

# **wxWidgets 2.8.8: A portable C++ and Python GUI toolkit**

Julian Smart, Robert Roebling, Vadim Zeitlin, Robin Dunn, et al

November, 2007

# Contents

<b>Copyright notice.....</b>	<b>xix</b>
<b>Introduction .....</b>	<b>1</b>
What is wxWidgets?.....	1
Why another cross-platform development tool? .....	1
wxWidgets requirements .....	3
Availability and location of wxWidgets.....	3
Acknowledgements.....	4
<b>Multi-platform development with wxWidgets .....</b>	<b>5</b>
Include files.....	5
Libraries .....	5
Configuration .....	5
Makefiles.....	6
Windows-specific files.....	6
Allocating and deleting wxWidgets objects .....	7
Architecture dependency.....	7
Conditional compilation.....	8
C++ issues.....	8
File handling .....	9
<b>Utilities and libraries supplied with wxWidgets.....</b>	<b>10</b>
<b>Programming strategies .....</b>	<b>12</b>
Strategies for reducing programming errors .....	12
Strategies for portability .....	12
Strategies for debugging .....	12
<b>Libraries list.....</b>	<b>15</b>
<b>Alphabetical class reference.....</b>	<b>18</b>
wxAboutDialogInfo .....	18
wxAcceleratorEntry .....	21
wxAcceleratorTable .....	23
wxAccessible .....	25
wxActivateEvent.....	33
wxActiveXContainer.....	34
wxActiveXEvent.....	38

wxAnimation .....	39
wxAnimationCtrl .....	42
wxApp .....	45
wxAppTraits .....	56
wxArchiveClassFactory .....	58
wxArchiveEntry .....	62
wxArchiveInputStream .....	64
wxArchiveIterator .....	65
wxArchiveNotifier .....	68
wxArchiveOutputStream .....	69
wxArray .....	71
wxArrayString .....	83
wxArtProvider .....	88
wxAuiDockArt .....	93
wxAuiTabArt .....	97
wxAuiManager .....	99
wxAuiNotebook .....	105
wxAuiPanelInfo .....	110
wxAutomationObject .....	119
wxBitmap .....	123
wxBitmapComboBox .....	135
wxBitmapButton .....	139
wxBitmapDataObject .....	145
wxBitmapHandler .....	146
wxBoxSizer .....	149
wxBrush .....	150
wxBrushList .....	156
wxBufferedDC .....	157
wxBufferedPaintDC .....	159
wxAutoBufferedPaintDC .....	160
wxBufferedInputStream .....	161
wxBufferedOutputStream .....	161
wxBusyCursor .....	162
wxBusyInfo .....	163
wxButton .....	164
wxCalculateLayoutEvent .....	167
wxCalendarCtrl .....	168
wxCalendarDateAttr .....	174
wxCalendarEvent .....	176
wxCaret .....	177

wxCheckBox.....	180
wxCheckListBox.....	183
wxChoice .....	186
wxChoicebook.....	189
wxClassInfo .....	190
wxCliet .....	191
wxClietDC.....	193
wxClietData .....	193
wxClietDataContainer.....	194
wxClipeboard.....	195
wxClipeboardTextEvent.....	198
wxClioseEvent.....	200
wxCmdLineParser.....	201
wxCollapsiblePane.....	210
wxCollapsiblePaneEvent.....	213
wxColour.....	214
wxColourData.....	218
wxColourDatabase.....	219
wxColourDialog .....	221
wxColourPickerCtrl.....	222
wxColourPickerEvent.....	224
wxComboBox .....	225
wxComboCtrl .....	232
wxComboPopup .....	246
wxCommand.....	249
wxCommandEvent.....	250
wxCommandProcessor.....	255
wxCondition .....	259
wxConfigBase.....	262
wxConection .....	276
wxCildFocusEvent .....	280
wxContextMenuEvent.....	281
wxContextHelp .....	282
wxContextHelpButton .....	284
wxControl.....	285
wxControlWithItems .....	286
wxCountingOutputStream .....	293
wxCriticalSection.....	294
wxCriticalSectionLocker.....	295
wxCSCnv .....	296



wxCursor .....	297
wxCustomDataObject .....	302
wxDataFormat .....	304
wxDatagramSocket .....	307
wxDataInputStream .....	308
wxDataObject .....	311
wxDataViewColumn .....	314
wxDataViewCtrl .....	317
wxDataViewEvent .....	321
wxDataViewListModelNotifier .....	323
wxDataViewModel .....	324
wxDataViewListModel .....	325
wxDataViewSortedListModel .....	328
wxDataViewRenderer .....	329
wxDataViewTextRenderer .....	331
wxDataViewProgressRenderer .....	331
wxDataViewToggleRenderer .....	332
wxDataViewBitmapRenderer .....	332
wxDataViewDateRenderer .....	332
wxDataViewCustomRenderer .....	333
wxDataObjectComposite .....	334
wxDataObjectSimple .....	335
wxDataOutputStream .....	337
wxDateEvent .....	339
wxDatePickerCtrl .....	340
wxDateSpan .....	343
wxDateTime .....	348
wxDateTimeHolidayAuthority .....	375
wxDateTimeWorkDays .....	375
wxDb .....	375
wxDbColDataPtr .....	406
wxDbColDef .....	406
wxDbColFor .....	407
wxDbCollnf .....	408
wxDbConnectInf .....	409
wxDbIdxDef .....	414
wxDbInf .....	415
wxDbTable .....	415
wxDbTableInf .....	451
wxDbGridCollInfo .....	451

wxDlgGridTableBase .....	453
wxDC.....	456
wxDCClipper.....	476
wxDDEClient .....	477
wxDDEConnection .....	478
wxDDEServer.....	482
wxDebugContext.....	483
wxDebugStreamBuf .....	487
wxDebugReport.....	488
wxDebugReportCompress .....	492
wxDebugReportPreview .....	493
wxDebugReportPreviewStd .....	494
wxDebugReportUpload.....	494
wxDelegateRendererNative .....	495
wxDialDialog .....	496
wxDialUpEvent.....	505
wxDialUpManager.....	506
wxDir .....	509
wxDirDialog.....	513
wxDirPickerCtrl.....	515
wxDirTraverser .....	518
wxDisplay.....	519
wxDllLoader .....	522
wxDocChildFrame.....	525
wxDocManager .....	527
wxDocMDIChildFrame.....	535
wxDocMDIParentFrame .....	537
wxDocParentFrame .....	538
wxDocTemplate.....	540
wxDocument.....	545
wxDragImage .....	552
wxDropFilesEvent .....	557
wxDropSource .....	558
wxDropTarget.....	561
wxDynamicLibrary.....	564
wxDynamicLibraryDetails .....	567
wxEncodingConverter.....	568
wxEraseEvent.....	571
wxEvent .....	572
wxEvtHandler .....	576

wxFFile.....	584
wxFFileInputStream .....	589
wxFFileOutputStream .....	590
wxFFileStream .....	591
wxFile .....	591
wxFileConfig .....	598
wxFileDataObject .....	599
wxFileDialog .....	600
wxFileDropTarget.....	604
wxFileHistory .....	605
wxFileInputStream .....	608
wxFileName .....	609
wxFileOutputStream .....	628
wxFilePickerCtrl.....	629
wxFileDirPickerEvent.....	632
wxFileStream.....	633
wxFileSystem .....	633
wxFileSystemHandler .....	636
wxFileType.....	639
wxFilterClassFactory.....	643
wxFilterInputStream .....	646
wxFilterOutputStream .....	647
wxFindDialogEvent .....	648
wxFindReplaceData.....	649
wxFindReplaceDialog .....	651
wxFlexGridSizer .....	652
wxFocusEvent .....	655
wxFont.....	655
wxFontData.....	668
wxFontDialog.....	670
wxFontEnumerator.....	672
wxFontList.....	673
wxFontMapper.....	674
wxFontPickerCtrl.....	678
wxFontPickerEvent .....	681
wxFrame .....	682
wxFSFile .....	692
wxFTP .....	694
wxGauge.....	701
wxGBPosition .....	705

wxGBSizerItem .....	706
wxGBSpan .....	708
wxGDIObject .....	709
wxGenericDirCtrl .....	709
wxGenericValidator .....	714
wxGLCanvas .....	715
wxGLContext .....	719
wxGraphicsBrush .....	721
wxGraphicsContext .....	721
wxGraphicsFont .....	727
wxGraphicsMatrix .....	727
wxGraphicsObject .....	729
wxGraphicsPath .....	729
wxGraphicsPen .....	732
wxGraphicsRenderer .....	733
wxGrid .....	735
wxGridCellAttr .....	769
wxGridBagSizer .....	772
wxGridCellBoolEditor .....	775
wxGridCellChoiceEditor .....	776
wxGridCellEditor .....	776
wxGridCellFloatEditor .....	778
wxGridCellNumberEditor .....	779
wxGridCellTextEditor .....	780
wxGridEditorCreatedEvent .....	781
wxGridEvent .....	782
wxGridRangeSelectEvent .....	785
wxGridSizeEvent .....	787
wxGridCellBoolRenderer .....	789
wxGridCellFloatRenderer .....	789
wxGridCellNumberRenderer .....	791
wxGridCellRenderer .....	791
wxGridCellStringRenderer .....	792
wxGridTableBase .....	793
wxGridSizer .....	797
wxHashMap .....	798
wxHashSet .....	803
wxHashTable .....	807
wxHelpController .....	809
wxHelpControllerHelpProvider .....	815

wxHelpEvent.....	815
wxHelpProvider .....	817
wxHtmlCell.....	819
wxHtmlColourCell.....	825
wxHtmlContainerCell .....	826
wxHtmlDCRenderer .....	831
wxHtmlEasyPrinting .....	833
wxHtmlFilter .....	837
wxHtmlHelpController .....	838
wxHtmlHelpData.....	843
wxHtmlHelpDialog.....	844
wxHtmlHelpFrame.....	846
wxHtmlHelpWindow .....	847
wxHtmlModalHelp .....	851
wxHtmlLinkInfo .....	851
wxHtmlListBox.....	853
wxSimpleHtmlListBox .....	855
wxHtmlParser .....	858
wxHtmlPrintout .....	863
wxHtmlTag.....	865
wxHtmlTagHandler.....	869
wxHtmlTagsModule .....	870
wxHtmlWidgetCell .....	871
wxHtmlWindow.....	872
wxHtmlLinkEvent.....	881
wxHtmlCellEvent.....	882
wxHtmlWinParser.....	883
wxHtmlWinTagHandler .....	888
wxHTTP .....	889
wxHyperlinkCtrl .....	890
wxHyperlinkEvent.....	894
wxIcon .....	894
wxIconBundle .....	901
wxIconLocation.....	902
wxIconizeEvent .....	903
wxIdleEvent .....	903
wxImage.....	906
wxImageHandler .....	930
wxImageList.....	933
wxIndividualLayoutConstraint .....	938

wxInitDialogEvent.....	941
wxInputStream .....	941
wxIPAddress.....	944
wxIPv4address .....	946
wxJoystick.....	948
wxJoystickEvent.....	954
wxKeyEvent.....	956
wxLayoutAlgorithm .....	961
wxLayoutConstraints.....	964
wxList<T> .....	966
wxListbook .....	974
wxListBox.....	974
wxListCtrl .....	980
wxListEvent.....	1000
wxListItem .....	1003
wxListItemAttr .....	1007
wxListView .....	1008
wxLocale .....	1011
wxLog .....	1018
wxLogChain .....	1025
wxLogGui.....	1027
wxLogNull .....	1027
wxLogPassThrough .....	1028
wxLogStderr .....	1029
wxLogStream.....	1029
wxLogTextCtrl.....	1030
wxLogWindow .....	1030
wxLongLong .....	1032
wxMask .....	1036
wxMaximizeEvent .....	1037
wxMBConv.....	1038
wxMBConvFile .....	1043
wxMBConvUTF7 .....	1044
wxMBConvUTF8 .....	1044
wxMBConvUTF16 .....	1045
wxMBConvUTF32 .....	1046
wxMDIChildFrame.....	1047
wxMDIClientWindow .....	1050
wxMDIParentFrame .....	1051
wxMediaCtrl .....	1058

wxMediaEvent .....	1066
wxMemoryBuffer .....	1066
wxMemoryDC .....	1069
wxMemoryFSHandler .....	1071
wxMemoryInputStream .....	1073
wxMemoryOutputStream .....	1073
wxMenu .....	1074
wxMenuBar .....	1088
wxMenuEvent .....	1098
wxMenuItem .....	1099
wxMessageDialog .....	1106
wxMetafile .....	1108
wxMetafileDC .....	1109
wxMimeTypesManager .....	1110
wxMiniFrame .....	1113
wxMirrorDC .....	1115
wxModule .....	1116
wxMouseCaptureChangedEvent .....	1118
wxMouseCaptureLostEvent .....	1119
wxMouseEvent .....	1119
wxMoveEvent .....	1128
wxMultiChoiceDialog .....	1129
wxMutex .....	1131
wxMutexLocker .....	1133
wxNode .....	1134
wxNotebook .....	1135
wxNotebookEvent .....	1144
wxNotebookSizer .....	1146
wxNotifyEvent .....	1146
wxObject .....	1148
wxObjectRefData .....	1151
wxOwnerDrawnComboBox .....	1152
wxOutputStream .....	1156
wxPageSetupDialog .....	1158
wxPageSetupDialogData .....	1159
wxPaintDC .....	1164
wxPaintEvent .....	1164
wxPalette .....	1166
wxPanel .....	1170
wxPasswordEntryDialog .....	1173

wxPathList .....	1174
wxPen .....	1175
wxPenList .....	1182
wxPickerBase .....	1183
wxPlatformInfo .....	1185
wxPoint .....	1193
wxPostScriptDC .....	1194
wxPowerEvent .....	1195
wxPreviewCanvas .....	1196
wxPreviewControlBar .....	1197
wxPreviewFrame .....	1199
wxPrintData .....	1200
wxPrintDialog .....	1206
wxPrintDialogData .....	1207
wxPrinter .....	1211
wxPrinterDC .....	1213
wxPrintout .....	1214
wxPrintPreview .....	1221
wxProcess .....	1224
wxProcessEvent .....	1229
wxProgressDialog .....	1230
wxPropertySheetDialog .....	1232
wxProtocol .....	1235
wxQuantize .....	1237
wxQueryLayoutInfoEvent .....	1238
wxRadioBox .....	1241
wxRadioButton .....	1249
wxRealPoint .....	1251
wxRect .....	1252
wxRecursionGuard .....	1259
wxRecursionGuardFlag .....	1260
wxRegEx .....	1260
wxRegion .....	1264
wxRegionIterator .....	1269
wxRegKey .....	1271
wxRendererNative .....	1276
wxRendererVersion .....	1280
wxRichTextAttr .....	1281
wxRichTextBuffer .....	1298
wxRichTextCharacterStyleDefinition .....	1315



