs, 133
  - wxDocManager::OnMenuCommand, 133
  - wxDocManager::RemoveDocument, 133
  - wxDocManager::SelectDocumentPath, 133
  - wxDocManager::SelectDocumentType, 134
  - wxDocManager::SelectViewType, 134
  - wxDocManager::SetMaxDocsOpen, 134
  - wxDocManager::wxDocManager, 129
  - wxDocParentFrame, 134
  - wxDocParentFrame::~wxDocParentFrame, 134
  - wxDocParentFrame::OnClose, 135
  - wxDocParentFrame::OnMenuCommand, 135
  - wxDocParentFrame::wxDocParentFrame, 134
  - wxDocTemplate, 137
  - wxDocTemplate overview, 374
  - wxDocTemplate::~wxDocTemplate, 138
  - wxDocTemplate::CreateDocument, 138
  - wxDocTemplate::CreateView, 138
  - wxDocTemplate::GetDefaultExtension, 138
  - wxDocTemplate::GetDescription, 138
  - wxDocTemplate::GetDirectory, 138
  - wxDocTemplate::GetDocumentManager, 138
  - wxDocTemplate::GetDocumentName, 139
  - wxDocTemplate::GetFileFilter, 139
  - wxDocTemplate::GetFlags, 139
  - wxDocTemplate::GetViewName, 139
  - wxDocTemplate::IsVisible, 139
  - wxDocTemplate::SetDefaultExtension, 139
  - wxDocTemplate::SetDescription, 139
  - wxDocTemplate::SetDirectory, 139
  - wxDocTemplate::SetDocumentManager, 140
  - wxDocTemplate::SetFileFilter, 140
  - wxDocTemplate::SetFlags, 140
  - wxDocTemplate::tDefaultExt, 135
  - wxDocTemplate::tDescription, 136
  - wxDocTemplate::tDirectory, 136
  - wxDocTemplate::tDocClassInfo, 136
  - wxDocTemplate::tDocTypeName, 136
  - wxDocTemplate::tDocumentManager, 136
  - wxDocTemplate::tFileFilter, 136
  - wxDocTemplate::tFlags, 136
  - wxDocTemplate::tViewClassInfo, 137
  - wxDocTemplate::tViewTypeName, 137
  - wxDocTemplate::wxDocTemplate, 137
  - wxDocument, 141
  - wxDocument overview, 373
  - wxDocument::~wxDocDocument, 141
  - wxDocument::AddView, 141
  - wxDocument::Close, 141
  - wxDocument::DeleteAllViews, 142
  - wxDocument::documentFile, 140
  - wxDocument::documentModified, 140
  - wxDocument::documentTemplate, 140
  - wxDocument::documentTitle, 141
  - wxDocument::documentTypeName, 141
  - wxDocument::documentViews, 141
  - wxDocument::GetCommandProcessor, 142
  - wxDocument::GetDocumentManager, 142
  - wxDocument::GetDocumentName, 142
  - wxDocument::GetDocumentTemplate, 142
  - wxDocument::GetDocumentWindow, 142
  - wxDocument::GetFilename, 142
  - wxDocument::GetFirstView, 143
  - wxDocument::GetPrintableName, 143
  - wxDocument::GetTitle, 143
  - wxDocument::IsModified, 143
  - wxDocument::LoadObject, 143
  - wxDocument::Modify, 143
  - wxDocument::OnChangedViewList, 144
  - wxDocument::OnCloseDocument, 144
  - wxDocument::OnCreate, 144
  - wxDocument::OnCreateCommandProcessor, 144
  - wxDocument::OnNewDocument, 144
  - wxDocument::OnOpenDocument, 144
  - wxDocument::OnSaveDocument, 145
  - wxDocument::OnSaveModified, 145
  - wxDocument::RemoveView, 145
  - wxDocument::Save, 145
  - wxDocument::SaveAs, 145
  - wxDocument::SaveObject, 145
  - wxDocument::SetCommandProcessor, 145
  - wxDocument::SetDocumentName, 146
  - wxDocument::SetDocumentTemplate, 146
  - wxDocument::SetFilename, 146
  - wxDocument::SetTitle, 146
  - wxDocument::wxDocDocument, 141
  - wxEdge, 189
-

- wxEmptyClipboard, 341
- wxEndBusyCursor, 344
- wxEnhDialogBox, 147
- wxEnhDialogBox::~wxEnhDialogBox, 147
- wxEnhDialogBox::AddCmd, 148
- wxEnhDialogBox::Fit, 148
- wxEnhDialogBox::GetCmd, 148
- wxEnhDialogBox::SetPin, 148
- wxEnhDialogBox::SetStatus, 148
- wxEnhDialogBox::Show, 148
- wxEnhDialogBox::userPanel, 147
- wxEnhDialogBox::wxEnhDialogBox, 147
- wxEntry, 344
- wxEnumClipboardFormats, 341
- wxError, 344
- wxEvent, 149
- wxEvent::~wxEvent, 149
- wxEvent::eventClass, 149
- wxEvent::eventHandle, 149
- wxEvent::eventObject, 149
- wxEvent::eventType, 149
- wxEvent::GetEventClass, 149
- wxEvent::GetEventObject, 150
- wxEvent::GetEventType, 150
- wxEvent::GetObject, 150
- wxEvent::ReadEvent, 150
- wxEvent::WriteEvent, 150
- wxEvent::wxEvent, 149
- wxEvtHandler, 151
- wxEvtHandler::~wxEvtHandler, 151
- wxEvtHandler::GetClientData, 151
- wxEvtHandler::GetNextHandler, 151
- wxEvtHandler::GetPreviousHandler, 151
- wxEvtHandler::nextHandler, 150
- wxEvtHandler::OnActivate, 151
- wxEvtHandler::OnChar, 151
- wxEvtHandler::OnCharHook, 152
- wxEvtHandler::OnClose, 152
- wxEvtHandler::OnCommand, 152
- wxEvtHandler::OnDefaultAction, 152
- wxEvtHandler::OnDropFiles, 152
- wxEvtHandler::OnEvent, 153
- wxEvtHandler::OnItemEvent, 153
- wxEvtHandler::OnItemLeftClick, 153
- wxEvtHandler::OnItemMove, 153
- wxEvtHandler::OnItemRightClick, 153
- wxEvtHandler::OnItemSelect, 154
- wxEvtHandler::OnItemSize, 154
- wxEvtHandler::OnKillFocus, 154
- wxEvtHandler::OnLeftClick, 154
- wxEvtHandler::OnMenuCommand, 154
- wxEvtHandler::OnMenuSelect, 155
- wxEvtHandler::OnMove, 155
- wxEvtHandler::OnPaint, 155
- wxEvtHandler::OnRightClick, 154
- wxEvtHandler::OnScroll, 155
- wxEvtHandler::OnSelect, 155
- wxEvtHandler::OnSetFocus, 155
- wxEvtHandler::OnSize, 155
- wxEvtHandler::previousHandler, 151
- wxEvtHandler::SetClientData, 156
- wxEvtHandler::SetNextHandler, 156
- wxEvtHandler::SetPreviousHandler, 156
- wxEvtHandler::wxEvtHandler, 151
- wxExecute, 345
- wxExit, 345
- wxFatalError, 345
- wxFileExists, 328
- wxFileHistory, 157
- wxFileHistory overview, 376
- wxFileHistory::~wxFileHistory, 157
- wxFileHistory::AddFileToHistory, 157
- wxFileHistory::fileHistory, 156
- wxFileHistory::FileHistoryLoad, 157
- wxFileHistory::fileHistoryN, 156
- wxFileHistory::FileHistorySave, 157
- wxFileHistory::FileHistoryUseMenu, 157
- wxFileHistory::fileMaxFiles, 156
- wxFileHistory::fileMenu, 157
- wxFileHistory::GetMaxFiles, 158
- wxFileHistory::GetNoHistoryFiles, 158
- wxFileHistory::wxFileHistory, 157
- wxFileNameFromPath, 328
- wxFileSelector, 333
- wxFindFirstFile, 328, 329
- wxFindMenuItemId, 345
- wxFindWindowByLabel, 345
- wxFindWindowByName, 346
- wxFIXED\_LENGTH, 368
- wxFont, 158
- wxFont::~wxFont, 159
- wxFont::GetFaceName, 159
- wxFont::GetFamily, 159
- wxFont::GetFontId, 159
- wxFont::GetPointSize, 159
- wxFont::GetStyle, 159
- wxFont::GetUnderlined, 160
- wxFont::GetWeight, 160
- wxFont::wxFont, 158
- wxFontData, 160
- wxFontData::~wxFontData, 160
- wxFontData::EnableEffects, 160
- wxFontData::GetAllowSymbols, 160
- wxFontData::GetChosenFont, 161
- wxFontData::GetColour, 161
- wxFontData::GetEnableEffects, 161
- wxFontData::GetInitialFont, 161
- wxFontData::GetShowHelp, 161
- wxFontData::operator =, 162
- wxFontData::SetAllowSymbols, 161
- wxFontData::SetChosenFont, 162
- wxFontData::SetColour, 162
- wxFontData::SetInitialFont, 162
- wxFontData::SetRange, 162
- wxFontData::SetShowHelp, 162
- wxFontData::wxFontData, 160
- wxFontDialog, 163
- wxFontDialog overview, 387
- wxFontDialog::~wxFontDialog, 163
- wxFontDialog::GetFontData, 163
- wxFontDialog::Show, 163
- wxFontDialog::wxFontDialog, 163
- wxFontList, 163
- wxFontList::AddFont, 164

wxFontList::FindOrCreateFont, 164  
 wxFontList::RemoveFont, 164  
 wxFontList::wxFontList, 163  
 wxFontNameDirectory, 164  
 wxFontNameDirectory::~wxFontNameDirectory, 164  
 wxFontNameDirectory::FindOrCreateFontId, 164  
 wxFontNameDirectory::GetAFMName, 165  
 wxFontNameDirectory::GetFamily, 165  
 wxFontNameDirectory::GetFontId, 165  
 wxFontNameDirectory::GetFontName, 165  
 wxFontNameDirectory::GetNewFontId, 165  
 wxFontNameDirectory::GetPostScriptName, 165  
 wxFontNameDirectory::GetScreenName, 165  
 wxFontNameDirectory::Initialize, 165  
 wxFontNameDirectory::wxFontNameDirectory, 164  
 wxForm, 167  
 wxForm::~wxForm, 168  
 wxForm::Add, 168  
 wxForm::AssociatePanel, 168  
 wxForm::Delete, 168  
 wxForm::FindItem, 168  
 wxForm::IsEditable, 169  
 wxForm::OnCancel, 169  
 wxForm::OnHelp, 169  
 wxForm::OnOk, 169  
 wxForm::OnRevert, 169  
 wxForm::OnUpdate, 169  
 wxForm::RevertValues, 169  
 wxForm::Set, 170  
 wxForm::SetEditable, 170  
 wxForm::UpdateValues, 170  
 wxForm::wxForm, 167  
 wxFormItem::GetPanellItem, 172  
 wxFrame, 172  
 wxFrame styles, 367  
 wxFrame::~wxFrame, 173  
 wxFrame::Centre, 173  
 wxFrame::Command, 174  
 wxFrame::Create, 174  
 wxFrame::CreateStatusLine, 174  
 wxFrame::Fit, 174  
 wxFrame::GetMenuBar, 174  
 wxFrame::GetTitle, 174  
 wxFrame::GetToolBar, 174  
 wxFrame::Iconize, 175  
 wxFrame::Iconized, 175  
 wxFrame::LoadAccelerators, 175  
 wxFrame::Maximize, 175  
 wxFrame::OnActivate, 175  
 wxFrame::OnClose, 176  
 wxFrame::OnMenuCommand, 176  
 wxFrame::OnMenuSelect, 177  
 wxFrame::OnSize, 178  
 wxFrame::SetIcon, 178  
 wxFrame::SetMenuBar, 178  
 wxFrame::SetStatusText, 179  
 wxFrame::SetStatusWidths, 179  
 wxFrame::SetTitle, 179  
 wxFrame::SetToolBar, 179  
 wxFrame::StatusLineExists, 180  
 wxFrame::wxFrame, 172  
 wxFunction, 180  
 wxGA\_HORIZONTAL, 368  
 wxGA\_PROGRESSBAR, 368  
 wxGA\_VERTICAL, 368  
 wxGauge, 180  
 wxGauge styles, 368  
 wxGauge::~wxGauge, 181  
 wxGauge::Create, 181  
 wxGauge::GetBezelFace, 181  
 wxGauge::GetRange, 181  
 wxGauge::GetValue, 181  
 wxGauge::SetBezelFace, 182  
 wxGauge::SetRange, 182  
 wxGauge::SetShadowWidth, 182  
 wxGauge::SetValue, 182  
 wxGauge::wxGauge, 180  
 wxGetActiveWindow, 346  
 wxGetClipboardData, 341  
 wxGetClipboardFormatName, 342  
 wxGetDisplayName, 346  
 wxGetElapsedTime, 346  
 wxGetEmailAddress, 330  
 wxGetFreeMemory, 346  
 wxGetHomeDir, 346  
 wxGetHostName, 329, 346  
 wxGetMousePosition, 347  
 wxGetMultipleChoice, 334  
 wxGetOsVersion, 347  
 wxGetPrinterCommand, 338  
 wxGetPrinterFile, 339  
 wxGetPrinterMode, 339  
 wxGetPrinterOptions, 339  
 wxGetPrinterOrientation, 339  
 wxGetPrinterPreviewCommand, 339  
 wxGetPrinterScaling, 339  
 wxGetPrinterTranslation, 339  
 wxGetResource, 347  
 wxGetSingleChoice, 334  
 wxGetSingleChoiceData, 335  
 wxGetSingleChoiceIndex, 334  
 wxGetTempFileName, 331  
 wxGetTextFromUser, 334  
 wxGetUserId, 330, 348  
 wxGetUserName, 330, 348  
 wxGetWorkingDirectory, 330  
 wxGroupBox, 182  
 wxGroupBox styles, 368  
 wxGroupBox::~wxGroupBox, 183  
 wxGroupBox::Create, 183  
 wxGroupBox::wxGroupBox, 182  
 wxHashTable, 185  
 wxHashTable::~wxHashTable, 185  
 wxHashTable::BeginFind, 186  
 wxHashTable::Clear, 186  
 wxHashTable::Delete, 186  
 wxHashTable::Get, 186  
 wxHashTable::MakeKey, 186  
 wxHashTable::Next, 186  
 wxHashTable::Put, 187  
 wxHashTable::wxHashTable, 185  
 wxHelpInstance, 187