wxRichTextCtrl .....	1316
wxRichTextEvent .....	1346
wxRichTextFileHandler .....	1350
wxRichTextFormattingDialog .....	1353
wxRichTextFormattingDialogFactory .....	1357
wxRichTextHeaderFooterData .....	1359
wxRichTextHTMLHandler .....	1362
wxRichTextListStyleDefinition .....	1365
wxRichTextParagraphStyleDefinition .....	1367
wxRichTextPrinting .....	1368
wxRichTextPrintout .....	1371
wxRichTextRange .....	1373
wxRichTextStyleDefinition .....	1376
wxRichTextStyleComboCtrl .....	1378
wxRichTextStyleListBox .....	1379
wxRichTextStyleListCtrl .....	1381
wxRichTextStyleOrganiserDialog .....	1383
wxRichTextStyleSheet .....	1387
wxRichTextXMLHandler .....	1390
wxSashEvent .....	1392
wxSashLayoutWindow .....	1394
wxSashWindow .....	1397
wxScopedArray .....	1402
wxScopedPtr .....	1404
wxScopedTiedPtr .....	1406
wxScreenDC .....	1407
wxScrollBar .....	1408
wxScrolledWindow .....	1414
wxScrollEvent .....	1423
wxScrollWinEvent .....	1426
wxSemaphore .....	1427
wxSetCursorEvent .....	1429
wxServer .....	1431
wxSimpleHelpProvider .....	1432
wxSearchCtrl .....	1432
wxSingleChoiceDialog .....	1435
wxSingleInstanceChecker .....	1438
wxSize .....	1440
wxSizeEvent .....	1443
wxSizer .....	1444

wxSizerFlags .....	1455
wxSizerItem .....	1458
wxSlider .....	1462
wxSocketAddress.....	1472
wxSocketBase .....	1472
wxSocketClient.....	1488
wxSocketEvent.....	1490
wxSocketInputStream .....	1491
wxSocketOutputStream .....	1492
wxSocketServer .....	1492
wxSound .....	1494
wxSpinButton.....	1496
wxSpinCtrl.....	1500
wxSpinEvent.....	1503
wxSplashScreen.....	1504
wxSplitterEvent.....	1505
wxSplitterWindow.....	1508
wxSplitterRenderParams.....	1518
wxStackFrame.....	1519
wxStackWalker.....	1521
wxStandardPaths .....	1522
wxStaticBitmap.....	1527
wxStaticBox .....	1530
wxStaticBoxSizer .....	1532
wxStaticLine .....	1532
wxStaticText .....	1534
wxStatusBar .....	1536
wxStdDialogButtonSizer .....	1541
wxStopWatch.....	1543
wxStreamBase .....	1544
wxStreamBuffer.....	1547
wxStreamToTextRedirector.....	1552
wxString .....	1553
wxStringBuffer .....	1579
wxStringBufferLength.....	1580
wxStringClientData.....	1581
wxStringInputStream.....	1582
wxStringOutputStream.....	1582
wxStringTokenizer.....	1583
wxSymbolPickerDialog .....	1586

wxSysColourChangedEvent.....	1590
wxSystemOptions .....	1590
wxSystemSettings.....	1594
wxTarClassFactory .....	1599
wxTarEntry.....	1599
wxTarInputStream.....	1603
wxTarOutputStream.....	1604
wxTaskBarIcon.....	1607
wxTCPClient.....	1609
wxTCPConnection .....	1610
wxTCPServer .....	1614
wxTempFile .....	1615
wxTempFileOutputStream.....	1618
wxTextAttr .....	1618
wxTextAttrEx .....	1623
wxTextCtrl.....	1633
wxTextDataObject.....	1653
wxTextDropTarget.....	1654
wxTextEntryDialog .....	1655
wxTextFile.....	1657
wxTextInputStream .....	1663
wxTextOutputStream .....	1666
wxTextValidator.....	1668
wxThread .....	1670
wxThreadHelper .....	1678
wxTimer .....	1680
wxTimerEvent.....	1682
wxTimeSpan.....	1683
wxTipProvider.....	1689
wxTipWindow .....	1691
wxToggleButton.....	1692
wxToolBar .....	1694
wxToolbook.....	1711
wxToolTip .....	1712
wxTopLevelWindow .....	1713
wxTreebook .....	1721
wxTreebookEvent .....	1726
wxTreeCtrl .....	1728
wxTreeItemId.....	1747
wxTreeEvent.....	1748

wxTreeItemData .....	1751
wxUpdateUIEvent .....	1752
wxURI .....	1757
wxURL .....	1763
wxURLDataObject .....	1766
wxValidator .....	1767
wxVariant .....	1769
wxVariantData .....	1778
wxView .....	1780
wxVListBox .....	1783
wxVScrolledWindow .....	1790
wxWindow .....	1795
wxWindow::ClientToWindowSize (p. ??)wxWindowUpdateLocker .....	1854
wxWindowCreateEvent .....	1854
wxWindowDC .....	1855
wxWindowDestroyEvent .....	1856
wxWindowDisabler .....	1857
wxWizard .....	1857
wxWizardEvent .....	1862
wxWizardPage .....	1863
wxWizardPageSimple .....	1865
wxXmlDocument .....	1866
wxXmlNode .....	1871
wxXmlProperty .....	1877
wxXmlResource .....	1878
wxXmlResourceHandler .....	1883
wxZipClassFactory .....	1888
wxZipEntry .....	1888
wxZipInputStream .....	1895
wxZipNotifier .....	1896
wxZipOutputStream .....	1897
wxZlibInputStream .....	1899
wxZlibOutputStream .....	1901
<b>Functions .....</b>	<b>1903</b>
Alphabetical functions and macros list .....	1903
Version macros .....	1908
Application initialization and termination .....	1909
Process control functions .....	1913
Thread functions .....	1917

---

File functions .....	1919
Network, user and OS functions .....	1925
String functions.....	1929
Dialog functions.....	1934
Math functions .....	1944
GDI functions.....	1944
Printer settings .....	1947
Clipboard functions .....	1949
Miscellaneous functions.....	1951
Byte order macros.....	1963
RTTI functions .....	1964
Log functions .....	1971
Time functions .....	1977
Debugging macros and functions .....	1979
Environment access functions .....	1983
<b>Constants.....</b>	<b>1985</b>
Preprocessor symbols defined by wxWidgets .....	1985
Standard event identifiers.....	1990
Keycodes .....	1993
Key Modifiers.....	1995
Language identifiers.....	1996
Stock items .....	2004
<b>Classes by category.....</b>	<b>2007</b>
<b>Topic overviews .....</b>	<b>2025</b>
Notes on using the reference .....	2026
Writing a wxWidgets application: a rough guide .....	2027
wxWidgets Hello World sample.....	2028
wxWidgets samples .....	2030
wxApp overview .....	2040
Window Sizing Overview .....	2042
Runtime class information (aka RTTI) overview .....	2044
Reference counting.....	2046
wxString overview .....	2047
Buffer classes overview .....	2052
Date and time classes overview .....	2052
Unicode support in wxWidgets.....	2056
wxMBConv classes overview .....	2059
Internationalization .....	2062

Writing non-English applications.....	2063
Container classes overview.....	2066
File classes and functions overview .....	2067
wxStreams overview .....	2067
wxLog classes overview .....	2069
Debugging overview .....	2072
wxConfig classes overview .....	2074
wxFileSystem .....	2075
Event handling overview.....	2077
C++ exceptions overview .....	2088
Window styles.....	2089
Window deletion overview .....	2089
wxDialog overview .....	2092
wxValidator overview .....	2092
Constraints overview.....	2094
Sizer overview .....	2098
XML-based resource system overview.....	2106
Scrolling overview .....	2115
Bitmaps and icons overview.....	2117
Device context overview.....	2120
wxFont overview.....	2121
Font encoding overview.....	2122
wxSplitterWindow overview .....	2123
wxTreeCtrl overview .....	2125
wxListCtrl overview .....	2126
wxImageList overview.....	2126
wxBookCtrl overview.....	2127
Common dialogs overview .....	2128
Document/view overview .....	2131
Toolbar overview.....	2138
wxGrid classes overview .....	2143
wxTipProvider overview.....	2144
Printing overview.....	2145
Printing under Unix (GTK+).....	2149
Multithreading overview .....	2149
Drag and drop overview.....	2150
wxDataObject overview .....	2151
Database classes overview .....	2152
Interprocess communication overview .....	2175
wxHTML overview.....	2179

wxRichTextCtrl overview .....	2188
wxAUI overview .....	2197
Environment variables .....	2198
wxPython overview .....	2198
Syntax of the builtin regular expression library .....	2210
Archive formats such as zip .....	2223
Backward compatibility .....	2230
<b>Platform details .....</b>	<b>2234</b>
wxGTK port.....	2234
wxMSW port.....	2234
wxMac port .....	2243
wxPalmOS port .....	2244
wxOS2 port.....	2244
wxMGL port .....	2244
wxX11 port.....	2244
<b>Index.....</b>	<b>2246</b>

## Copyright notice

Copyright (c) 1992-2006 Julian Smart, Robert Roebling, Vadim Zeitlin and other members  
of the wxWidgets team  
Portions (c) 1996 Artificial Intelligence Applications Institute

Please also see the wxWindows license files (preamble.txt, lgpl.txt, gpl.txt, licence.txt, licendoc.txt) for conditions of software and documentation use. Note that we use the old name wxWindows in the license, pending recognition of the new name by OSI.

## wxWindows Library License, Version 3.1

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

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

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

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public Licence as published by the Free Software Foundation; either version 2 of the Licence, 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 Licence for more details.

You should have received a copy of the GNU Library General Public Licence along with this software, usually in a file named COPYING.LIB. If not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

### EXCEPTION NOTICE

1. As a special exception, the copyright holders of this library give permission for additional uses of the text contained in this release of the library as licenced under the wxWindows Library Licence, applying either version 3.1 of the Licence, or (at your option) any later version of the Licence as published by the copyright holders of version 3.1 of the Licence document.
2. The exception is that you may use, copy, link, modify and distribute under your own terms, binary object code versions of works based on the Library.
3. If you copy code from files distributed under the terms of the GNU General Public Licence or the GNU Library General Public Licence into a copy of this library, as this licence permits, the exception does not apply to the code that you add in this way. To avoid misleading anyone as to the status of such modified files, you must delete this exception notice from such code and/or adjust the licensing conditions notice accordingly.



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

## **GNU Library General Public License, Version 2**

Copyright (C) 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA

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

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

### **Preamble**

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

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

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

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

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

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

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

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

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

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

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

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

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

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

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

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

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

The "Library", below, refers to any such software library or work which has been

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

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

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

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

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

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

- a) The modified work must itself be a software library.
- b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
- d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

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

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those

sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

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

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

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

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

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

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

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

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

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

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

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

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

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

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

- a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
- b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
- c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
- d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

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

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

distribute.

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

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

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

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

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

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

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

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

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by

public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

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

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

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

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

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

#### NO WARRANTY

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

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

EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

## END OF TERMS AND CONDITIONS

### Appendix: How to Apply These Terms to Your New Libraries

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

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

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

```
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, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

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

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

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

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

That's all there is to it!



# Introduction

## What is wxWidgets?

wxWidgets is a C++ framework providing GUI (Graphical User Interface) and other facilities on more than one platform. Version 2 currently supports all desktop versions of MS Windows, Unix with GTK+, Unix with Motif, and MacOS. An OS/2 port is in progress.

wxWidgets was originally developed at the Artificial Intelligence Applications Institute, University of Edinburgh, for internal use, and was first made publicly available in 1992. Version 2 is a vastly improved version written and maintained by Julian Smart, Robert Roebling, Vadim Zeitlin, Vaclav Slavik and many others.

This manual contains a class reference and topic overviews. For a selection of wxWidgets tutorials, please see the documentation page on the wxWidgets web site (<http://www.wxwidgets.org>).

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.

## Why another cross-platform development tool?

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

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

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

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

Version 2 goes much further than previous versions in terms of generality and features, allowing applications to be produced that are often indistinguishable from those produced

using single-platform toolkits such as Motif, GTK+ and MFC.

The importance of using a platform-independent class library cannot be overstated, since GUI application development is very time-consuming, and sustained popularity of particular GUIs cannot be guaranteed. Code can very quickly become obsolete if it addresses the wrong platform or audience. wxWidgets helps to insulate the programmer from these winds of change. Although wxWidgets may not be suitable for every application (such as an OLE-intensive program), it provides access to most of the functionality a GUI program normally requires, plus many extras such as network programming, PostScript output, and HTML rendering; and it can of course be extended as needs dictate. As a bonus, it provides a far cleaner and easier programming interface than the native APIs. Programmers may find it worthwhile to use wxWidgets even if they are developing on only one platform.

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

- Low cost (free, in fact!)
- You get the source.
- Available on a variety of popular platforms.
- Works with almost all popular C++ compilers and Python.
- Over 50 example programs.
- Over 1000 pages of printable and on-line documentation.
- Includes Tex2RTF, to allow you to produce your own documentation in Windows Help, HTML and Word RTF formats.
- Simple-to-use, object-oriented API.
- Flexible event system.
- Graphics calls include lines, rounded rectangles, splines, polylines, etc.
- Constraint-based and sizer-based layouts.
- Print/preview and document/view architectures.
- Toolbar, notebook, tree control, advanced list control classes.
- PostScript generation under Unix, normal MS Windows printing on the PC.
- MDI (Multiple Document Interface) support.
- Can be used to create DLLs under Windows, dynamic libraries on Unix.
- Common dialogs for file browsing, printing, colour selection, etc.
- Under MS Windows, support for creating metafiles and copying them to the clipboard.

- An API for invoking help from applications.
- Ready-to-use HTML window (supporting a subset of HTML).
- Network support via a family of socket and protocol classes.
- Support for platform independent image processing.
- Built-in support for many file formats (BMP, PNG, JPEG, GIF, XPM, PNM, PCX).

## **wxWidgets requirements**

To make use of wxWidgets, you currently need one of the following setups.

(a) MS-Windows:

1. A 32-bit or 64-bit PC running MS Windows.
2. A Windows compiler: MS Visual C++ (embedded Visual C++ for wxWinCE port), Borland C++, Watcom C++, Cygwin, MinGW, Metrowerks CodeWarrior, Digital Mars C++. See `install.txt` for details about compiler version supported.
3. At least 100 MB of disk space for source tree and additional space for libraries and application building (depends on compiler and build settings).

(b) Unix:

1. Almost any C++ compiler, including GNU C++ (EGCS 1.1.1 or above).
2. Almost any Unix workstation, and one of: GTK+ 1.2, GTK+ 2.0, Motif 1.2 or higher, Lesstif. If using the wxX11 port, no such widget set is required.
3. At least 100 MB of disk space for source tree and additional space for libraries and application building (depends on compiler and build settings).

(c) Mac OS/Mac OS X:

1. A PowerPC Mac running Mac OS 8.6/9.x (eg. Classic) or Mac OS X 10.x.
2. CodeWarrior 5.3, 6 or 7 for Classic Mac OS.
3. The Apple Developer Tools (eg. GNU C++), CodeWarrior 7 or above for Mac OS X.
4. At least 100 MB of disk space for source tree and additional space for libraries and application building (depends on compiler and build settings).

## **Availability and location of wxWidgets**

wxWidgets is available by anonymous FTP and World Wide Web from <ftp://biolpc22.york.ac.uk/pub> (<ftp://biolpc22.york.ac.uk/pub>) and/or <http://www.wxwidgets.org> (<http://www.wxwidgets.org>).

You can also buy a CD-ROM using the form on the Web site.

## Acknowledgements

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

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

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

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

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

# Multi-platform development with wxWidgets

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

## Include files

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

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

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

#ifdef __BORLANDC__
#pragma hdrstop
#endif

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

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

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

## Libraries

Most ports of wxWidgets can create either a static library or a shared library. wxWidgets can also be built in multilib and monolithic variants. See the *libraries list* (p. 15) for more information on these.

## Configuration

When using project files and makefiles directly to build wxWidgets, options are configurable in the file `"wx/XXX/setup.h"` where XXX is the required platform (such as

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

When using the 'configure' script to configure wxWidgets (on Unix and other platforms where configure is available), the corresponding `setup.h` files are generated automatically along with suitable makefiles. When using the RPM packages for installing wxWidgets on Linux, a correct `setup.h` is shipped in the package and this must not be changed.

## Makefiles

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

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

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

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

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

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

## Windows-specific files

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

### Resource file

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

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

which includes essential internal wxWidgets 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.

## Allocating and deleting wxWidgets objects

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

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

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

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

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

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

## Architecture dependency

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