- wxHelpInstance::~wxHelpInstance, 187
- wxHelpInstance::DisplayBlock, 188
- wxHelpInstance::DisplayContents, 188
- wxHelpInstance::DisplaySection, 188
- wxHelpInstance::Initialize, 187
- wxHelpInstance::KeywordSearch, 188
- wxHelpInstance::LoadFile, 188
- wxHelpInstance::OnQuit, 188
- wxHelpInstance::Quit, 189
- wxHelpInstance::wxHelpInstance, 187
- wxHORIZONTAL, 369
- wxHORIZONTAL\_LABEL, 368
- wxHSCROLL, 369, 370
- wxIcon, 183, 184
- wxIcon::~wxIcon, 184
- wxIcon::GetHeight, 185
- wxIcon::GetWidth, 185
- wxIcon::wxIcon, 183
- wxICONIZE, 367
- wxIndividualLayoutConstraint, 190
- wxIndividualLayoutConstraint::Above, 190
- wxIndividualLayoutConstraint::Absolute, 190
- wxIndividualLayoutConstraint::AsIs, 190
- wxIndividualLayoutConstraint::Below, 190
- wxIndividualLayoutConstraint::LeftOf, 190
- wxIndividualLayoutConstraint::PercentOf, 191
- wxIndividualLayoutConstraint::RightOf, 191
- wxIndividualLayoutConstraint::SameAs, 191
- wxIndividualLayoutConstraint::Set, 191
- wxIndividualLayoutConstraint::Unconstrained, 190
- wxIndividualLayoutConstraint::wxIndividualLayoutConstraint, 189
- wxInitClipboard, 348
- wxIntPoint, 191
- wxIntPoint::wxIntPoint, 191
- wxIPCCleanUp, 348
- wxIPCInitialize, 348
- wxIsAbsolutePath, 329
- wxIsBusy, 349
- wxIsClipboardFormatAvailable, 342
- wxIsWild, 331
- wxItem styles, 368
- wxItem::Centre, 192
- wxItem::Command, 192
- wxItem::GetBackgroundColour, 192
- wxItem::GetButtonColour, 192
- wxItem::GetLabel, 192
- wxItem::GetLabelColour, 192
- wxItem::SetBackgroundColour, 193
- wxItem::SetButtonColour, 193
- wxItem::SetButtonFont, 193
- wxItem::SetLabel, 193
- wxItem::SetLabelColour, 193
- wxItem::SetLabelFont, 193
- wxKeyEvent, 195
- wxKeyEvent::controlDown, 193
- wxKeyEvent::ControlDown, 195
- wxKeyEvent::keyCode, 194
- wxKeyEvent::KeyCode, 195
- wxKeyEvent::Position, 196
- wxKeyEvent::shiftDown, 195
- wxKeyEvent::ShiftDown, 196
- wxKeyEvent::wxKeyEvent, 195
- wxKill, 348
- wxLayoutConstraints, 196
- wxLayoutConstraints::bottom, 196
- wxLayoutConstraints::centreX, 197
- wxLayoutConstraints::centreY, 197
- wxLayoutConstraints::height, 197
- wxLayoutConstraints::left, 197
- wxLayoutConstraints::right, 197
- wxLayoutConstraints::top, 197
- wxLayoutConstraints::width, 197
- wxLayoutConstraints::wxLayoutConstraints, 196
- wxLB\_ALWAYS\_SB, 369
- wxLB\_EXTENDED, 369
- wxLB\_MULTIPLE, 369
- wxLB\_NEEDED\_SB, 368
- wxLB\_SINGLE, 369
- wxList, 198
- wxList::~wxList, 199
- wxList::Append, 199
- wxList::Clear, 199
- wxList::DeleteContents, 199
- wxList::DeleteNode, 199
- wxList::DeleteObject, 199
- wxList::Find, 200
- wxList::First, 200
- wxList::Insert, 200
- wxList::Last, 200
- wxList::Member, 200
- wxList::Nth, 200
- wxList::Number, 201
- wxList::Sort, 201
- wxList::wxList, 198
- wxListBox, 202
- wxListBox styles, 368
- wxListBox::~wxListBox, 203
- wxListBox::Append, 203
- wxListBox::Clear, 203
- wxListBox::Create, 203
- wxListBox::Delete, 203
- wxListBox::Deselect, 203
- wxListBox::FindString, 204
- wxListBox::GetClientData, 204
- wxListBox::GetSelection, 204
- wxListBox::GetSelections, 204
- wxListBox::GetString, 204
- wxListBox::GetStringSelection, 204
- wxListBox::Number, 204
- wxListBox::Selected, 204
- wxListBox::Set, 205
- wxListBox::SetClientData, 205
- wxListBox::SetFirstItem, 205
- wxListBox::SetSelection, 205
- wxListBox::SetString, 205
- wxListBox::SetStringSelection, 205
- wxListBox::wxListBox, 202
- wxLoadUserResource, 349
- wxMakeConstraintFunction, 171
- wxMakeConstraintRange, 172
- wxMakeConstraintStrings, 171
- wxMakeFormBool, 171