```
wxInt32, wxInt16, wxInt8, wxUInt32, wxUInt16 = wxWord, wxUInt8 = wxByte
```

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

The macros handling bit-swapping with respect to the applications endianness are

described in the *Byte order macros* (p. 1963) section.

## Conditional compilation

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

## C++ issues

The following documents some miscellaneous C++ issues.

### Templates

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

### RTTI

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

### Type of NULL

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

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

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

### 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 wxWidgets itself and any wxWidgets 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 wxWidgets, 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++.

## File handling

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

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

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

## Utilities and libraries supplied with wxWidgets

In addition to the core wxWidgets library, a number of further libraries and utilities are supplied with each distribution.

Some are under the 'contrib' hierarchy which mirrors the structure of the main wxWidgets hierarchy. See also the 'utils' hierarchy. The first place to look for documentation about these tools and libraries is under the wxWidgets 'docs' hierarchy, for example `docs/htmlhelp/fl.chm`.

For other user-contributed packages, please see the Contributions page on the wxWidgets Web site (<http://www.wxwidgets.org>).

**Helpview** Helpview is a program for displaying wxWidgets HTML Help files. In many cases, you may wish to use the wxWidgets HTML Help classes from within your application, but this provides a handy stand-alone viewer. See *wxHTML Notes* (p. 2179) for more details. You can find it in `samples/html/helpview`.

**Tex2RTF** Supplied with wxWidgets is a utility called Tex2RTF for converting LaTeX manuals HTML, MS HTML Help, wxHTML Help, RTF, and Windows Help RTF formats. Tex2RTF is used for the wxWidgets 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. You can find it under `utils/tex2rtf`.

**Helpgen** Helpgen takes C++ header files and generates a Tex2RTF-compatible documentation file for each class it finds, using comments as appropriate. This is a good way to start a reference for a set of classes. Helpgen can be found in `utils/HelpGen`.

**Emulator** Xnest-based display emulator for X11-based PDA applications. On some systems, the Xnest window does not synchronise with the 'skin' window. This program can be found in `utils/emulator`.

**XRC resource system** This is the sizer-aware resource system, and uses XML-based resource specifications that can be generated by tools such as wxDesigner (<http://www.roebling.de>). You can find this in `src/xrc`, `include/wx/xrc`, `samples/xrc`. For more information, see the *XML-based resource system overview* (p. 2106).

**Object Graphics Library** OGL defines an API for applications that need to display objects connected by lines. The objects can be moved around and interacted with. You can find this in `contrib/src/ogl`, `contrib/include/wx/ogl`, and `contrib/samples/ogl`.

**Frame Layout library** FL provides sophisticated pane dragging and docking facilities. You can find this in `contrib/src/fl`, `contrib/include/wx/fl`, and `contrib/samples/fl`.

**Gizmos library** Gizmos is a collection of useful widgets and other classes. Classes

include `wxLEDNumberCtrl`, `wxEditableListBox`, `wxMultiCellCanvas`. You can find this in `contrib/src/gizmos`, `contrib/include/wx/gizmos`, and `contrib/samples/gizmos`.

**Net library** Net is a collection of very simple mail and web related classes. Currently there is only `wxEmail`, which makes it easy to send email messages via MAPI on Windows or `sendmail` on Unix. You can find this in `contrib/src/net` and `contrib/include/wx/net`.

**Animate library** Animate allows you to load animated GIFs and play them on a window. The library can be extended to use other animation formats. You can find this in `contrib/src/animate`, `contrib/include/wx/animate`, and `contrib/samples/animate`.

**MMedia library** Mmedia supports a variety of multimedia functionality. The status of this library is currently unclear. You can find this in `contrib/src/mmedia`, `contrib/include/wx/mmedia`, and `contrib/samples/mmedia`.

**Styled Text Control library** STC is a wrapper around Scintilla, a syntax-highlighting text editor. You can find this in `contrib/src/stc`, `contrib/include/wx/stc`, and `contrib/samples/stc`.

**Plot** Plot is a simple curve plotting library. You can find this in `contrib/src/plot`, `contrib/include/wx/plot`, and `contrib/samples/plot`.

## Programming strategies

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

### Strategies for reducing programming errors

#### Use ASSERT

Although I haven't done this myself within wxWidgets, 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 wxWidgets and your application. Using ASSERT is an example of 'defensive programming': it can alert you to problems later on.

#### Use wxString in preference to character arrays

Using wxString can be much safer and more convenient than using wxChar \*. Again, I haven't practiced 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 is 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.

### Strategies for portability

#### Use relative positioning or constraints

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

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

#### Use wxWidgets resource files

Use .xrc (wxWidgets resource files) where possible, because they can be easily changed independently of source code.

### Strategies for debugging

### **Positive thinking**

It is 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.

### **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!

### **Use a debugger**

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

### **Use logging functions**

There is a variety of logging functions that you can use in your program: see *Logging functions* (p. 1971).

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).

### **Use the wxWidgets debugging facilities**

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

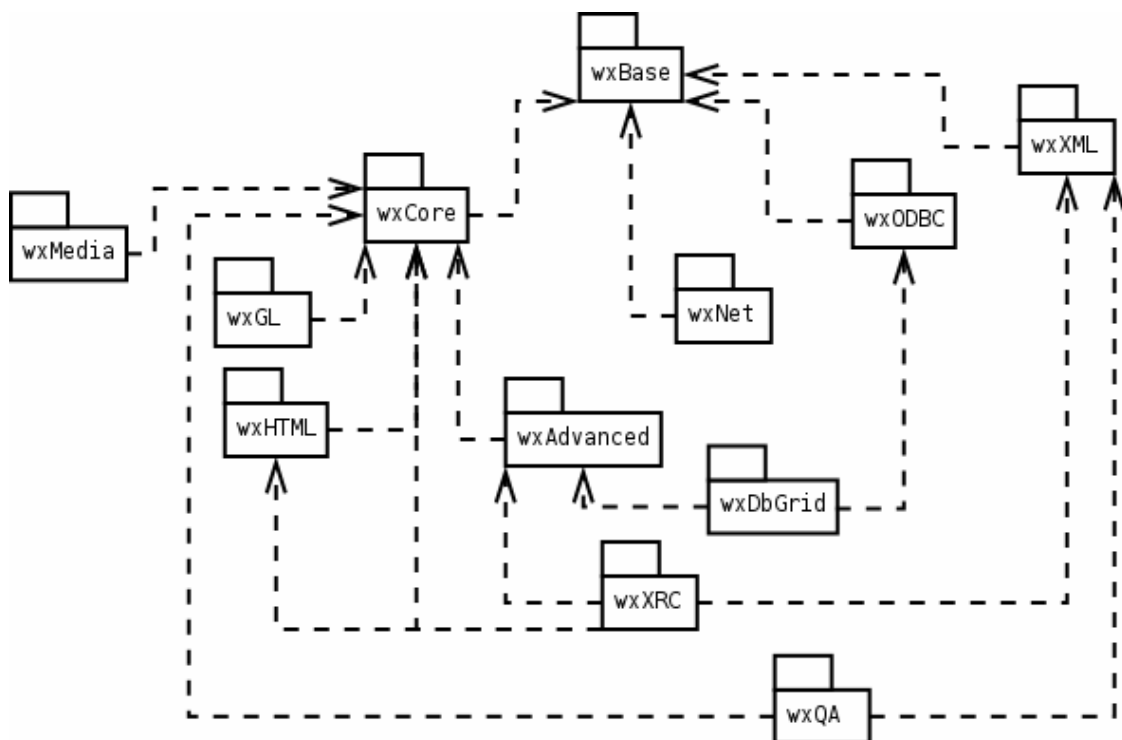
You should also use *debug macros* (p. 1979) as part of a 'defensive programming' strategy, scattering `wxASSERTs` liberally to test for problems in your code as early as possible. Forward thinking will save a surprising amount of time in the long run.

See the *debugging overview* (p. 2072) for further information.

## Libraries list

Starting from version 2.5.0 wxWidgets can be built either as a single large library (this is called the *monolithic build*) or as several smaller libraries (*multilib build*). Multilib build is the default.

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



### wxAui

This contains the Advanced User Interface docking library.

### wxBase

Every wxWidgets application must link against this library. It contains mandatory classes that any wxWidgets code depends on (e.g. *wxString* (p. 1553)) and portability classes that abstract differences between platforms. wxBase can be used to develop console mode applications, it does not require any GUI libraries or running X Window System on Unix.

### wxNet

Classes for network access:

- wxSocket classes (*wxSocketClient* (p. 1488), *wxSocketServer* (p. 1492) and related classes)

- *wxSocketOutputStream* (p. 1492) and *wxSocketInputStream* (p. 1491)
- sockets-based IPC classes (*wxTCPServer* (p. 482), *wxTCPClient* (p. 477) and *wxTCPConnection* (p. 478))
- *wxURL* (p. 1763)
- *wxInternetFSHandler* (a *wxFileSystem handler* (p. 2075)) Requires *wxBase*.

## **wxRichText**

This contains generic rich text control functionality.

## **wxXML**

This library contains simple classes for parsing XML documents. Note that their API *will* change in the future and backward compatibility will not be preserved. Use of this library in your applications is not recommended, it is only meant for use by XML resources system. Future versions of *wxWidgets* will contain new XML handling classes with DOM-like API. Requires *wxBase*.

## **wxCore**

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

## **wxAdvanced**

Advanced or rarely used GUI classes:

- *wxBufferedDC*
- *wxCalendarCtrl* (p. 168)
- *wxGrid classes* (p. 2143)
- *wxJoystick* (p. 948)
- *wxLayoutAlgorithm* (p. 961)
- *wxSplashScreen* (p. 1504)
- *wxTaskBarIcon* (p. 1607)
- *wxSound* (p. 1494)
- *wxWizard* (p. 1857)
- *wxSashLayoutWindow* (p. 1394)
- *wxSashWindow* (p. 1397)

Requires *wxCore* and *wxBase*.



**wxMedia**

Miscellaneous classes related to multimedia. Currently this library only contains *wxMediaCtrl* (p. 1058) but more classes will be added in the future.

Requires wxCore and wxBase.

**wxGL**

This library contains *wxGLCanvas* (p. 715) class for integrating OpenGL library with wxWidgets. Unlike all others, this library is *not* part of the monolithic library, it is always built as separate library. Requires wxCore and wxBase.

**wxHTML**

Simple HTML renderer and other *HTML rendering classes* (p. 2179) are contained in this library, as well as *wxHtmlHelpController* (p. 838), *wxBestHelpController* (p. 809) and *wxHtmlListBox* (p. 853). Requires wxCore and wxBase.

**wxODBC**

*Database classes* (p. 2152). Requires wxBase.

**wxQA**

This is the library containing extra classes for quality assurance. Currently it only contains *wxDebugReport* (p. 488) and related classes, but more will be added to it in the future.

Requires wxCore, wxBase and wxXML.

**wxDbGrid**

*wxDbGridTableBase* (p. 453) class which combines *wxGrid* (p. 735) and *wxDbTable* (p. 415). Requires wxODBC and wxAdvanced.

**wxXRC**

This library contains *wxXmlResource* (p. 1878) class that provides access to XML resource files in XRC format. Requires wxXML, wxCore, wxAdvanced and wxHTML.

# Alphabetical class reference

## **wxAboutDialogInfo**

wxAboutDialogInfo contains information shown in the standard *About* dialog displayed by the *wxAboutBox()* (p. 1934) function.

This class contains the general information about the program, such as its name, version, copyright and so on, as well as lists of the program developers, documentation writers, artists and translators. The simple properties from the former group are represented as a string with the exception of the program icon and the program web site, while the lists from the latter group are stored as *wxArrayString* (p. 83) and can be either set entirely at once using *SetDevelopers* (p. 20) and similar functions or built one by one using *AddDeveloper* (p. 19) etc.

Please also notice that while all the main platforms have the native implementation of the about dialog, they are often more limited than the generic version provided by wxWidgets and so the generic version is used if wxAboutDialogInfo has any fields not supported by the native version. Currently GTK+ version supports all the possible fields natively but MSW and Mac versions don't support URLs, licence text nor custom icons in the about dialog and if either of those is used, *wxAboutBox()* (p. 1934) will automatically use the generic version so you should avoid specifying these fields to achieve more native look and feel.

### **Derived from**

No base class

### **Include files**

<wx/aboutdlg.h>

## **wxAboutDialogInfo::wxAboutDialogInfo**

### **wxAboutDialogInfo()**

Default constructor leaves all fields are initially uninitialized, in general you should call at least *SetVersion* (p. 21), *SetCopyright* (p. 19) and *SetDescription* (p. 20).

## **wxAboutDialogInfo::AddArtist**

### **void AddArtist(const wxString& artist)**

Adds an artist name to be shown in the program credits.

### **See also**

*SetArtists* (p. 19)

### **wxAboutDialogInfo::AddDeveloper**

**void AddDeveloper(const wxString& *developer*)**

Adds a developer name to be shown in the program credits.

#### **See also**

*SetDevelopers* (p. 20)

### **wxAboutDialogInfo::AddDocWriter**

**void AddDocWriter(const wxString& *docwriter*)**

Adds a documentation writer name to be shown in the program credits.

#### **See also**

*SetDocWriters* (p. 20)

### **wxAboutDialogInfo::AddTranslator**

**void AddTranslator(const wxString& *translator*)**

Adds a translator name to be shown in the program credits. Notice that if no translator names are specified explicitly, *wxAboutBox()* (p. 1934) will try to use the translation of the string `translator-credits` from the currently used message catalog -- this can be used to show just the name of the translator of the program in the current language.

#### **See also**

*SetTranslators* (p. 21)

### **wxAboutDialogInfo::SetArtists**

**void SetArtists(const wxArrayString& *artists*)**

Sets the the list of artists to be shown in the program credits.

#### **See also**

*AddArtist* (p. 18)

### **wxAboutDialogInfo::SetCopyright**

**void SetCopyright(const wxString& *copyright*)**

Set the short string containing the program copyright information. Notice that any occurrences of " ( C ) " in *copyright* will be replaced by the copyright symbol (circled C) automatically, which means that you can avoid using this symbol in the program source

code which can be problematic,

### **wxAboutDialogInfo::SetDescription**

**void SetDescription(const wxString& desc)**

Set brief, but possibly multiline, description of the program.

### **wxAboutDialogInfo::SetDevelopers**

**void SetDevelopers(const wxArrayString& developers)**

Set the list of developers of the program.

#### **See also**

*AddDeveloper* (p. 19)

### **wxAboutDialogInfo::SetDocWriters**

**void SetDocWriters(const wxArrayString& docwriters)**

Set the list of documentation writers.

#### **See also**

*AddDocWriter* (p. 19)

### **wxAboutDialogInfo::SetIcon**

**void SetIcon(const wxIcon& icon)**

Set the icon to be shown in the dialog. By default the icon of the main frame will be shown if the native about dialog supports custom icons. If it doesn't but a valid icon is specified using this method, the generic about dialog is used instead so you should avoid calling this function for maximally native look and feel.

### **wxAboutDialogInfo::SetLicence**

**void SetLicence(const wxString& licence)**

Set the long, multiline string containing the text of the program licence.

Only GTK+ version supports showing the licence text in the native about dialog currently so the generic version will be used under all the other platforms if this method is called. To preserve the native look and feel it is advised that you do not call this method but provide a separate menu item in the "Help" menu for displaying the text of your program licence.

### **wxAboutDialogInfo::SetLicense**

**void SetLicense(const wxString& licence)**

This is the same as *SetLicence* (p. 20).

### **wxAboutDialogInfo::SetName**

**void SetName(const wxString& name)**

Set the name of the program. If this method is not called, the string returned by *wxApp::GetAppName()* (p. 47) will be shown in the dialog.

### **wxAboutDialogInfo::SetTranslators**

**void SetTranslators(const wxArrayString& translators)**

Set the list of translators. Please see *AddTranslator* (p. 19) for additional discussion.

### **wxAboutDialogInfo::SetVersion**

**void SetVersion(const wxString& version)**

Set the version of the program. The version is in free format, i.e. not necessarily in the *x.y.z* form but it shouldn't contain the "version" word.

### **wxAboutDialogInfo::SetWebSite**

**void SetWebSite(const wxString& url, const wxString& desc = wxEmptyString)**

Set the web site for the program and its description (which defaults to URL itself if empty).

Please notice that only GTK+ version currently supports showing the link in the native about dialog so if this method is called, the generic version will be used under all the other platforms.

## **wxAcceleratorEntry**

An object used by an application wishing to create an *accelerator table* (p. 23).

### **Derived from**

None

### **Include files**

<wx/accel.h>

### **See also**

*wxAcceleratorTable* (p. 23), *wxWindow::SetAcceleratorTable* (p. 1834)

### **wxAcceleratorEntry::wxAcceleratorEntry**

**wxAcceleratorEntry()**

Default constructor.

**wxAcceleratorEntry(int flags, int keyCode, int cmd)**

Constructor.

**Parameters**

*flags*

One of wxACCEL\_ALT, wxACCEL\_SHIFT, wxACCEL\_CTRL and wxACCEL\_NORMAL. Indicates which modifier key is held down.

*keyCode*

The keycode to be detected. See *Keycodes* (p. 1993) for a full list of keycodes.

*cmd*

The menu or control command identifier.

**wxAcceleratorEntry::GetCommand****int GetCommand() const**

Returns the command identifier for the accelerator table entry.

**wxAcceleratorEntry::GetFlags****int GetFlags() const**

Returns the flags for the accelerator table entry.

**wxAcceleratorEntry::GetKeyCode****int GetKeyCode() const**

Returns the keycode for the accelerator table entry.

**wxAcceleratorEntry::Set****void Set(int flags, int keyCode, int cmd)**

Sets the accelerator entry parameters.

**Parameters**

*flags*

One of wxACCEL\_ALT, wxACCEL\_SHIFT, wxACCEL\_CTRL and wxACCEL\_NORMAL. Indicates which modifier key is held down.

*keyCode*

The keycode to be detected. See *Keycodes* (p. 1993) for a full list of keycodes.

*cmd*

The menu or control command identifier.

## **wxAcceleratorTable**

An accelerator table allows the application to specify a table of keyboard shortcuts for menu or button commands.

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

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/accel.h>

### **Predefined objects**

Objects:

### **wxNullAcceleratorTable**

### **Example**

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

### **Remarks**

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

### **See also**

*wxAcceleratorEntry* (p. 21), *wxWindow::SetAcceleratorTable* (p. 1834)

## **wxAcceleratorTable::wxAcceleratorTable**

**wxAcceleratorTable()**

Default constructor.

**wxAcceleratorTable(const wxAcceleratorTable& *bitmap*)**

Copy constructor, uses *reference counting* (p. 2046).

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

Creates from an array of *wxAcceleratorEntry* (p. 21) objects.

**wxAcceleratorTable(const wxString& *resource*)**

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

**Parameters**

*n*

Number of accelerator entries.

*entries*

The array of entries.

*resource*

Name of a Windows accelerator.

**wxPython note:** The wxPython constructor accepts a list of *wxAcceleratorEntry* objects, or 3-tuples consisting of flags, keyCode, and cmd values like you would construct *wxAcceleratorEntry* objects with.

**wxPerl note:** The wxPerl constructor accepts a list of either *Wx::AcceleratorEntry* objects or references to 3-element arrays ( flags, keyCode, cmd ), like the parameters of *Wx::AcceleratorEntry::new*.

**wxAcceleratorTable::~~wxAcceleratorTable****~wxAcceleratorTable()**

Destroys the *wxAcceleratorTable* object. See *reference-counted object destruction* (p. 2046) for more info.

**wxAcceleratorTable::IsOk****bool IsOk() const**

Returns true if the accelerator table is valid.

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



Assignment operator, using *reference counting* (p. 2046).

### Parameters

*accel*

Accelerator table to assign.

### Return value

Returns a reference to this object.

## wxAccessible

The `wxAccessible` class allows `wxWidgets` applications, and `wxWidgets` itself, to return extended information about user interface elements to client applications such as screen readers. This is the main way in which `wxWidgets` implements accessibility features.

At present, only Microsoft Active Accessibility is supported by this class.

To use this class, derive from `wxAccessible`, implement appropriate functions, and associate an object of the class with a window using `wxWindow::SetAccessible` (p. 1834).

All functions return an indication of success, failure, or not implemented using values of the `wxAccStatus` enum type.

If you return `wxACC_NOT_IMPLEMENTED` from any function, the system will try to implement the appropriate functionality. However this will not work with all functions.

Most functions work with an *object id*, which can be zero to refer to 'this' UI element, or greater than zero to refer to the *n*th child element. This allows you to specify elements that don't have a corresponding `wxWindow` or `wxAccessible`; for example, the sash of a splitter window.

For details on the semantics of functions and types, please refer to the Microsoft Active Accessibility 1.2 documentation.

This class is compiled into `wxWidgets` only if the `wxUSE_ACCESSIBILITY` setup symbol is set to 1.

### Derived from

`wxObject` (p. 1148)

### Include files

`<wx/access.h>`

### Data structures

Functions return a `wxAccStatus` error code, which may be one of the following:

```
typedef enum
{
```

```
    wxACC_FAIL,           // The function failed
    wxACC_FALSE,          // The function returned false
    wxACC_OK,             // The function completed successfully
    wxACC_NOT_IMPLEMENTED, // The function is not implemented
    wxACC_NOT_SUPPORTED    // The function is not supported
} wxAccStatus
```

Directions of navigation are represented by the following:

```
typedef enum
{
    wxNAVDIR_DOWN,
    wxNAVDIR_FIRSTCHILD,
    wxNAVDIR_LASTCHILD,
    wxNAVDIR_LEFT,
    wxNAVDIR_NEXT,
    wxNAVDIR_PREVIOUS,
    wxNAVDIR_RIGHT,
    wxNAVDIR_UP
} wxNavDir
```

The role of a user interface element is represented by the following type:

```
typedef enum {
    wxROLE_NONE,
    wxROLE_SYSTEM_ALERT,
    wxROLE_SYSTEM_ANIMATION,
    wxROLE_SYSTEM_APPLICATION,
    wxROLE_SYSTEM_BORDER,
    wxROLE_SYSTEM_BUTTONDROPDOWN,
    wxROLE_SYSTEM_BUTTONDROPDOWNGRID,
    wxROLE_SYSTEM_BUTTONMENU,
    wxROLE_SYSTEM_CARET,
    wxROLE_SYSTEM_CELL,
    wxROLE_SYSTEM_CHARACTER,
    wxROLE_SYSTEM_CHART,
    wxROLE_SYSTEM_CHECKBUTTON,
    wxROLE_SYSTEM_CLIENT,
    wxROLE_SYSTEM_CLOCK,
    wxROLE_SYSTEM_COLUMN,
    wxROLE_SYSTEM_COLUMNHEADER,
    wxROLE_SYSTEM_COMBOBOX,
    wxROLE_SYSTEM_CURSOR,
    wxROLE_SYSTEM_DIAGRAM,
    wxROLE_SYSTEM_DIAL,
    wxROLE_SYSTEM_DIALOG,
    wxROLE_SYSTEM_DOCUMENT,
    wxROLE_SYSTEM_DROPLIST,
    wxROLE_SYSTEM_EQUATION,
    wxROLE_SYSTEM_GRAPHIC,
    wxROLE_SYSTEM_GRIP,
    wxROLE_SYSTEM_GROUPING,
```

```
wxROLE_SYSTEM_HELPBALLOON,  
wxROLE_SYSTEM_HOTKEYFIELD,  
wxROLE_SYSTEM_INDICATOR,  
wxROLE_SYSTEM_LINK,  
wxROLE_SYSTEM_LIST,  
wxROLE_SYSTEM_LISTITEM,  
wxROLE_SYSTEM_MENUBAR,  
wxROLE_SYSTEM_MENUITEM,  
wxROLE_SYSTEM_MENUPOPUP,  
wxROLE_SYSTEM_OUTLINE,  
wxROLE_SYSTEM_OUTLINEITEM,  
wxROLE_SYSTEM_PAGETAB,  
wxROLE_SYSTEM_PAGETABLIST,  
wxROLE_SYSTEM_PANE,  
wxROLE_SYSTEM_PROGRESSBAR,  
wxROLE_SYSTEM_PROPERTYPAGE,  
wxROLE_SYSTEM_PUSHBUTTON,  
wxROLE_SYSTEM_RADIOBUTTON,  
wxROLE_SYSTEM_ROW,  
wxROLE_SYSTEM_ROWHEADER,  
wxROLE_SYSTEM_SCROLLBAR,  
wxROLE_SYSTEM_SEPARATOR,  
wxROLE_SYSTEM_SLIDER,  
wxROLE_SYSTEM_SOUND,  
wxROLE_SYSTEM_SPINBUTTON,  
wxROLE_SYSTEM_STATICTEXT,  
wxROLE_SYSTEM_STATUSBAR,  
wxROLE_SYSTEM_TABLE,  
wxROLE_SYSTEM_TEXT,  
wxROLE_SYSTEM_TITLEBAR,  
wxROLE_SYSTEM_TOOLBAR,  
wxROLE_SYSTEM_TOOLTIP,  
wxROLE_SYSTEM_WHITESPACE,  
wxROLE_SYSTEM_WINDOW  
} wxAccRole
```

Objects are represented by the following type:

```
typedef enum {  
    wxOBJID_WINDOW = 0x00000000,  
    wxOBJID_SYSMENU = 0xFFFFFFFF,  
    wxOBJID_TITLEBAR = 0xFFFFFFFEE,  
    wxOBJID_MENU = 0xFFFFFFFDD,  
    wxOBJID_CLIENT = 0xFFFFFFFCC,  
    wxOBJID_VSCROLL = 0xFFFFFFFBB,  
    wxOBJID_HSCROLL = 0xFFFFFFFBA,  
    wxOBJID_SIZEGRIP = 0xFFFFFFF9,  
    wxOBJID_CARET = 0xFFFFFFF8,  
    wxOBJID_CURSOR = 0xFFFFFFF7,  
    wxOBJID_ALERT = 0xFFFFFFF6,  
    wxOBJID_SOUND = 0xFFFFFFF5  
} wxAccObject
```

Selection actions are identified by this type:

```
typedef enum
{
    wxACC_SEL_NONE           = 0,
    wxACC_SEL_TAKEFOCUS      = 1,
    wxACC_SEL_TAKESELECTION  = 2,
    wxACC_SEL_EXTENDSELECTION = 4,
    wxACC_SEL_ADDSELECTION   = 8,
    wxACC_SEL_REMOVESELECTION = 16
} wxAccSelectionFlags
```

States are represented by the following:

```
#define wxACC_STATE_SYSTEM_ALERT_HIGH      0x00000001
#define wxACC_STATE_SYSTEM_ALERT_MEDIUM   0x00000002
#define wxACC_STATE_SYSTEM_ALERT_LOW      0x00000004
#define wxACC_STATE_SYSTEM_ANIMATED       0x00000008
#define wxACC_STATE_SYSTEM_BUSY           0x00000010
#define wxACC_STATE_SYSTEM_CHECKED        0x00000020
#define wxACC_STATE_SYSTEM_COLLAPSED      0x00000040
#define wxACC_STATE_SYSTEM_DEFAULT        0x00000080
#define wxACC_STATE_SYSTEM_EXPANDED       0x00000100
#define wxACC_STATE_SYSTEM_EXTSELECTABLE  0x00000200
#define wxACC_STATE_SYSTEM_FLOATING       0x00000400
#define wxACC_STATE_SYSTEM_FOCUSABLE      0x00000800
#define wxACC_STATE_SYSTEM_FOCUSED        0x00001000
#define wxACC_STATE_SYSTEM_HOTTRACKED     0x00002000
#define wxACC_STATE_SYSTEM_INVISIBLE      0x00004000
#define wxACC_STATE_SYSTEM_MARQUEED       0x00008000
#define wxACC_STATE_SYSTEM_MIXED          0x00010000
#define wxACC_STATE_SYSTEM_MULTISELECTABLE 0x00020000
#define wxACC_STATE_SYSTEM_OFFSCREEN      0x00040000
#define wxACC_STATE_SYSTEM_PRESSED        0x00080000
#define wxACC_STATE_SYSTEM_PROTECTED      0x00100000
#define wxACC_STATE_SYSTEM_READONLY       0x00200000
#define wxACC_STATE_SYSTEM_SELECTABLE     0x00400000
#define wxACC_STATE_SYSTEM_SELECTED       0x00800000
#define wxACC_STATE_SYSTEM_SELFVOICING    0x01000000
#define wxACC_STATE_SYSTEM_UNAVAILABLE    0x02000000
```

Event identifiers that can be sent via *wxAccessible::NotifyEvent* (p. 32) are as follows:

```
#define wxACC_EVENT_SYSTEM_SOUND          0x0001
#define wxACC_EVENT_SYSTEM_ALERT          0x0002
#define wxACC_EVENT_SYSTEM_FOREGROUND     0x0003
#define wxACC_EVENT_SYSTEM_MENUSTART      0x0004
#define wxACC_EVENT_SYSTEM_MENUEND        0x0005
#define wxACC_EVENT_SYSTEM_MENUPOPUPSTART 0x0006
#define wxACC_EVENT_SYSTEM_MENUPOPUPEND   0x0007
#define wxACC_EVENT_SYSTEM_CAPTURESTART    0x0008
#define wxACC_EVENT_SYSTEM_CAPTUREEND      0x0009
```