wxMakeFormButton, 170  
 wxMakeFormDouble, 171  
 wxMakeFormFloat, 171  
 wxMakeFormLong, 170  
 wxMakeFormMessage, 170  
 wxMakeFormNewLine, 170  
 wxMakeFormShort, 171  
 wxMakeFormString, 171  
 wxMakeMetaFilePlaceable, 336  
 wxMatchWild, 331  
 wxMAXIMIZE, 367  
 wxMAXIMIZE\_BOX, 367  
 wxMDI\_CHILD, 367  
 wxMDI\_PARENT, 367  
 wxMemoryDC, 206  
 wxMemoryDC::SelectObject, 206  
 wxMemoryDC::wxMemoryDC, 206  
 wxMenu, 207  
 wxMenu::~~wxMenu, 207  
 wxMenu::Append, 207  
 wxMenu::AppendSeparator, 207  
 wxMenu::Break, 207  
 wxMenu::Check, 207  
 wxMenu::Checked, 208  
 wxMenu::Enable, 208  
 wxMenu::FindItem, 208  
 wxMenu::FindItemForId, 208  
 wxMenu::GetHelpString, 208  
 wxMenu::GetLabel, 208  
 wxMenu::GetTitle, 208  
 wxMenu::SetHelpString, 209  
 wxMenu::SetLabel, 209  
 wxMenu::SetTitle, 209  
 wxMenu::wxMenu, 207  
 wxMenuBar, 209  
 wxMenuBar::~~wxMenuBar, 209  
 wxMenuBar::Append, 209  
 wxMenuBar::Check, 210  
 wxMenuBar::Checked, 210  
 wxMenuBar::Enable, 210  
 wxMenuBar::EnableTop, 210  
 wxMenuBar::FindItemById, 210  
 wxMenuBar::FindMenuItem, 210  
 wxMenuBar::GetHelpString, 210  
 wxMenuBar::GetLabel, 211  
 wxMenuBar::GetLabelTop, 211  
 wxMenuBar::SetHelpString, 211  
 wxMenuBar::SetLabel, 211  
 wxMenuBar::SetLabelTop, 211  
 wxMenuBar::wxMenuBar, 209  
 wxMessage, 211, 212  
 wxMessage styles, 369  
 wxMessage::~~wxMessage, 212  
 wxMessage::Create, 212  
 wxMessage::wxMessage, 211  
 wxMessageBox, 335  
 wxMetaFile, 212  
 wxMetaFile::~~wxMetaFile, 213  
 wxMetaFile::Ok, 213  
 wxMetaFile::Play, 213  
 wxMetaFile::SetClipboard, 213  
 wxMetaFile::wxMetaFile, 212  
 wxMetaFileDC, 214  
 wxMetaFileDC::~~wxMetaFileDC, 214  
 wxMetaFileDC::Close, 214  
 wxMetaFileDC::wxMetaFileDC, 214  
 wxMINIMIZE, 367  
 wxMINIMIZE\_BOX, 367  
 wxMkdir, 331  
 wxMouseEvent, 215  
 wxMouseEvent::Button, 216  
 wxMouseEvent::ButtonDClick, 216  
 wxMouseEvent::ButtonDown, 216  
 wxMouseEvent::ButtonUp, 216  
 wxMouseEvent::controlDown, 214  
 wxMouseEvent::ControlDown, 216  
 wxMouseEvent::Dragging, 216  
 wxMouseEvent::Entering, 217  
 wxMouseEvent::IsButton, 217  
 wxMouseEvent::Leaving, 217  
 wxMouseEvent::LeftDClick, 217  
 wxMouseEvent::leftDown, 214, 215  
 wxMouseEvent::LeftDown, 217  
 wxMouseEvent::LeftIsDown, 217  
 wxMouseEvent::LeftUp, 217  
 wxMouseEvent::MiddleDClick, 218  
 wxMouseEvent::middleDown, 214  
 wxMouseEvent::MiddleDown, 218  
 wxMouseEvent::MiddleIsDown, 218  
 wxMouseEvent::MiddleUp, 218  
 wxMouseEvent::Moving, 218  
 wxMouseEvent::Position, 218  
 wxMouseEvent::RightDClick, 219  
 wxMouseEvent::rightDown, 215  
 wxMouseEvent::RightDown, 219  
 wxMouseEvent::RightIsDown, 219  
 wxMouseEvent::RightUp, 219  
 wxMouseEvent::shiftDown, 215  
 wxMouseEvent::ShiftDown, 219  
 wxMouseEvent::wxMouseEvent, 215  
 wxMouseEvent::x, 215  
 wxMouseEvent::y, 215  
 wxMultiText::GetLineLength, 220  
 wxMultiText::GetLineText, 220  
 wxMultiText::GetNumberOfLines, 220  
 wxMultiText::GetValue, 220  
 wxMultiText::PositionToXY, 220  
 wxMultiText::ShowPosition, 220  
 wxMultiText::XYToPosition, 221  
 wxNATIVE\_IMPL, 370  
 wxNEEDED\_SB, 368  
 wxNode::Data, 221  
 wxNode::Next, 221  
 wxNode::Previous, 221  
 wxNode::SetData, 221  
 wxNotifyEvent, 337  
 wxNow, 349  
 wxObject::\_\_type, 222  
 wxObject::Dump, 222  
 wxObject::GetClassInfo, 222  
 wxObject::IsKindOf, 222  
 wxObject::LoadObject, 222  
 wxObject::operator delete, 223  
 wxObject::operator new, 223

- wxObject::SaveObject, 223
- wxOpenClipboard, 342
- wxPageSetupData, 223
- wxPageSetupData::~wxPageSetupData, 223
- wxPageSetupData::EnableHelp, 223
- wxPageSetupData::EnableMargins, 223
- wxPageSetupData::EnableOrientation, 224
- wxPageSetupData::EnablePaper, 224
- wxPageSetupData::EnablePrinter, 224
- wxPageSetupData::GetDefaultInfo, 226
- wxPageSetupData::GetDefaultMinMargins, 225
- wxPageSetupData::GetEnableHelp, 225
- wxPageSetupData::GetEnableMargins, 225
- wxPageSetupData::GetEnableOrientation, 225
- wxPageSetupData::GetEnablePaper, 225
- wxPageSetupData::GetEnablePrinter, 225
- wxPageSetupData::GetMarginBottomRight, 224
- wxPageSetupData::GetMarginTopLeft, 224
- wxPageSetupData::GetMinMarginBottomRight, 225
- wxPageSetupData::GetMinMarginTopLeft, 224
- wxPageSetupData::GetOrientation, 225
- wxPageSetupData::GetPaperSize, 224
- wxPageSetupData::SetDefaultInfo, 227
- wxPageSetupData::SetDefaultMinMargins, 227
- wxPageSetupData::SetMarginBottomRight, 226
- wxPageSetupData::SetMarginTopLeft, 226
- wxPageSetupData::SetMinMarginBottomRight, 226
- wxPageSetupData::SetMinMarginTopLeft, 226
- wxPageSetupData::SetOrientation, 226
- wxPageSetupData::SetPaperSize, 226
- wxPageSetupData::wxPageSetupData, 223
- wxPageSetupDialog, 227
- wxPageSetupDialog::~wxPageSetupDialog, 227
- wxPageSetupDialog::GetPageSetupData, 227
- wxPageSetupDialog::Show, 228
- wxPageSetupDialog::wxPageSetupDialog, 227
- wxPanel, 228
- wxPanel styles, 370
- wxPanel::~wxPanel, 228
- wxPanel::Create, 229
- wxPanel::CreateItem, 229
- wxPanel::DrawAllStaticItems, 229
- wxPanel::Fit, 229
- wxPanel::GetBackgroundColour, 230
- wxPanel::GetButtonColour, 230
- wxPanel::GetButtonFont, 229
- wxPanel::GetCursor, 229
- wxPanel::GetDefaultItem, 229
- wxPanel::GetHorizontalSpacing, 230
- wxPanel::GetLabelColour, 230
- wxPanel::GetLabelFont, 230
- wxPanel::GetPanelDC, 230
- wxPanel::GetVerticalSpacing, 230
- wxPanel::LoadFromResource, 230
- wxPanel::NewLine, 231
- wxPanel::OnCommand, 231
- wxPanel::OnDefaultAction, 231
- wxPanel::OnEvent, 232
- wxPanel::OnItemEvent, 232
- wxPanel::OnItemLeftClick, 232
- wxPanel::OnItemMove, 232
- wxPanel::OnItemRightClick, 233
- wxPanel::OnItemSize, 233
- wxPanel::OnLeftClick, 233
- wxPanel::OnPaint, 233
- wxPanel::OnRightClick, 233
- wxPanel::PaintSelectionHandles, 234
- wxPanel::SetBackgroundColour, 234
- wxPanel::SetButtonColour, 234
- wxPanel::SetButtonFont, 234
- wxPanel::SetHorizontalSpacing, 234, 235
- wxPanel::SetLabelColour, 235
- wxPanel::SetLabelFont, 235
- wxPanel::SetLabelPosition, 234
- wxPanel::SetVerticalSpacing, 235
- wxPanel::Tab, 235
- wxPanel::wxPanel, 228
- wxPathList, 236
- wxPathList::Add, 236
- wxPathList::AddEnvList, 236
- wxPathList::EnsureFileAccessible, 236
- wxPathList::FindValidPath, 236
- wxPathList::Member, 236
- wxPathList::wxPathList, 236
- wxPathOnly, 329
- wxPen, 237
- wxPen::~wxPen, 237
- wxPen::GetCap, 237
- wxPen::GetColour, 237
- wxPen::GetDashes, 238
- wxPen::GetJoin, 238
- wxPen::GetStipple, 238
- wxPen::GetStyle, 238
- wxPen::GetWidth, 238
- wxPen::SetCap, 238
- wxPen::SetColour, 238
- wxPen::SetDashes, 239
- wxPen::SetJoin, 239
- wxPen::SetStipple, 239
- wxPen::SetStyle, 239
- wxPen::SetWidth, 239
- wxPen::wxPen, 237
- wxPenList, 239
- wxPenList::AddPen, 240
- wxPenList::FindOrCreatePen, 240
- wxPenList::RemovePen, 240
- wxPenList::wxPenList, 239
- wxPoint, 240
- wxPoint::wxPoint, 240
- wxPostDelete, 349
- wxPostScriptDC, 241
- wxPostScriptDC::GetStream, 241
- wxPostScriptDC::wxPostScriptDC, 240
- wxPreviewCanvas, 241
- wxPreviewCanvas::~wxPreviewCanvas, 241
- wxPreviewCanvas::OnPaint, 241
- wxPreviewCanvas::wxPreviewCanvas, 241
- wxPreviewControlBar, 243
- wxPreviewControlBar::~wxPreviewControlBar, 243
- wxPreviewControlBar::buttonFlags, 242
- wxPreviewControlBar::buttonFont, 242

---

wxPreviewControlBar::closeButton, 242  
 wxPreviewControlBar::CreateButtons, 243  
 wxPreviewControlBar::GetPrintPreview, 243  
 wxPreviewControlBar::GetZoomControl, 243  
 wxPreviewControlBar::nextPageButton, 242  
 wxPreviewControlBar::OnPaint, 243  
 wxPreviewControlBar::previousPageButton, 242  
 wxPreviewControlBar::printPreview, 242  
 wxPreviewControlBar::SetZoomControl, 244  
 wxPreviewControlBar::wxPreviewControlbar, 243  
 wxPreviewControlBar::zoomControl, 242  
 wxPreviewFrame, 244  
 wxPreviewFrame::~wxPreviewFrame, 244  
 wxPreviewFrame::controlBar, 244  
 wxPreviewFrame::CreateCanvas, 245  
 wxPreviewFrame::CreateControlBar, 244  
 wxPreviewFrame::Initialize, 245  
 wxPreviewFrame::OnClose, 245  
 wxPreviewFrame::previewCanvas, 244  
 wxPreviewFrame::printPreview, 244  
 wxPreviewFrame::wxPreviewFrame, 244  
 wxPrintData, 245  
 wxPrintData::~wxPrintData, 245  
 wxPrintData::EnableHelp, 245  
 wxPrintData::EnablePageNumbers, 246  
 wxPrintData::EnablePrintToFile, 246  
 wxPrintData::EnableSelection, 246  
 wxPrintData::GetAllPages, 246  
 wxPrintData::GetCollate, 246  
 wxPrintData::GetFromPage, 246  
 wxPrintData::GetMaxPage, 246  
 wxPrintData::GetMinPage, 246  
 wxPrintData::GetNoCopies, 247  
 wxPrintData::GetToPage, 247  
 wxPrintData::SetCollate, 247  
 wxPrintData::SetFromPage, 247  
 wxPrintData::SetMaxPage, 247  
 wxPrintData::SetMinPage, 247  
 wxPrintData::SetNoCopies, 247  
 wxPrintData::SetPrintToFile, 248  
 wxPrintData::SetSetupDialog, 248  
 wxPrintData::SetToPage, 248  
 wxPrintData::wxPrintData, 245  
 wxPrintDialog, 248  
 wxPrintDialog overview, 387  
 wxPrintDialog::~wxPrintDialog, 248  
 wxPrintDialog::GetPrintData, 248  
 wxPrintDialog::GetPrintDC, 248  
 wxPrintDialog::Show, 249  
 wxPrintDialog::wxPrintDialog, 248  
 wxPrinter, 249  
 wxPrinter::~wxPrinter, 249  
 wxPrinter::Abort, 249  
 wxPrinter::CreateAbortWindow, 249  
 wxPrinter::GetPrintData, 250  
 wxPrinter::Print, 250  
 wxPrinter::PrintDialog, 250  
 wxPrinter::ReportError, 250  
 wxPrinter::Setup, 250  
 wxPrinter::wxPrinter, 249  
 wxPrinterDC, 250  
 wxPrinterDC::wxPrinterDC, 250  
 wxPrintout, 251  
 wxPrintout::~wxPrintout, 251  
 wxPrintout::GetDC, 251  
 wxPrintout::GetPageInfo, 251  
 wxPrintout::GetPageSizeMM, 251  
 wxPrintout::GetPageSizePixels, 252  
 wxPrintout::GetPPIPrinter, 252  
 wxPrintout::GetPPIScreen, 252  
 wxPrintout::HasPage, 252  
 wxPrintout::IsPreview, 252  
 wxPrintout::OnBeginDocument, 252  
 wxPrintout::OnBeginPrinting, 253  
 wxPrintout::OnEndDocument, 253  
 wxPrintout::OnEndPrinting, 253  
 wxPrintout::OnPreparePrinting, 253  
 wxPrintout::OnPrintPage, 253  
 wxPrintout::wxPrintout, 251  
 wxPrintPreview, 254  
 wxPrintPreview::~wxPrintPreview, 254  
 wxPrintPreview::DrawBlankPage, 254  
 wxPrintPreview::GetCanvas, 254  
 wxPrintPreview::GetCurrentPage, 254  
 wxPrintPreview::GetFrame, 254  
 wxPrintPreview::GetMaxPage, 255  
 wxPrintPreview::GetMinPage, 255  
 wxPrintPreview::GetPrintData, 255  
 wxPrintPreview::GetPrintout, 255  
 wxPrintPreview::GetPrintoutForPrinting, 255  
 wxPrintPreview::Ok, 255  
 wxPrintPreview::PaintPage, 255  
 wxPrintPreview::Print, 256  
 wxPrintPreview::RenderPage, 256  
 wxPrintPreview::SetCanvas, 256  
 wxPrintPreview::SetCurrentPage, 256  
 wxPrintPreview::SetFrame, 256  
 wxPrintPreview::SetPrintout, 256  
 wxPrintPreview::SetZoom, 256  
 wxPrintPreview::wxPrintPreview, 254  
 wxQueryCol, 257  
 wxQueryCol overview, 395  
 wxQueryCol::~wxQueryCol, 257  
 wxQueryCol::AppendField, 258  
 wxQueryCol::BindVar, 257  
 wxQueryCol::FillVar, 257  
 wxQueryCol::GetData, 257  
 wxQueryCol::GetName, 257  
 wxQueryCol::GetSize, 258  
 wxQueryCol::GetType, 257  
 wxQueryCol::IsNullable, 258  
 wxQueryCol::IsRowDirty, 258  
 wxQueryCol::SetData, 258  
 wxQueryCol::SetFieldDirty, 258  
 wxQueryCol::SetName, 258  
 wxQueryCol::SetNullable, 258  
 wxQueryCol::SetType, 259  
 wxQueryCol::wxQueryCol, 257  
 wxQueryField, 259  
 wxQueryField overview, 395  
 wxQueryField::~wxQueryField, 259  
 wxQueryField::AllocData, 259  
 wxQueryField::ClearData, 259  
 wxQueryField::GetData, 259