```
#define wxACC_EVENT_SYSTEM_MOVESIZESTART      0x000A
#define wxACC_EVENT_SYSTEM_MOVESIZEEND        0x000B
#define wxACC_EVENT_SYSTEM_CONTEXTHELPSTART   0x000C
#define wxACC_EVENT_SYSTEM_CONTEXTHELPEND     0x000D
#define wxACC_EVENT_SYSTEM_DRAGDROPSTART      0x000E
#define wxACC_EVENT_SYSTEM_DRAGDROPEND        0x000F
#define wxACC_EVENT_SYSTEM_DIALOGSTART        0x0010
#define wxACC_EVENT_SYSTEM_DIALOGEND          0x0011
#define wxACC_EVENT_SYSTEM_SCROLLINGSTART      0x0012
#define wxACC_EVENT_SYSTEM_SCROLLINGEND        0x0013
#define wxACC_EVENT_SYSTEM_SWITCHSTART        0x0014
#define wxACC_EVENT_SYSTEM_SWITCHEND          0x0015
#define wxACC_EVENT_SYSTEM_MINIMIZESTART      0x0016
#define wxACC_EVENT_SYSTEM_MINIMIZEEND        0x0017
#define wxACC_EVENT_OBJECT_CREATE              0x8000
#define wxACC_EVENT_OBJECT_DESTROY             0x8001
#define wxACC_EVENT_OBJECT_SHOW                0x8002
#define wxACC_EVENT_OBJECT_HIDE                0x8003
#define wxACC_EVENT_OBJECT_REORDER             0x8004
#define wxACC_EVENT_OBJECT_FOCUS               0x8005
#define wxACC_EVENT_OBJECT_SELECTION           0x8006
#define wxACC_EVENT_OBJECT_SELECTIONADD        0x8007
#define wxACC_EVENT_OBJECT_SELECTIONREMOVE     0x8008
#define wxACC_EVENT_OBJECT_SELECTIONWITHIN     0x8009
#define wxACC_EVENT_OBJECT_STATECHANGE         0x800A
#define wxACC_EVENT_OBJECT_LOCATIONCHANGE      0x800B
#define wxACC_EVENT_OBJECT_NAMECHANGE          0x800C
#define wxACC_EVENT_OBJECT_DESCRIPTIONCHANGE   0x800D
#define wxACC_EVENT_OBJECT_VALUECHANGE         0x800E
#define wxACC_EVENT_OBJECT_PARENTCHANGE       0x800F
#define wxACC_EVENT_OBJECT_HELPCHANGE          0x8010
#define wxACC_EVENT_OBJECT_DEFACTIONCHANGE     0x8011
#define wxACC_EVENT_OBJECT_ACCELERATORCHANGE   0x8012
```

### **wxAccessible::wxAccessible**

**wxAccessible(wxWindow\* win = NULL)**

Constructor, taking an optional window. The object can be associated with a window later.

### **wxAccessible::~~wxAccessible**

**~wxAccessible()**

Destructor.

### **wxAccessible::DoDefaultAction**

**virtual wxAccStatus DoDefaultAction(int childId)**

Performs the default action for the object. *childId* is 0 (the action for this object) or greater than 0 (the action for a child). Return `wxACC_NOT_SUPPORTED` if there is no default action for this window (e.g. an edit control).

### **wxAccessible::GetChild**

**virtual wxAccStatus GetChild(int *childId*, wxAccessible\*\* *child*)**

Gets the specified child (starting from 1). If *child* is NULL and the return value is `wxACC_OK`, this means that the child is a simple element and not an accessible object.

### **wxAccessible::GetChildCount**

**virtual wxAccStatus GetChildCount(int\* *childCount*)**

Returns the number of children in *childCount*.

### **wxAccessible::GetDefaultAction**

**virtual wxAccStatus GetDefaultAction(int *childId*, wxString\* *actionName*)**

Gets the default action for this object (0) or a child (greater than 0). Return `wxACC_OK` even if there is no action. *actionName* is the action, or the empty string if there is no action. The retrieved string describes the action that is performed on an object, not what the object does as a result. For example, a toolbar button that prints a document has a default action of "Press" rather than "Prints the current document."

### **wxAccessible::GetDescription**

**virtual wxAccStatus GetDescription(int *childId*, wxString\* *description*)**

Returns the description for this object or a child.

### **wxAccessible::GetFocus**

**virtual wxAccStatus GetFocus(int\* *childId*, wxAccessible\*\* *child*)**

Gets the window with the keyboard focus. If *childId* is 0 and *child* is NULL, no object in this subhierarchy has the focus. If this object has the focus, *child* should be 'this'.

### **wxAccessible::GetHelpText**

**virtual wxAccStatus GetHelpText(int *childId*, wxString\* *helpText*)**

Returns help text for this object or a child, similar to tooltip text.

### **wxAccessible::GetKeyboardShortcut**

**virtual wxAccStatus GetKeyboardShortcut(int *childId*, wxString\* *shortcut*)**

Returns the keyboard shortcut for this object or child. Return e.g. ALT+K.

### **wxAcessible::GetLocation**

**virtual wxAccStatus GetLocation(wxRect& rect, int elementId)**

Returns the rectangle for this object (id is 0) or a child element (id is greater than 0). *rect* is in screen coordinates.

### **wxAcessible::GetName**

**virtual wxAccStatus GetName(int childId, wxString\* name)**

Gets the name of the specified object.

### **wxAcessible::GetParent**

**virtual wxAccStatus GetParent(wxAcessible\*\* parent)**

Returns the parent of this object, or NULL.

### **wxAcessible::GetRole**

**virtual wxAccStatus GetRole(int childId, wxAccRole\* role)**

Returns a role constant describing this object. See *wxAcessible* (p. 25) for a list of these roles.

### **wxAcessible::GetSelections**

**virtual wxAccStatus GetSelections(wxVariant\* selections)**

Gets a variant representing the selected children of this object.

Acceptable values are:

- a null variant (IsNull() returns TRUE)
- a list variant (GetType() == wxT("list"))
- an integer representing the selected child element, or 0 if this object is selected (GetType() == wxT("long"))
- a "void\*" pointer to a wxAcessible child object

### **wxAcessible::GetState**

**virtual wxAccStatus GetState(int childId, long\* state)**

Returns a state constant. See *wxAcessible* (p. 25) for a list of these states.

**wxAccessible::GetValue****virtual wxAccStatus GetValue(int childId, wxString\* strValue)**

Returns a localized string representing the value for the object or child.

**wxAccessible::GetWindow****wxWindow\* GetWindow()**

Returns the window associated with this object.

**wxAccessible::HitTest****virtual wxAccStatus HitTest(const wxPoint& pt, int\* childId, wxAccessible\*\* childObject)**

Returns a status value and object id to indicate whether the given point was on this or a child object. Can return either a child object, or an integer representing the child element, starting from 1.

*pt* is in screen coordinates.

**wxAccessible::Navigate****virtual wxAccStatus Navigate(wxNavDir navDir, int fromId, int\* toId, wxAccessible\*\* toObject)**

Navigates from *fromId* to *toId/toObject*.

**wxAccessible::NotifyEvent****virtual static void NotifyEvent(int eventType, wxWindow\* window, wxAccObject objectType, int objectType)**

Allows the application to send an event when something changes in an accessible object.

**wxAccessible::Select****virtual wxAccStatus Select(int childId, wxAccSelectionFlags selectFlags)**

Selects the object or child. See *wxAccessible* (p. 25) for a list of the selection actions.

**wxAccessible::SetWindow****void SetWindow(wxWindow\* window)**

Sets the window associated with this object.



## wxActivateEvent

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

### Derived from

*wxEvent* (p. 572)

*wxObject* (p. 1148)

### Include files

<wx/event.h>

### Event table macros

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

<b>EVT_ACTIVATE(func)</b>	Process a <code>wxEVT_ACTIVATE</code> event.
<b>EVT_ACTIVATE_APP(func)</b>	Process a <code>wxEVT_ACTIVATE_APP</code> event.
<b>EVT_HIBERNATE(func)</b>	Process a hibernate event, supplying the member function. This event applies to <code>wxApp</code> only, and only on Windows SmartPhone and PocketPC. It is generated when the system is low on memory; the application should free up as much memory as possible, and restore full working state when it receives a <code>wxEVT_ACTIVATE</code> or <code>wxEVT_ACTIVATE_APP</code> event.

### Remarks

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

An application is activated or deactivated when one of its frames becomes activated, or a frame becomes inactivated resulting in all application frames being inactive.

Please note that usually you should call *event.Skip()* (p. 575) in your handlers for these events as not doing so can result in strange effects.

### See also

*Event handling overview* (p. 2077), *wxApp::IsActive* (p. 49)

## wxActivateEvent::wxActivateEvent

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

Constructor.

### **wxActivateEvent::GetActive**

#### **bool GetActive() const**

Returns true if the application or window is being activated, false otherwise.

## **wxActiveXContainer**

`wxActiveXContainer` is a host for an activex control on Windows (and as such is a platform-specific class). Note that the `HWND` that the class contains is the actual `HWND` of the activex control so using dynamic events and connecting to `wxEVT_SIZE`, for example, will receive the actual size message sent to the control.

It is somewhat similar to the ATL class `CAXWindow` in operation.

The size of the activex control's content is generally guaranteed to be that of the client size of the parent of this `wxActiveXContainer`.

You can also process activex events through `wxEVT_ACTIVEX` or the corresponding message map macro `EVT_ACTIVEX`.

### **See also**

*wxActiveXEvent* (p. 38)

### **Derived from**

*wxControl* (p. 285)

### **Include files**

<wx/msw/ole/activex.h>

### **Example**

This is an example of how to use the Adobe Acrobat Reader ActiveX control to read PDF files (requires Acrobat Reader 4 and up). Controls like this are typically found and dumped from `OLEVIEW.exe` that is distributed with Microsoft Visual C++. This example also demonstrates how to create a backend for *wxMediaCtrl* (p. 1058).

```
//+++++
+++++
//
// wxPDFMediaBackend
//
//
http://partners.adobe.com/public/developer/en/acrobat/sdk/pdf/iac/
IACOverview.pdf
//+++++
+++++
```

```
#include "wx/mediactrl.h"          // wxMediaBackendCommonBase
#include "wx/msw/ole/activex.h"    // wxActiveXContainer
#include "wx/msw/ole/automtn.h"    // wxAutomationObject

const IID DIID__DPdf =
{0xCA8A9781,0x280D,0x11CF,{0xA2,0x4D,0x44,0x45,0x53,0x54,0x00,0x00}};
const IID DIID__DPdfEvents =
{0xCA8A9782,0x280D,0x11CF,{0xA2,0x4D,0x44,0x45,0x53,0x54,0x00,0x00}};
const CLSID CLSID_Pdf =
{0xCA8A9780,0x280D,0x11CF,{0xA2,0x4D,0x44,0x45,0x53,0x54,0x00,0x00}};

class WXDLLIMPEXP_MEDIA wxPDFMediaBackend : public
wxMediaBackendCommonBase
{
public:
    wxPDFMediaBackend() : m_pAX(NULL) {}
    virtual ~wxPDFMediaBackend()
    {
        if(m_pAX)
        {
            m_pAX->DissociateHandle();
            delete m_pAX;
        }
    }
    virtual bool CreateControl(wxControl* ctrl, wxWindow* parent,
                               wxWindowID id,
                               const wxPoint& pos,
                               const wxSize& size,
                               long style,
                               const wxValidator& validator,
                               const wxString& name)
    {
        IDispatch* pDispatch;
        if( ::CoCreateInstance(CLSID_Pdf, NULL,
                               CLSCTX_INPROC_SERVER,
                               DIID__DPdf,
                               (void**)&pDispatch) != 0 )
            return false;

        m_PDF.SetDispatchPtr(pDispatch); // wxAutomationObject will
        release itself

        if ( !ctrl->wxControl::Create(parent, id, pos, size,
                                       (style & ~wxBORDER_MASK) |
wxBORDER_NONE,
                                       validator, name) )
            return false;

        m_ctrl = wxStaticCast(ctrl, wxMediaCtrl);
        m_pAX = new wxActiveXContainer(ctrl,
                                       DIID__DPdf,
                                       pDispatch);
    }
};
```

```
wxPDFMediaBackend::ShowPlayerControls(wxMEDIACTRLPLAYERCONTROLS_NO
NE);
    return true;
}

virtual bool Play()
{
    return true;
}
virtual bool Pause()
{
    return true;
}
virtual bool Stop()
{
    return true;
}

virtual bool Load(const wxString& fileName)
{
    if(m_PDF.CallMethod(wxT("LoadFile"), fileName).GetBool())
    {
        m_PDF.CallMethod(wxT("setCurrentPage"),
wxVariant((long)0));
        NotifyMovieLoaded(); // initial refresh
        wxSizeEvent event;
        m_pAX->OnSize(event);
        return true;
    }

    return false;
}
virtual bool Load(const wxURI& location)
{
    return m_PDF.CallMethod(wxT("LoadFile"),
location.BuildUnescapedURI()).GetBool();
}
virtual bool Load(const wxURI& WXUNUSED(location),
                    const wxURI& WXUNUSED(proxy))
{
    return false;
}

virtual wxMediaState GetState()
{
    return wxMEDIASTATE_STOPPED;
}

virtual bool SetPosition(wxLongLong where)
{
    m_PDF.CallMethod(wxT("setCurrentPage"),
wxVariant((long)where.GetValue()));
    return true;
}
virtual wxLongLong GetPosition()
{
    return 0;
}
```

```
    }
    virtual wxLongLong GetDuration()
    {
        return 0;
    }

    virtual void Move(int WXUNUSED(x), int WXUNUSED(y),
                     int WXUNUSED(w), int WXUNUSED(h))
    {
    }
    wxSize GetVideoSize() const
    {
        return wxDefaultSize;
    }

    virtual double GetPlaybackRate()
    {
        return 0;
    }
    virtual bool SetPlaybackRate(double)
    {
        return false;
    }

    virtual double GetVolume()
    {
        return 0;
    }
    virtual bool SetVolume(double)
    {
        return false;
    }

    virtual bool ShowPlayerControls(wxMediaCtrlPlayerControls
flags)
    {
        if(flags)
        {
            m_PDF.CallMethod(wxT("setShowToolbar"), true);
            m_PDF.CallMethod(wxT("setShowScrollbars"), true);
        }
        else
        {
            m_PDF.CallMethod(wxT("setShowToolbar"), false);
            m_PDF.CallMethod(wxT("setShowScrollbars"), false);
        }

        return true;
    }

    wxActiveXContainer* m_pAX;
    wxAutomationObject m_PDF;

    DECLARE_DYNAMIC_CLASS(wxPDFMediaBackend)
};

IMPLEMENT_DYNAMIC_CLASS(wxPDFMediaBackend, wxMediaBackend);
```

---

```
Put this in one of your existant source files and then create a wxMediaCtrl with //[this]
is the parent window, "myfile.pdf" is the PDF file to open
wxMediaCtrl* mymediactrl = new wxMediaCtrl(this, wxT("myfile.pdf"),
wxID_ANY,
wxDefaultPosition,
wxSize(300,300),
0,
wxT("wxPDFMediaBackend"));
```

### **wxActiveXContainer::wxActiveXContainer**