---

wxQueryField::GetSize, 259  
wxQueryField::GetType, 260  
wxQueryField::IsDirty, 260  
wxQueryField::SetData, 260  
wxQueryField::SetDirty, 260  
wxQueryField::SetSize, 260  
wxQueryField::SetType, 260  
wxQueryField::wxQueryField, 259  
wxRadioButton, 260, 261, 369  
wxRadioButton::~wxRadioButton, 261  
wxRadioButton::Create, 261  
wxRadioButton::Enable, 262  
wxRadioButton::FindString, 262  
wxRadioButton::GetSelection, 262  
wxRadioButton::GetString, 263  
wxRadioButton::GetStringSelection, 262  
wxRadioButton::Number, 262  
wxRadioButton::SetSelection, 262  
wxRadioButton::SetStringSelection, 262  
wxRadioButton::Show, 263  
wxRadioButton::wxRadioButton, 260  
wxRadioButton, 263, 369  
wxRadioButton::~wxRadioButton, 264  
wxRadioButton::Create, 264  
wxRadioButton::GetValue, 264  
wxRadioButton::SetValue, 264  
wxRadioButton::wxRadioButton, 263  
wxRB\_GROUP, 369  
wxREADONLY, 370  
wxRecordSet, 264  
wxRecordSet overview, 395  
wxRecordSet::~wxRecordSet, 265  
wxRecordSet::AddNew, 265  
wxRecordSet::BeginQuery, 265  
wxRecordSet::BindVar, 265  
wxRecordSet::CanAppend, 265  
wxRecordSet::Cancel, 265  
wxRecordSet::CanRestart, 266  
wxRecordSet::CanScroll, 266  
wxRecordSet::CanTransact, 266  
wxRecordSet::CanUpdate, 266  
wxRecordSet::ConstructDefaultSQL, 266  
wxRecordSet::Delete, 266  
wxRecordSet::Edit, 266  
wxRecordSet::EndQuery, 267  
wxRecordSet::ExecuteSQL, 267  
wxRecordSet::FillVars, 267  
wxRecordSet::GetColName, 267  
wxRecordSet::GetColType, 267  
wxRecordSet::GetColumns, 267  
wxRecordSet::GetCurrentRecord, 268  
wxRecordSet::GetDatabase, 268  
wxRecordSet::GetDataSources, 268  
wxRecordSet::GetDefaultConnect, 269  
wxRecordSet::GetDefaultSQL, 269  
wxRecordSet::GetErrorCode, 269  
wxRecordSet::GetFieldData, 269  
wxRecordSet::GetFieldDataPtr, 269  
wxRecordSet::GetFilter, 270  
wxRecordSet::GetForeignKeys, 270  
wxRecordSet::GetNumberCols, 271  
wxRecordSet::GetNumberFields, 271  
wxRecordSet::GetNumberParams, 271  
wxRecordSet::GetNumberRecords, 271  
wxRecordSet::GetOptions, 271  
wxRecordSet::GetPrimaryKeys, 271  
wxRecordSet::GetResultSet, 271  
wxRecordSet::GetSortString, 272  
wxRecordSet::GetSQL, 272  
wxRecordSet::GetTableName, 272  
wxRecordSet::GetTables, 272  
wxRecordSet::GetType, 272  
wxRecordSet::GoTo, 272  
wxRecordSet::IsBOF, 272  
wxRecordSet::IsColNullable, 273  
wxRecordSet::IsDeleted, 273  
wxRecordSet::IsEOF, 273  
wxRecordSet::IsFieldDirty, 273  
wxRecordSet::IsFieldNull, 273  
wxRecordSet::IsOpen, 273  
wxRecordSet::Move, 273  
wxRecordSet::MoveFirst, 274  
wxRecordSet::MoveLast, 274  
wxRecordSet::MoveNext, 274  
wxRecordSet::MovePrev, 274  
wxRecordSet::Query, 274  
wxRecordSet::RecordCountFinal, 274  
wxRecordSet::Requery, 274  
wxRecordSet::SetDefaultSQL, 275  
wxRecordSet::SetFieldDirty, 274  
wxRecordSet::SetFieldNull, 275  
wxRecordSet::SetOptions, 275  
wxRecordSet::SetTableName, 275  
wxRecordSet::SetType, 275  
wxRecordSet::Update, 275  
wxRecordSet::wxRecordSet, 264  
wxRegisterClipboardFormat, 342  
wxRegisterEventClass, 337  
wxRegisterEventName, 338  
wxRegisterExternalEventHandlers, 338  
wxRelationship, 189  
wxRemoveFile, 331  
wxRemoveSecondaryEventHandler, 338  
wxRenameFile, 331  
wxRESIZE\_BORDER, 367  
wxResourceAddIdentifier, 355  
wxResourceClear, 355  
wxResourceCreateBitmap, 355  
wxResourceCreateIcon, 356  
wxResourceCreateMenuBar, 356  
wxResourceGetIdentifier, 357  
wxResourceParseData, 357  
wxResourceParseFile, 357  
wxResourceParseString, 357  
wxResourceRegisterBitmapData, 358  
wxRETAINED, 370  
wxRmdir, 331  
wxScreenDC, 275  
wxScreenDC::wxScreenDC, 275  
wxScrollBar, 276  
wxScrollBar::~wxScrollBar, 276  
wxScrollBar::Create, 276  
wxScrollBar::GetValue, 277  
wxScrollBar::GetValues, 277