```
wxActiveXContainer(    wxWindow* parent,    REFIID iid,    IUnknown*
pUnk,                )
```

Creates this activex container.

*parent*

parent of this control. Must not be NULL.

*iid*

COM IID of pUnk to query. Must be a valid interface to an activex control.

*pUnk*

Interface of activex control

### **wxActiveXEvent**

An event class for handling activex events passed from *wxActiveXContainer* (p. 34). ActiveX events are basically a function call with the parameters passed through an array of *wxVariants* along with a return value that is a *wxVariant* itself. What type the parameters or return value are depends on the context (i.e. what the .idl specifies).

Note that unlike the third party wxActiveX function names are not supported.

#### **Derived from**

*wxCommandEvent* (p. 250)

#### **Include files**

<wx/msw/ole/activex.h>

#### **Event table macros**

**EVT\_ACTIVEX(func)**

Sent when the activex control hosted by *wxActiveXContainer* (p. 34) recieves an activex event.

**wxActiveXEvent::ParamCount****size\_t ParamCount() const**

Obtains the number of parameters passed through the activex event.

**wxActiveXEvent::ParamType****wxString ParamType(size\_t idx) const**

Obtains the param type of the param number idx specifies as a string.

**wxActiveXEvent::ParamName****wxString ParamName(size\_t idx) const**

Obtains the param name of the param number idx specifies as a string.

**wxActiveXEvent::operator[]****wxVariant& operator[](size\_t idx)**

Obtains the actual parameter value specified by idx.

**wxActiveXEvent::GetDispatchId****DISPID GetDispatchId(int idx) const**

Returns the dispatch id of this activex event. This is the numeric value from the .idl file specified by the id().

**wxAnimation**

This class encapsulates the concept of a platform-dependent animation. An animation is a sequence of frames of the same size. Sound is not supported by wxAnimation.

**Derived from**

*wxGDIObject* (p. 709)  
*wxObject* (p. 1148)

**Include files**

<wx/animate.h>

**Predefined objects**

Objects:

**wxNullAnimation****See also**

*wxAnimationCtrl* (p. 42)

**wxAnimation::wxAnimation****wxAnimation()**

Default constructor.

**wxAnimation(const wxAnimation& anim)**

Copy constructor, uses *reference counting* (p. 2046).

**wxAnimation(const wxString& name, wxAnimationType type = wxANIMATION\_TYPE\_ANY)**

Loads an animation from a file.

*name*

The name of the file to load.

*type*

See *LoadFile* (p. 41) for more info.

**wxAnimation::~~wxAnimation****~wxAnimation()**

Destructor. See *reference-counted object destruction* (p. 2046) for more info.

**wxAnimation::GetDelay****int GetDelay(unsigned int i) const**

Returns the delay for the *i*-th frame in milliseconds. If *-1* is returned the frame is to be displayed forever.

**wxAnimation::GetFrameCount****unsigned int GetFrameCount() const**

Returns the number of frames for this animation.

**wxAnimation::GetFrame**



**wxImage GetFrame(unsigned int i) const**

Returns the *i*-th frame as a *wxImage* (p. 906).

**wxAnimation::GetSize****wxSize GetSize() const**

Returns the size of the animation.

**wxAnimation::IsOk****bool IsOk() const**

Returns `true` if animation data is present.

**wxAnimation::Load**

**bool Load(wxInputStream& stream, wxAnimationType type = wxANIMATION\_TYPE\_ANY)**

Loads an animation from the given stream.

**Parameters**

*stream*

The stream to use to load the animation.

*type*

One of the following values:

`wxANIMATION_TYPE_GIF` Load an animated GIF file.

`wxANIMATION_TYPE_ANI` Load an ANI file.

`wxANIMATION_TYPE_ANY` Try to autodetect the filetype.

**Return value**

`true` if the operation succeeded, `false` otherwise.

**wxAnimation::LoadFile**

**bool LoadFile(const wxString& name, wxAnimationType type = wxANIMATION\_TYPE\_ANY)**

Loads an animation from a file.

**Parameters**

*name*

A filename.

*type*

One of the following values:

`wxANIMATION_TYPE_GIF` Load an animated GIF file.

`wxANIMATION_TYPE_ANI` Load an ANI file.

`wxANIMATION_TYPE_ANY` Try to autodetect the filetype.

### Return value

`true` if the operation succeeded, `false` otherwise.

### **`wxAnimation::operator =`**

**`wxAnimation& operator =(const wxAnimation& brush)`**

Assignment operator, using *reference counting* (p. 2046).

## **wxAnimationCtrl**

This is a static control which displays an animation. `wxAnimationCtrl` API is simple as possible and won't give you full control on the animation; if you need it then use `wxMediaCtrl` (p. 1058).

This control is useful to display a (small) animation while doing a long task (e.g. a "throbber").

It is only available if `wxUSE_ANIMATIONCTRL` is set to 1 (the default).

### Derived from

`wxControl` (p. 285)

`wxWindow` (p. 1795)

`wxEvtHandler` (p. 576)

`wxObject` (p. 1148)

### Include files

`<wx/animate.h>`

### Window styles

**`wxAC_DEFAULT_STYLE`** The default style: `wxNO_BORDER`.

**`wxAC_NO_AUTORESIZE`** By default, the control will adjust its size to exactly fit to the size of the animation when `SetAnimation` (p. 44) is called. If

this style flag is given, the control will not change its size

### See also

*wxAnimation* (p. 39)

## **wxAnimationCtrl::wxAnimationCtrl**

**wxAnimationCtrl**(*wxWindow* \*parent, *wxWindowID* id, **const wxAnimation&** anim, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = *wxAC\_DEFAULT\_STYLE*, **const wxString&** name = "animationctrl")

Initializes the object and calls *Create* (p. 43) with all the parameters.

## **wxAnimationCtrl::Create**

**bool** Create(*wxWindow* \*parent, *wxWindowID* id, **const wxAnimation&** anim, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = *wxAC\_DEFAULT\_STYLE*, **const wxString&** name = "animationctrl")

### Parameters

*parent*

Parent window, must be non-NULL.

*id*

The identifier for the control.

*anim*

The initial animation shown in the control.

*pos*

Initial position.

*size*

Initial size.

*style*

The window style, see *wxAC\_\** flags.

*name*

Control name.

After control creation you must explicitly call *Play* (p. 44) to start to play the animation. Until that function won't be called, the first frame of the animation is displayed.

**Return value**

`true` if the control was successfully created or `false` if creation failed.

**wxAnimationCtrl::GetAnimation****wxAnimation GetAnimation() const**

Returns the animation associated with this control.

**wxAnimationCtrl::GetInactiveBitmap****wxBitmap GetInactiveBitmap() const**

Returns the inactive bitmap shown in this control when the; see *SetInactiveBitmap* (p. 44) for more info.

**wxAnimationCtrl::IsPlaying****bool IsPlaying() const**

Returns `true` if the animation is being played.

**wxAnimationCtrl::LoadFile****bool LoadFile(const wxString & file, wxAnimationType animType = wxANIMATION\_TYPE\_ANY)**

Loads the animation from the given file and calls *SetAnimation* (p. 44). See *wxAnimation::LoadFile* (p. 41) for more info.

**wxAnimationCtrl::Play****bool Play()**

Starts playing the animation. The animation is always played in loop mode (unless the last frame of the animation has an infinite delay time) and always start from the first frame (even if you *stopped* (p. 45) it while some other frame was displayed).

**wxAnimationCtrl::SetAnimation****void SetAnimation(const wxAnimation & anim)**

Sets the animation to play in this control. If the previous animation is being played, it's *Stopped* (p. 45).

Until *Play* (p. 44) isn't called, a static image, the first frame of the given animation or the background colour will be shown (see *SetInactiveBitmap* (p. 44) for more info).

**wxAnimationCtrl::SetInactiveBitmap**

**void SetInactiveBitmap(const wxBitmap& bmp)**

Sets the bitmap to show on the control when it's not playing an animation. If you set as inactive bitmap `wxNullBitmap` (which is the default), then the first frame of the animation is instead shown when the control is inactive; in this case, if there's no valid animation associated with the control (see *SetAnimation* (p. 44)), then the background colour of the window is shown.

If the control is not playing the animation, the given bitmap will be immediately shown, otherwise it will be shown as soon as *Stop* (p. 45) is called.

Note that the inactive bitmap, if smaller than the control's size, will be centered in the control; if bigger, it will be stretched to fit it.

**wxAnimationCtrl::Stop****void Stop()**

Stops playing the animation. The control will show the first frame of the animation, a custom static image or the window's background colour as specified by the last *SetInactiveBitmap* (p. 44) call.

**wxApp**

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

- set and get application-wide properties;
- implement the windowing system message or event loop;
- initiate application processing via *wxApp::OnInit* (p. 52);
- allow default processing of events not handled by other objects in the application.

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

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

**Derived from**

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

**Include files**

`<wx/app.h>`

**See also**

*wxApp overview* (p. 2040)

**wxApp::wxApp****wxApp()**

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

**wxApp::~~wxApp****virtual ~wxApp()**

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****wxChar \*\* argv**

Command line arguments (after environment-specific processing).

**wxApp::CreateLogTarget****virtual wxLog\* CreateLogTarget()**

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

**See also**

*wxLog* (p. 1018)

**wxApp::CreateTraits****virtual wxAppTraits \* CreateTraits()**

Creates the *wxAppTraits* (p. 56) object when *GetTraits* (p. 48) needs it for the first time.

**See also**

*wxAppTraits* (p. 56)

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

Dispatches the next event in the windowing system event queue.

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

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

#### See also

*wxApp::Pending* (p. 53)

### **wxApp::ExitMainLoop**

**virtual void ExitMainLoop()**

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

### **wxApp::FilterEvent**

**int FilterEvent(wxEvent& event)**

This function is called before processing any event and allows the application to preempt the processing of some events. If this method returns -1 the event is processed normally, otherwise either `true` or `false` should be returned and the event processing stops immediately considering that the event had been already processed (for the former return value) or that it is not going to be processed at all (for the latter one).

### **wxApp::GetAppName**

**wxString GetAppName() const**

Returns the application name.

#### Remarks

`wxWidgets` sets this to a reasonable default before calling *wxApp::OnInit* (p. 52), but the application can reset it at will.

### **wxApp::GetClassName**

**wxString GetClassName() const**

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

#### See also

*wxApp::SetClassName* (p. 54)

### **wxApp::GetExitOnFrameDelete**

**bool GetExitOnFrameDelete() const**

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

**See also**

*wxApp::SetExitOnFrameDelete* (p. 54),  
*wxApp shutdown overview* (p. 2041)

**wxApp::GetInstance****static wxAppConsole \* GetInstance()**

Returns the one and only global application object. Usually `wxTheApp` is used instead.

**See also**

*wxApp::SetInstance* (p. 54)

**wxApp::GetTopWindow****virtual wxWindow \* GetTopWindow() const**

Returns a pointer to the top window.

**Remarks**

If the top window hasn't been set using *wxApp::SetTopWindow* (p. 54), this function will find the first top-level window (frame or dialog) and return that.

**See also**

*SetTopWindow* (p. 54)

**wxApp::GetTraits****wxAppTraits \* GetTraits()**

Returns a pointer to the *wxAppTraits* (p. 56) object for the application. If you want to customize the *wxAppTraits* (p. 56) object, you must override the *CreateTraits* (p. 46) function.

**wxApp::GetUseBestVisual****bool GetUseBestVisual() const**

Returns true if the application will use the best visual on systems that support different visuals, false otherwise.

**See also**

*SetUseBestVisual* (p. 55)



**wxApp::GetVendorName****wxString GetVendorName() const**

Returns the application's vendor name.

**wxApp::IsActive****bool IsActive() const**

Returns `true` if the application is active, i.e. if one of its windows is currently in the foreground. If this function returns `false` and you need to attract users attention to the application, you may use `wxTopLevelWindow::RequestUserAttention` (p. 1716) to do it.

**wxApp::IsMainLoopRunning****static bool IsMainLoopRunning()**

Returns `true` if the main event loop is currently running, i.e. if the application is inside `OnRun` (p. 52).

This can be useful to test whether the events can be dispatched. For example, if this function returns `false`, non-blocking sockets cannot be used because the events from them would never be processed.

**wxApp::MainLoop****virtual int MainLoop()**

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

**Return value**

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

**wxApp::OnAssertFailure****void OnAssertFailure(const wxChar \*file, int line, const wxChar \*func, const wxChar \*cond, const wxChar \*msg)**

This function is called when an assert failure occurs, i.e. the condition specified in `wxASSERT` (p. 1980) macro evaluated to `false`. It is only called in debug mode (when `__WXDEBUG__` is defined) as asserts are not left in the release code at all.

The base class version shows the default assert failure dialog box proposing to the user to stop the program, continue or ignore all subsequent asserts.

**Parameters**

*file*

the name of the source file where the assert occurred

*line*

the line number in this file where the assert occurred

*func*

the name of the function where the assert occurred, may be empty if the compiler doesn't support C99 `__FUNCTION__`

*cond*

the condition of the failed assert in text form

*msg*

the message specified as argument to `wxASSERT_MSG` (p. 1980) or `wxFail_MSG` (p. 1981), will be `NULL` if just `wxASSERT` (p. 1980) or `wxFail` (p. 1981) was used

### **wxApp::OnCmdLineError**

**bool OnCmdLineError(wxCmdLineParser& parser)**

Called when command line parsing fails (i.e. an incorrect command line option was specified by the user). The default behaviour is to show the program usage text and abort the program.

Return `true` to continue normal execution or `false` to return `false` from `OnInit` (p. 52) thus terminating the program.

#### **See also**

*OnInitCmdLine* (p. 52)

### **wxApp::OnCmdLineHelp**

**bool OnCmdLineHelp(wxCmdLineParser& parser)**

Called when the help option (`--help`) was specified on the command line. The default behaviour is to show the program usage text and abort the program.

Return `true` to continue normal execution or `false` to return `false` from `OnInit` (p. 52) thus terminating the program.

#### **See also**

*OnInitCmdLine* (p. 52)

### **wxApp::OnCmdLineParsed**

**bool OnCmdLineParsed(wxCmdLineParser& parser)**

Called after the command line had been successfully parsed. You may override this method to test for the values of the various parameters which could be set from the command line.

Don't forget to call the base class version unless you want to suppress processing of the standard command line options.

Return `true` to continue normal execution or `false` to return `false` from *OnInit* (p. 52) thus terminating the program.

### See also

*OnInitCmdLine* (p. 52)

## **wxApp::OnExceptionInMainLoop**

### **virtual bool OnExceptionInMainLoop()**

This function is called if an unhandled exception occurs inside the main application event loop. It can return `true` to ignore the exception and to continue running the loop or `false` to exit the loop and terminate the program. In the latter case it can also use C++ `throw` keyword to rethrow the current exception.

The default behaviour of this function is the latter in all ports except under Windows where a dialog is shown to the user which allows him to choose between the different options. You may override this function in your class to do something more appropriate.

Finally note that if the exception is rethrown from here, it can be caught in *OnUnhandledException* (p. 52).

## **wxApp::OnExit**

### **virtual int OnExit()**

Override this member function for any processing which needs to be done as the application is about to exit. *OnExit* is called after destroying all application windows and controls, but before *wxWidgets* cleanup. Note that it is not called at all if *OnInit* (p. 52) failed.

The return value of this function is currently ignored, return the same value as returned by the base class method if you override it.

## **wxApp::OnFatalException**

### **void OnFatalException()**

This function may be called if something fatal happens: an unhandled exception under Win32 or a fatal signal under Unix, for example. However, this will not happen by default: you have to explicitly call *wxHandleFatalExceptions* (p. 1911) to enable this.

Generally speaking, this function should only show a message to the user and return. You may attempt to save unsaved data but this is not guaranteed to work and, in fact, probably

won't.

### See also

*wxHandleFatalExceptions* (p. 1911)

## **wxApp::OnInit**

### **bool OnInit()**

This must be provided by the application, and will usually create the application's main window, optionally calling *wxApp::SetTopWindow* (p. 54). You may use *OnExit* (p. 51) to clean up anything initialized here, provided that the function returns `true`.

Notice that if you want to use the command line processing provided by *wxWidgets* you have to call the base class version in the derived class *OnInit()*.

Return `true` to continue processing, `false` to exit the application immediately.

## **wxApp::OnInitCmdLine**

### **void OnInitCmdLine(wxCmdLineParser& parser)**

Called from *OnInit* (p. 52) and may be used to initialize the parser with the command line options for this application. The base class version adds support for a few standard options only.

## **wxApp::OnRun**

### **virtual int OnRun()**

This virtual function is where the execution of a program written in *wxWidgets* starts. The default implementation just enters the main loop and starts handling the events until it terminates, either because *ExitMainLoop* (p. 47) has been explicitly called or because the last frame has been deleted and *GetExitOnFrameDelete* (p. 47) flag is `true` (this is the default).

The return value of this function becomes the exit code of the program, so it should return 0 in case of successful termination.

## **wxApp::OnUnhandledException**

### **virtual void OnUnhandledException()**

This function is called when an unhandled C++ exception occurs inside *OnRun()* (p. 52) (the exceptions which occur during the program startup and shutdown might not be caught at all). Note that the exception type is lost by now, so if you want to really handle the exception you should override *OnRun()* (p. 52) and put a try/catch clause around the call to the base class version there.

## **wxApp::ProcessMessage**

**bool ProcessMessage(WXMSG \*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 wxWidgets with another class library with its own message loop, you should make sure that this function is called to allow wxWidgets to receive messages. For example, to allow co-existence with the Microsoft Foundation Classes, override the `PreTranslateMessage` function:

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

**wxApp::Pending****virtual bool Pending()**

Returns true if unprocessed events are in the window system event queue.

**See also**

*wxApp::Dispatch* (p. 46)

**wxApp::SendIdleEvents****bool SendIdleEvents(wxWindow\* win, wxIdleEvent& event)**

Sends idle events to a window and its children.

Please note that this function is internal to wxWidgets and shouldn't be used by user code.

**Remarks**

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

**See also**

*wxIdleEvent* (p. 903)

**wxApp::SetAppName****void SetAppName(const wxString& name)**

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

**See also**

*wxApp::GetAppName* (p. 47)

### **wxApp::SetClassName**

**void SetClassName(const wxString& name)**

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

#### **See also**

*wxApp::GetClassName* (p. 47)

### **wxApp::SetExitOnFrameDelete**

**void SetExitOnFrameDelete(bool flag)**

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

#### **Parameters**

*flag*

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

#### **See also**

*wxApp::GetExitOnFrameDelete* (p. 47),  
*wxApp shutdown overview* (p. 2041)

### **wxApp::SetInstance**

**static void SetInstance(wxAppConsole\* app)**

Allows external code to modify global `wxTheApp`, but you should really know what you're doing if you call it.

#### **Parameters**

*app*

Replacement for the global application object.

#### **See also**

*wxApp::GetInstance* (p. 48)

### **wxApp::SetTopWindow**

**void SetTopWindow(wxWindow\* window)**

Sets the 'top' window. You can call this from within *wxApp::OnInit* (p. 52) to let *wxWidgets* know which is the main window. You don't have to set the top window; it is only a convenience so that (for example) certain dialogs without parents can use a specific window as the top window. If no top window is specified by the application, *wxWidgets* just uses the first frame or dialog in its top-level window list, when it needs to use the top window.

**Parameters**

*window*

The new top window.

**See also**

*wxApp::GetTopWindow* (p. 48), *wxApp::OnInit* (p. 52)

**wxApp::SetVendorName**

**void SetVendorName(const wxString& name)**

Sets the name of application's vendor. The name will be used in registry access. A default name is set by *wxWidgets*.

**See also**

*wxApp::GetVendorName* (p. 49)

**wxApp::SetUseBestVisual**

**void SetUseBestVisual(bool flag, bool forceTrueColour = false)**

Allows the programmer to specify whether the application will use the best visual on systems that support several visual on the same display. This is typically the case under Solaris and IRIX, where the default visual is only 8-bit whereas certain applications are supposed to run in TrueColour mode.

If *forceTrueColour* is true then the application will try to force using a TrueColour visual and abort the app if none is found.

Note that this function has to be called in the constructor of the *wxApp* instance and won't have any effect when called later on.

This function currently only has effect under GTK.

**Parameters**

*flag*

If true, the app will use the best visual.

**wxApp::HandleEvent**

**virtual void HandleEvent(wxEvtHandler \*handler, wxEventFunction func, wxEvent& event) const**

This function simply invokes the given method *func* of the specified event handler *handler* with the *event* as parameter. It exists solely to allow to catch the C++ exceptions which could be thrown by all event handlers in the application in one place: if you want to do this, override this function in your wxApp-derived class and add try/catch clause(s) to it.

## **wxApp::Yield**

**bool Yield(bool onlyIfNeeded = false)**

Yields control to pending messages in the windowing system. This can be useful, for example, when a time-consuming process writes to a text window. Without an occasional yield, the text window will not be updated properly, and on systems with cooperative multitasking, such as Windows 3.1 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: see `::wxSafeYield` (p. 1912) for a better function.

Note that `Yield()` will not flush the message logs. This is intentional as calling `Yield()` is usually done to quickly update the screen and popping up a message box dialog may be undesirable. If you do wish to flush the log messages immediately (otherwise it will be done during the next idle loop iteration), call `wxLog::FlushActive` (p. 1023).

Calling `Yield()` recursively is normally an error and an assert failure is raised in debug build if such situation is detected. However if the *onlyIfNeeded* parameter is `true`, the method will just silently return `false` instead.

## **wxAppTraits**

The **wxAppTraits** class defines various configurable aspects of a *wxApp* (p. 45). You can access it using `wxApp::GetTraits` (p. 48) function and you can create your own *wxAppTraits* (p. 56) overriding the `wxApp::CreateTraits` (p. 46) function.

By default, `wxWidgets` creates a `wxConsoleAppTraits` object for console applications (i.e. those applications linked against `wxBase` library only - see the *Libraries list* (p. 15) page) and a `wxGUIAppTraits` object for GUI applications.

### **Derived from**

None

### **Include files**

<wx/apptrait.h>

### **See also**

*wxApp overview* (p. 2040), *wxApp* (p. 45)



**wxAppTraits::CreateFontMapper****virtual wxFontMapper \* CreateFontMapper()**

Creates the global font mapper object used for encodings/charset mapping.

**wxAppTraits::CreateLogTarget****virtual wxLog \* CreateLogTarget()**

Creates the default log target for the application.

**wxAppTraits::CreateMessageOutput****virtual wxMessageOutput \* CreateMessageOutput()**

Creates the global object used for printing out messages.

**wxAppTraits::CreateRenderer****virtual wxRendererNative \* CreateRenderer()**

Returns the renderer to use for drawing the generic controls (return value may be `NULL` in which case the default renderer for the current platform is used); this is used in GUI mode only and always returns `NULL` in console.

NOTE: returned pointer will be deleted by the caller.

**wxAppTraits::GetDesktopEnvironment****virtual wxString GetDesktopEnvironment() const**

This method returns the name of the desktop environment currently running in a Unix desktop. Currently only "KDE" or "GNOME" are supported and the code uses the X11 session protocol vendor name to figure out, which desktop environment is running. The method returns an empty string otherwise and on all other platforms.

**wxAppTraits::GetStandardPaths****virtual wxStandardPaths & GetStandardPaths()**

Returns the `wxStandardPaths` object for the application. It's normally the same for `wxBase` and `wxGUI` except in the case of `wxMac` and `wxCocoa`.

**wxAppTraits::GetToolkitVersion****virtual wxPortId GetToolkitVersion(int \*major = NULL, int \*minor = NULL)**

Returns the `wxWidgets` port ID used by the running program and eventually fills the given pointers with the values of the major and minor digits of the native toolkit currently used. The version numbers returned are thus detected at run-time and not compile-time (except when this is not possible e.g. `wxMotif`).

E.g. if your program is using `wxGTK` port this function will return `wxPORT_GTK` and put in given pointers the versions of the `GTK` library in use.

See *wxPlatformInfo* (p. 1185) for more details.

### **wxAppTraits::HasStderr**

**virtual bool HasStderr()**

Returns `true` if `fprintf(stderr)` goes somewhere, `false` otherwise.

### **wxAppTraits::IsUsingUniversalWidgets**

**bool IsUsingUniversalWidgets() const**

Returns `true` if the library was built as `wxUniversal`. Always returns `false` for `wxBase`-only apps.

### **wxAppTraits::ShowAssertDialog**

**virtual bool ShowAssertDialog(const wxString & msg)**

Shows the assert dialog with the specified message in GUI mode or just prints the string to `stderr` in console mode.

Returns `true` to suppress subsequent asserts, `false` to continue as before.

## **wxArchiveClassFactory**

Allows the creation of streams to handle archive formats such as `zip` and `tar`.

For example, given a filename you can search for a factory that will handle it and create a stream to read it:

```
factory = wxArchiveClassFactory::Find(filename,
wxSTREAM_FILEEXT);
if (factory)
    stream = factory->NewStream(new
wxFFileInputStream(filename));
```

*Find()* (p. 59) can also search for a factory by MIME type or `wxFileSystem` protocol. The available factories can be enumerated using *GetFirst()* and *GetNext()* (p. 60).

### **Derived from**

*wxObject* (p. 1148)

### Include files

<wx/archive.h>

### Data structures

```
enum wxStreamProtocolType
{
    wxSTREAM_PROTOCOL, // wxFileSystem protocol (should be only one)
    wxSTREAM_MIMETYPE, // MIME types the stream handles
    wxSTREAM_ENCODING, // Not used for archives
    wxSTREAM_FILEEXT   // File extensions the stream handles
};
```

### See also

*Archive formats such as zip* (p. 2223)

*Generic archive programming* (p. 2227)

*wxArchiveEntry* (p. 62)

*wxArchiveInputStream* (p. 64)

*wxArchiveOutputStream* (p. 69)

*wxFilterClassFactory* (p. 643)

### **wxArchiveClassFactory::Get/SetConv**

**wxMBConv& GetConv() const**

**void SetConv(wxMBConv& conv)**

The *wxMBConv* (p. 1038) object that the created streams will use when translating meta-data. The initial default, set by the constructor, is *wxConvLocal*.

### **wxArchiveClassFactory::CanHandle**

**bool CanHandle(const wxChar\* protocol, wxStreamProtocolType type = wxSTREAM\_PROTOCOL) const**

Returns true if this factory can handle the given protocol, MIME type or file extension.

When using *wxSTREAM\_FILEEXT* for the second parameter, the first parameter can be a complete filename rather than just an extension.

### **wxArchiveClassFactory::Find**

**static const wxArchiveClassFactory\* Find(const wxChar\* protocol, wxStreamProtocolType type = wxSTREAM\_PROTOCOL)**

A static member that finds a factory that can handle a given protocol, MIME type or file

extension. Returns a pointer to the class factory if found, or NULL otherwise. It does not give away ownership of the factory.

When using `wxSTREAM_FILEEXT` for the second parameter, the first parameter can be a complete filename rather than just an extension.

### **wxArchiveClassFactory::GetFirst/GetNext**

**static const wxArchiveClassFactory\* GetFirst()**

**const wxArchiveClassFactory\* GetNext() const**

`GetFirst` and `GetNext` can be used to enumerate the available factories.

For example, to list them:

```
wxString list;
const wxArchiveClassFactory *factory =
wxArchiveClassFactory::GetFirst();

while (factory) {
    list << factory->GetProtocol() << _T("\n");
    factory = factory->GetNext();
}
```

`GetFirst()/GetNext()` return a pointer to a factory or NULL if no more are available. They do not give away ownership of the factory.

### **wxArchiveClassFactory::GetInternalName**

**wxString GetInternalName(const wxString& name, wxPathFormat format =  
wxPATH\_NATIVE) const**

Calls the static `GetInternalName()` function for the archive entry type, for example `wxZipEntry::GetInternalName()` (p. 1892).

### **wxArchiveClassFactory::GetProtocol**

**wxString GetProtocol() const**

Returns the `wxFileSystem` protocol supported by this factory. Equivalent to `wxString(*GetProtocols())`.

### **wxArchiveClassFactory::GetProtocols**

**const wxChar \* const\* GetProtocols(wxStreamProtocolType type =  
wxSTREAM\_PROTOCOL) const**

Returns the protocols, MIME types or file extensions supported by this factory, as an array of null terminated strings. It does not give away ownership of the array or strings.

For example, to list the file extensions a factory supports:

```
wxString list;
const wxChar *const *p;

for (p = factory->GetProtocols(wxSTREAM_FILEEXT); *p; p++)
    list << *p << _T("\n");
```

### **wxArchiveClassFactory::NewEntry**

**wxArchiveEntry\* NewEntry() const**

Create a new *wxArchiveEntry* (p. 62) object of the appropriate type.

### **wxArchiveClassFactory::NewStream**

**wxArchiveInputStream\* NewStream(wxInputStream& stream) const**

**wxArchiveOutputStream\* NewStream(wxOutputStream& stream) const**

**wxArchiveInputStream\* NewStream(wxInputStream\* stream) const**

**wxArchiveOutputStream\* NewStream(wxOutputStream\* stream) const**

Create a new input or output stream to read or write an archive.

If the parent stream is passed as a pointer then the new archive stream takes ownership of it. If it is passed by reference then it does not.

### **wxArchiveClassFactory::PushFront**

**void PushFront()**

Adds this class factory to the list returned by *GetFirst()/GetNext()* (p. 60).

It is not necessary to do this to use the archive streams. It is usually used when implementing streams, typically the implementation will add a static instance of its factory class.

It can also be used to change the order of a factory already in the list, bringing it to the front. This isn't a thread safe operation so can't be done when other threads are running that will be using the list.

The list does not take ownership of the factory.

### **wxArchiveClassFactory::Remove**

**void Remove()**

Removes this class factory from the list returned by *GetFirst()/GetNext()* (p. 60).

Removing from the list isn't a thread safe operation so can't be done when other threads

are running that will be using the list.

The list does not own the factories, so removing a factory does not delete it.

## **wxArchiveEntry**

An abstract base class which serves as a common interface to archive entry classes such as *wxZipEntry* (p. 1888). These hold the meta-data (filename, timestamp, etc.), for entries in archive files such as zips and tars.

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/archive.h>

### **See also**

*Archive formats such as zip* (p. 2223)

*Generic archive programming* (p. 2227)

*wxArchiveInputStream* (p. 64)

*wxArchiveOutputStream* (p. 69)

*wxArchiveNotifier* (p. 68)

### **Non-seekable streams**

This information applies only when reading archives from non-seekable streams. When the stream is seekable *GetNextEntry()* (p. 65) returns a fully populated *wxArchiveEntry* (p. 62). See '*Archives on non-seekable streams* (p. 2228)' for more information.

For generic programming, when the worst case must be assumed, you can rely on all the fields of *wxArchiveEntry* being fully populated when *GetNextEntry()* returns, with the the following exceptions:

*GetSize()* (p. 63) Guaranteed to be available after the entry has been read to *Eof()* (p. 942), or *CloseEntry()* (p. 65) has been called

*IsReadOnly()* (p. 64) Guaranteed to be available after the end of the archive has been reached, i.e. after *GetNextEntry()* returns NULL and *Eof()* is true

### **wxArchiveEntry::Clone**

**wxArchiveEntry\* Clone() const**

Returns a copy of this entry object.

### **wxArchiveEntry::Get/SetDateTime**

**wxDateTime GetDateTime() const**

**void SetDateTime(const wxDateTime& dt)**

The entry's timestamp.

**wxArchiveEntry::GetInternalFormat**

**wxPathFormat GetInternalFormat() const**

Returns the path format used internally within the archive to store filenames.

**wxArchiveEntry::GetInternalName**

**wxString GetInternalName() const**

Returns the entry's filename in the internal format used within the archive. The name can include directory components, i.e. it can be a full path.

The names of directory entries are returned without any trailing path separator. This gives a canonical name that can be used in comparisons.

**See also**

*Looking up an archive entry by name* (p. 2225)

**wxArchiveEntry::Get/SetName**

**wxString GetName(wxPathFormat format = wxPATH\_NATIVE) const**

**void SetName(const wxString& name, wxPathFormat format = wxPATH\_NATIVE)**

The entry's name, by default in the native format. The name can include directory components, i.e. it can be a full path.

If this is a directory entry, (i.e. if *IsDir()* (p. 64) is true) then *GetName()* returns the name with a trailing path separator.

Similarly, setting a name with a trailing path separator sets *IsDir()*.

**wxArchiveEntry::GetOffset**

**off\_t GetOffset() const**

Returns a numeric value unique to the entry within the archive.

**wxArchiveEntry::Get/SetSize**

**off\_t GetSize() const**

**void SetSize(off\_t size)**

The size of the entry's data in bytes.

### **wxArchiveEntry::IsDir/SetIsDir**

**bool IsDir() const**

**void SetIsDir(bool isDir = true)**

True if this is a directory entry.

Directory entries are entries with no data, which are used to store the meta-data of directories. They also make it possible for completely empty directories to be stored.

The names of entries within an archive can be complete paths, and unarchivers typically create whatever directories are necessary as they restore files, even if the archive contains no explicit directory entries.

### **wxArchiveEntry::IsReadOnly/SetIsReadOnly**

**bool IsReadOnly() const**

**void SetIsReadOnly(bool isReadOnly = true)**

True if the entry is a read-only file.

### **wxArchiveEntry::Set/UnsetNotifier**

**void SetNotifier(wxArchiveNotifier& notifier)**

**void UnsetNotifier()**

Sets the *notifier* (p. 68) for this entry. Whenever the *wxArchiveInputStream* (p. 64) updates this entry, it will then invoke the associated notifier's *OnEntryUpdated* (p. 68) method.

Setting a notifier is not usually necessary. It is used to handle certain cases when modifying an archive in a pipeline (i.e. between non-seekable streams).

### **See also**

*Archives on non-seekable streams* (p. 2228)

*wxArchiveNotifier* (p. 68)

## **wxArchiveInputStream**

An abstract base class which serves as a common interface to archive input streams such as *wxZipInputStream* (p. 1895).

*GetNextEntry()* (p. 65) returns an *wxArchiveEntry* (p. 62) object containing the meta-data for the next entry in the archive (and gives away ownership). Reading from the *wxArchiveInputStream* then returns the entry's data. *Eof()* becomes true after an attempt has been made to read past the end of the entry's data. When there are no more entries,



GetNextEntry() returns NULL and sets Eof().

**Derived from**

*wxFilterInputStream* (p. 646)

**Include files**

<wx/archive.h>

**Data structures** typedef wxArchiveEntry entry\_type

**See also**

*Archive formats such as zip* (p. 2223)

*wxArchiveEntry* (p. 62)

*wxArchiveOutputStream* (p. 69)

**wxArchiveInputStream::CloseEntry**

**bool CloseEntry()**

Closes the current entry. On a non-seekable stream reads to the end of the current entry first.

**wxArchiveInputStream::GetNextEntry**

**wxArchiveEntry\* GetNextEntry()**

Closes the current entry if one is open, then reads the meta-data for the next entry and returns it in a *wxArchiveEntry* (p. 62) object, giving away ownership. Reading this *wxArchiveInputStream* then returns the entry's data.

**wxArchiveInputStream::OpenEntry**

**bool OpenEntry(wxArchiveEntry& entry)**

Closes the current entry if one is open, then opens the entry specified by the *wxArchiveEntry* (p. 62) object.

*entry* must be from the same archive file that this *wxArchiveInputStream* is reading, and it must be reading it from a seekable stream.

**See also**

*Looking up an archive entry by name* (p. 2225)

**wxArchivewriterator**

An input iterator template class that can be used to transfer an archive's catalogue to a

container. It is only available if `wxUSE_STL` is set to 1 in `setup.h`, and the uses for it outlined below require a compiler which supports member templates.