wxScrollBar::SetObjectLength, 277  
 wxScrollBar::SetPageLength, 277  
 wxScrollBar::SetValue, 277  
 wxScrollBar::SetViewLength, 277  
 wxScrollBar::wxScrollBar, 276  
 wxSDI, 367  
 wxSendEvent, 338  
 wxServer, 278  
 wxServer::Create, 278  
 wxServer::OnAcceptConnection, 278  
 wxServer::wxServer, 278  
 wxSetClipboardData, 342  
 wxSetCursor, 336  
 wxSetDisplayName, 349  
 wxSetPrinterCommand, 339  
 wxSetPrinterFile, 340  
 wxSetPrinterMode, 340  
 wxSetPrinterOptions, 340  
 wxSetPrinterOrientation, 340  
 wxSetPrinterPreviewCommand, 340  
 wxSetPrinterScaling, 340  
 wxSetPrinterTranslation, 340  
 wxSetWorkingDirectory, 332  
 wxShell, 350  
 wxSleep, 350  
 wxSlider, 278  
 wxSlider styles, 369  
 wxSlider::~~wxSlider, 279  
 wxSlider::Create, 279  
 wxSlider::GetMax, 279  
 wxSlider::GetMin, 279  
 wxSlider::GetValue, 279  
 wxSlider::SetRange, 279  
 wxSlider::SetValue, 280  
 wxSlider::wxSlider, 278  
**wxSP\_3D**, 280  
**wxSP\_BORDER**, 280  
**wxSP\_NOBORDER**, 280  
 wxSplitterWindow, 280  
 wxSplitterWindow::~~wxSplitterWindow, 281  
 wxSplitterWindow::GetMinimumPaneSize, 281  
 wxSplitterWindow::GetSashPosition, 281  
 wxSplitterWindow::GetSplitMode, 281  
 wxSplitterWindow::GetWindow1, 281  
 wxSplitterWindow::GetWindow2, 282  
 wxSplitterWindow::Initialize, 282  
 wxSplitterWindow::IsSplit, 282  
 wxSplitterWindow::OnDoubleClickSash, 282  
 wxSplitterWindow::OnUnsplit, 283  
 wxSplitterWindow::SetMinimumPaneSize, 283  
 wxSplitterWindow::SetSashPosition, 283  
 wxSplitterWindow::SetSplitMode, 284  
 wxSplitterWindow::SplitHorizontally, 284  
 wxSplitterWindow::SplitVertically, 285  
 wxSplitterWindow::Unsplit, 285  
 wxSplitterWindow::wxSplitterWindow, 280  
 wxStartTimer, 350  
 wxSTAY\_ON\_TOP, 367  
 wxString, 286  
 wxString::~~wxString, 286  
 wxString::After, 287  
 wxString::Alloc, 286  
 wxString::Allocation, 286  
 wxString::Append, 287  
 wxString::At, 287  
 wxString::Before, 287  
 wxString::Capitalize, 287  
 wxString::Cat, 288  
 wxString::Chars, 288  
 wxString::CompareTo, 289  
 wxString::Contains, 289  
 wxString::Copy, 289  
 wxString::Del, 289  
 wxString::DownCase, 290  
 wxString::Elem, 290  
 wxString::Empty, 290  
 wxString::Error, 290  
 wxString::First, 290  
 wxString::Firstchar, 290  
 wxString::Freq, 290  
 wxString::From, 290  
 wxString::GetData, 291  
 wxString::GSub, 291  
 wxString::Index, 291  
 wxString::Insert, 291  
 wxString::IsAscii, 291  
 wxString::IsDefined, 291  
 wxString::IsNull, 292  
 wxString::IsNumber, 292  
 wxString::IsWord, 292  
 wxString::Last, 292  
 wxString::Lastchar, 292  
 wxString::Length, 292  
 wxString::LowerCase, 292  
 wxString::Matches, 292  
 wxString::OK, 293  
 wxString::operator (), 295  
 wxString::operator [], 295  
 wxString::operator +=, 295  
 wxString::operator <<, 295  
 wxString::operator =, 294  
 wxString::operator >>, 295  
 wxString::operator const char \*, 295  
 wxString::Prepend, 293  
 wxString::Readline, 293  
 wxString::Remove, 293  
 wxString::Replace, 293  
 wxString::Replicate, 293  
 wxString::Reverse, 293  
 wxString::sprintf, 294  
 wxString::Strip, 294  
 wxString::SubString, 294  
 wxString::Through, 294  
 wxString::Uppercase, 294  
 wxString::UpperCase, 294  
 wxString::wxString, 286  
 wxStringEq, 332  
 wxStringList, 298  
 wxStringList::~~wxStringList, 298  
 wxStringList::Add, 298  
 wxStringList::Delete, 298  
 wxStringList::ListToArray, 298  
 wxStringList::Member, 298  
 wxStringList::Sort, 298

wxStringList::wxStringList, 298  
 wxStringMatch, 332  
 wxStripMenuCodes, 350  
 wxSubType, 350  
 wxSYSTEM\_MENU, 367  
 wxTB\_3DBUTTONS, 370  
 wxTE\_PASSWORD, 369  
 wxTE\_PROCESS\_ENTER, 369  
 wxTE\_READONLY, 369  
 wxText, 299  
 wxText/wxMultiText styles, 369  
 wxText::~~wxText, 300  
 wxText::Copy, 300  
 wxText::Create, 300  
 wxText::Cut, 300  
 wxText::GetInsertionPoint, 300  
 wxText::GetLastPosition, 300  
 wxText::GetValue, 300  
 wxText::Paste, 300  
 wxText::Remove, 301  
 wxText::Replace, 301  
 wxText::SetEditable, 301  
 wxText::SetInsertionPoint, 301  
 wxText::SetInsertionPointEnd, 301  
 wxText::SetSelection, 301  
 wxText::SetValue, 301  
 wxText::wxText, 299  
 wxTextWindow, 302  
 wxTextWindow styles, 369  
 wxTextWindow::~~wxTextWindow, 303  
 wxTextWindow::Clear, 303  
 wxTextWindow::Copy, 303  
 wxTextWindow::Create, 303  
 wxTextWindow::Cut, 303  
 wxTextWindow::DiscardEdits, 303  
 wxTextWindow::GetContents, 303  
 wxTextWindow::GetInsertionPoint, 304  
 wxTextWindow::GetLastPosition, 304  
 wxTextWindow::GetLineLength, 304  
 wxTextWindow::GetLineText, 304  
 wxTextWindow::GetNumberOfLines, 304  
 wxTextWindow::LoadFile, 304  
 wxTextWindow::Modified, 305  
 wxTextWindow::OnChar, 305  
 wxTextWindow::operator <<, 307  
 wxTextWindow::Paste, 305  
 wxTextWindow::PositionToXY, 305  
 wxTextWindow::Remove, 305  
 wxTextWindow::Replace, 305  
 wxTextWindow::SaveFile, 306  
 wxTextWindow::SetEditable, 306  
 wxTextWindow::SetFont, 306  
 wxTextWindow::SetInsertionPoint, 306  
 wxTextWindow::SetInsertionPointEnd, 306  
 wxTextWindow::SetSelection, 304  
 wxTextWindow::ShowPosition, 306  
 wxTextWindow::WriteText, 307  
 wxTextWindow::wxTextWindow, 302  
 wxTextWindow::XYToPosition, 307  
 wxTHICK\_FRAME, 367  
 wxTimer, 307  
 wxTimer::~~wxTimer, 307  
 wxTimer::Interval, 308  
 wxTimer::Notify, 308  
 wxTimer::Start, 308  
 wxTimer::Stop, 308  
 wxTimer::wxTimer, 307  
 wxTINY\_CAPTION\_HORIZ, 367  
 wxTINY\_CAPTION\_VERT, 367  
 wxToLower, 351  
 wxToolBar, 308  
 wxToolBar styles, 370  
 wxToolBar::~~wxToolBar, 309  
 wxToolBar::AddSeparator, 309  
 wxToolBar::AddTool, 309  
 wxToolBar::CreateTools, 310  
 wxToolBar::DrawTool, 310  
 wxToolBar::EnableTool, 310  
 wxToolBar::FindToolForPosition, 310  
 wxToolBar::GetMaxSize, 310  
 wxToolBar::GetToolClientData, 310  
 wxToolBar::GetToolEnabled, 311  
 wxToolBar::GetToolLongHelp, 311  
 wxToolBar::GetToolShortHelp, 311  
 wxToolBar::GetToolState, 311  
 wxToolBar::Layout, 311  
 wxToolBar::OnLeftClick, 311  
 wxToolBar::OnMouseEnter, 312  
 wxToolBar::OnRightClick, 312  
 wxToolBar::SetMargins, 312  
 wxToolBar::SetToolLongHelp, 312  
 wxToolBar::SetToolPacking, 312  
 wxToolBar::SetToolSeparation, 312  
 wxToolBar::SetToolShortHelp, 312  
 wxToolBar::ToggleTool, 313  
 wxToolBar::wxToolBar, 308  
 wxToUpper, 351  
 wxTrace, 351  
 WXTRACE, 355  
 wxTraceLevel, 351  
 WXTRACELEVEL, 355  
 wxTransferFileToStream, 332  
 wxTransferStreamToFile, 332  
 wxTypeTree, 314  
 wxTypeTree::AddType, 314  
 wxTypeTree::GetName, 314  
 wxTypeTree::wxTypeTree, 314  
 wxUnix2DosFilename, 329  
 wxUpdateIterator, 315  
 wxUpdateIterator::GetHeight, 315  
 wxUpdateIterator::GetWidth, 315  
 wxUpdateIterator::GetX, 315  
 wxUpdateIterator::GetY, 315  
 wxUpdateIterator::operator ++, 315  
 wxUpdateIterator::wxUpdateIterator, 315  
 wxUSER\_COLOURS, 367, 370  
 wxVERTICAL, 369  
 wxVERTICAL\_LABEL, 368  
 wxView, 316  
 wxView overview, 374  
 wxView::~~wxView, 316  
 wxView::Activate, 316  
 wxView::Close, 317  
 wxView::GetDocument, 317

wxView::GetDocumentManager, 317  
 wxView::GetFrame, 317  
 wxView::GetViewName, 317  
 wxView::OnActivateView, 317  
 wxView::OnChangeFilename, 317  
 wxView::OnClose, 318  
 wxView::OnCreate, 318  
 wxView::OnCreatePrintout, 318  
 wxView::OnUpdate, 318  
 wxView::SetDocument, 318  
 wxView::SetFrame, 318  
 wxView::SetViewName, 319  
 wxView::viewDocument, 316  
 wxView::viewFrame, 316  
 wxView::viewTypeName, 316  
 wxView::wxView, 316  
 wxVSCROLL, 367, 370  
 wxWindow, 319  
 wxWindow::~~wxWindow, 319  
 wxWindow::AddChild, 319  
 wxWindow::CaptureMouse, 319  
 wxWindow::Center, 319  
 wxWindow::Centre, 320  
 wxWindow::ClientToScreen, 320  
 wxWindow::Close, 320  
 wxWindow::DestroyChildren, 320  
 wxWindow::DragAcceptFiles, 320  
 wxWindow::Enable, 320  
 wxWindow::GetCharHeight, 321  
 wxWindow::GetCharWidth, 321  
 wxWindow::GetChildren, 321  
 wxWindow::GetClientSize, 321  
 wxWindow::GetConstraints, 321  
 wxWindow::GetEventHandler, 321  
 wxWindow::GetGrandParent, 321  
 wxWindow::GetHandle, 322  
 wxWindow::GetLabel, 322  
 wxWindow::GetName, 322  
 wxWindow::GetParent, 322  
 wxWindow::GetPosition, 322  
 wxWindow::GetSize, 322  
 wxWindow::GetTextExtent, 322  
 wxWindow::GetUserEditMode, 323  
 wxWindow::GetWindowStyleFlag, 323

wxWindow::IsShown, 323  
 wxWindow::Layout, 323  
 wxWindow::Lower, 323  
 wxWindow::MakeModal, 323  
 wxWindow::Move, 324  
 wxWindow::PopupMenu, 324  
 wxWindow::Raise, 324  
 wxWindow::Refresh, 324  
 wxWindow::ReleaseMouse, 324  
 wxWindow::ScreenToClient, 324  
 wxWindow::SetAutoLayout, 325  
 wxWindow::SetClientSize, 326  
 wxWindow::SetColourMap, 326  
 wxWindow::SetConstraints, 325  
 wxWindow::SetCursor, 327  
 wxWindow::SetDoubleClick, 325  
 wxWindow::SetEventHandler, 327  
 wxWindow::SetFocus, 325  
 wxWindow::SetName, 325  
 wxWindow::SetSize, 325  
 wxWindow::SetSizeHints, 326  
 wxWindow::SetTitle, 327  
 wxWindow::SetUserEditMode, 327  
 wxWindow::Show, 327  
 wxWindow::wxWindow, 319  
 wxWindows predefined command identifiers, 377  
 wxWriteResource, 351  
 wxYield, 352

## —X—

x, 191, 215, 240  
 X features, 19  
 XYToPosition, 221, 307

## —Y—

y, 191, 215, 240

## —Z—

zoomControl, 242