```
template <class Arc, class T = typename Arc::entry_type*>
class wxArchiveIterator
{
    // this constructor creates an 'end of sequence' object
    wxArchiveIterator();

    // template parameter 'Arc' should be the type of an archive input
    stream
    wxArchiveIterator(Arc& arc) {

        /* ... */
    };
};
```

The first template parameter should be the type of archive input stream (e.g. *wxArchiveInputStream* (p. 64)) and the second can either be a pointer to an entry (e.g. *wxArchiveEntry* (p. 62)\*), or a string/pointer pair (e.g. `std::pair<wxString, wxArchiveEntry*>`).

The `<wx/archive.h>` header defines the following typedefs:

```
typedef wxArchiveIterator<wxArchiveInputStream> wxArchiveIter;

typedef wxArchiveIterator<wxArchiveInputStream,
    std::pair<wxString, wxArchiveEntry*> >
wxArchivePairIter;
```

The header for any implementation of this interface should define similar typedefs for its types, for example in `<wx/zipstrm.h>` there is:

```
typedef wxArchiveIterator<wxZipInputStream> wxZipIter;

typedef wxArchiveIterator<wxZipInputStream,
    std::pair<wxString, wxZipEntry*> > wxZipPairIter;
```

Transferring the catalogue of an archive *arc* to a vector *cat*, can then be done something like this:

```
std::vector<wxArchiveEntry*> cat((wxArchiveIter)arc,
    wxArchiveIter());
```

When the iterator is dereferenced, it gives away ownership of an entry object. So in the above example, when you have finished with *cat* you must delete the pointers it contains.

If you have smart pointers with normal copy semantics (i.e. not `auto_ptr` or *wxScopedPtr* (p. 1404)), then you can create an iterator which uses them instead. For example, with a smart pointer class for zip entries *ZipEntryPtr*:

```
typedef std::vector<ZipEntryPtr> ZipCatalog;
```

```
typedef wxArchiveIterator<wxZipInputStream, ZipEntryPtr>
ZipIter;
ZipCatalog cat((ZipIter)zip, ZipIter());
```

Iterators that return `std::pair` objects can be used to populate a `std::multimap`, to allow entries to be looked up by name. The string is initialised using the `wxArchiveEntry` object's `GetInternalName()` (p. 63) function.

```
typedef std::multimap<wxString, wxZipEntry*> ZipCatalog;
ZipCatalog cat((wxZipPairIter)zip, wxZipPairIter());
```

Note that this iterator also gives away ownership of an entry object each time it is dereferenced. So in the above example, when you have finished with *cat* you must delete the pointers it contains.

Or if you have them, a pair containing a smart pointer can be used (again *ZipEntryPtr*), no worries about ownership:

```
typedef std::multimap<wxString, ZipEntryPtr> ZipCatalog;
typedef wxArchiveIterator<wxZipInputStream,
    std::pair<wxString, ZipEntryPtr> > ZipPairIter;
ZipCatalog cat((ZipPairIter)zip, ZipPairIter());
```

## Derived from

No base class

## Include files

<wx/archive.h>

## See also

*wxArchiveEntry* (p. 62)

*wxArchiveInputStream* (p. 64)

*wxArchiveOutputStream* (p. 69)

```
Data structurestypedef std::input_iterator_tag iterator_category
typedef T value_type
typedef ptrdiff_t difference_type
typedef T* pointer
typedef T& reference
```

## wxArchiveIterator::wxArchiveIterator

### wxArchiveIterator()

Construct an 'end of sequence' instance.

**wxArchiveIterator(Arc& arc)**

Construct iterator that returns all the entries in the archive input stream *arc*.

**wxArchiveIterator::operator\*****const T& operator\*() const**

Returns an entry object from the archive input stream, giving away ownership.

**wxArchiveIterator::operator++****wxArchiveIterator& operator++()****wxArchiveIterator& operator++(int)**

Position the input iterator at the next entry in the archive input stream.

**wxArchiveNotifier**

If you need to know when a *wxArchiveInputStream* (p. 64) updates a *wxArchiveEntry* (p. 62) object, you can create a notifier by deriving from this abstract base class, overriding *OnEntryUpdated()* (p. 68). An instance of your notifier class can then be assigned to the *wxArchiveEntry* object using *wxArchiveEntry::SetNotifier()* (p. 64). Your *OnEntryUpdated()* method will then be invoked whenever the input stream updates the entry.

Setting a notifier is not usually necessary. It is used to handle certain cases when modifying an archive in a pipeline (i.e. between non-seekable streams). See *Archives on non-seekable streams* (p. 2228).

**Derived from**

No base class

**Include files**

<wx/archive.h>

**See also**

*Archives on non-seekable streams* (p. 2228)

*wxArchiveEntry* (p. 62)

*wxArchiveInputStream* (p. 64)

*wxArchiveOutputStream* (p. 69)

**wxArchiveNotifier::OnEntryUpdated**

**void OnEntryUpdated(class wxArchiveEntry& entry)**

This method must be overridden in your derived class.

## **wxArchiveOutputStream**

An abstract base class which serves as a common interface to archive output streams such as *wxZipOutputStream* (p. 1897).

*PutNextEntry()* (p. 70) is used to create a new entry in the output archive, then the entry's data is written to the *wxArchiveOutputStream*. Another call to *PutNextEntry()* closes the current entry and begins the next.

### **Derived from**

*wxFilterOutputStream* (p. 647)

### **Include files**

<wx/archive.h>

### **See also**

*Archive formats such as zip* (p. 2223)

*wxArchiveEntry* (p. 62)

*wxArchiveInputStream* (p. 64)

## **wxArchiveOutputStream::~wxArchiveOutputStream**

**~wxArchiveOutputStream()**

Calls *Close()* (p. 69) if it has not already been called.

## **wxArchiveOutputStream::Close**

**bool Close()**

Closes the archive, returning true if it was successfully written. Called by the destructor if not called explicitly.

## **wxArchiveOutputStream::CloseEntry**

**bool CloseEntry()**

Close the current entry. It is called implicitly whenever another new entry is created with *CopyEntry()* (p. 70) or *PutNextEntry()* (p. 70), or when the archive is closed.

## **wxArchiveOutputStream::CopyArchiveMetaData**

**bool CopyArchiveMetaData(wxArchiveInputStream& stream)**

Some archive formats have additional meta-data that applies to the archive as a whole. For example in the case of zip there is a comment, which is stored at the end of the zip file. `CopyArchiveMetaData()` can be used to transfer such information when writing a modified copy of an archive.

Since the position of the meta-data can vary between the various archive formats, it is best to call `CopyArchiveMetaData()` before transferring the entries. The `wxArchiveOutputStream` (p. 69) will then hold on to the meta-data and write it at the correct point in the output file.

When the input archive is being read from a non-seekable stream, the meta-data may not be available when `CopyArchiveMetaData()` is called, in which case the two streams set up a link and transfer the data when it becomes available.

### **wxArchiveOutputStream::CopyEntry**

**bool CopyEntry(wxArchiveEntry\* entry, wxArchiveInputStream& stream)**

Takes ownership of *entry* and uses it to create a new entry in the archive. *entry* is then opened in the input stream *stream* and its contents copied to this stream.

For archive types which compress entry data, `CopyEntry()` is likely to be much more efficient than transferring the data using `Read()` and `Write()` since it will copy them without decompressing and recompressing them.

*entry* must be from the same archive file that *stream* is accessing. For non-seekable streams, *entry* must also be the last thing read from *stream*.

### **wxArchiveOutputStream::PutNextDirEntry**

**bool PutNextDirEntry(const wxString& name, const wxDateTime& dt = wxDateTime::Now())**

Create a new directory entry (see `wxArchiveEntry::IsDir()` (p. 64)) with the given name and timestamp.

`PutNextEntry()` (p. 70) can also be used to create directory entries, by supplying a name with a trailing path separator.

### **wxArchiveOutputStream::PutNextEntry**

**bool PutNextEntry(wxArchiveEntry\* entry)**

Takes ownership of *entry* and uses it to create a new entry in the archive. The entry's data can then be written by writing to this `wxArchiveOutputStream`.

**bool PutNextEntry(const wxString& name, const wxDateTime& dt = wxDateTime::Now(), off\_t size = wxInvalidOffset)**

Create a new entry with the given name, timestamp and size. The entry's data can then be written by writing to this `wxArchiveOutputStream`.

## wxArray

This section describes the so called *dynamic arrays*. This is a C array-like data structure i.e. the member access time is constant (and not linear according to the number of container elements as for linked lists). However, these arrays are dynamic in the sense that they will automatically allocate more memory if there is not enough of it for adding a new element. They also perform range checking on the index values but in debug mode only, so please be sure to compile your application in debug mode to use it (see *debugging overview* (p. 2072) for details). So, unlike the arrays in some other languages, attempt to access an element beyond the arrays bound doesn't automatically expand the array but provokes an assertion failure instead in debug build and does nothing (except possibly crashing your program) in the release build.

The array classes were designed to be reasonably efficient, both in terms of run-time speed and memory consumption and the executable size. The speed of array item access is, of course, constant (independent of the number of elements) making them much more efficient than linked lists (*wxList* (p. 966)). Adding items to the arrays is also implemented in more or less constant time - but the price is preallocating the memory in advance. In the *memory management* (p. 74) section you may find some useful hints about optimizing wxArray memory usage. As for executable size, all wxArray functions are inline, so they do not take *any space at all*.

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

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

wxSortedArray is a wxArray variant which should be used when searching in the array is a frequently used operation. It requires you to define an additional function for comparing two elements of the array element type and always stores its items in the sorted order (according to this function). Thus, it is *Index()* (p. 80) function execution time is  $O(\log(N))$  instead of  $O(N)$  for the usual arrays but the *Add()* (p. 79) method is slower: it is  $O(\log(N))$  instead of constant time (neglecting time spent in memory allocation routine). However, in

a usual situation elements are added to an array much less often than searched inside it, so `wxSortedArray` may lead to huge performance improvements compared to `wxArray`. Finally, it should be noticed that, as `wxArray`, `wxSortedArray` can be only used for storing integral types or pointers.

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

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

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

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

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

...

// now that we have MyDirectory declaration in scope we may finish the
// definition of ArrayOfDirectories -- note that this expands into some
C++
// code and so should only be compiled once (i.e., don't put this in
the
// header, but into a source file or you will get linking errors)
#include <wx/arrimpl.cpp> // this is a magic incantation which must
be done!
WX_DEFINE_OBJARRAY(ArrayOfDirectories);

// that's all!
```

It is not as elegant as writing

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

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



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

```
WX_DEFINE_ARRAY_INT(int, ArrayOfInts);
WX_DEFINE_SORTED_ARRAY_INT(int, ArrayOfSortedInts);
```

i.e. there is only one `DEFINE` macro and no need for separated `DECLARE` one. For the arrays of the primitive types, the macros

`WX_DEFINE_ARRAY_CHAR/SHORT/INT/SIZE_T/LONG/DOUBLE` should be used depending on the `sizeof` of the values (notice that storing values of smaller type, e.g. shorts, in an array of larger one, e.g. `ARRAY_INT`, does *not* work on all architectures!).

**See also:**

*Container classes overview* (p. 2066), *wxList* (p. 966)

**Include files**

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

**Macros for template array definition**

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

`WX_DEFINE_ARRAY` (p. 75)  
`WX_DEFINE_EXPORTED_ARRAY` (p. 75)  
`WX_DEFINE_USER_EXPORTED_ARRAY` (p. 75)  
`WX_DEFINE_SORTED_ARRAY` (p. 76)  
`WX_DEFINE_SORTED_EXPORTED_ARRAY` (p. 76)  
`WX_DEFINE_SORTED_USER_EXPORTED_ARRAY` (p. 76)  
`WX_DECLARE_EXPORTED_OBJARRAY` (p. 76)  
`WX_DECLARE_USER_EXPORTED_OBJARRAY` (p. 76)  
`WX_DEFINE_OBJARRAY` (p. 77)  
`WX_DEFINE_EXPORTED_OBJARRAY` (p. 77)  
`WX_DEFINE_USER_EXPORTED_OBJARRAY` (p. 77)

To slightly complicate the matters even further, the operator `->` defined by default for the array iterators by these macros only makes sense if the array element type is not a pointer itself and, although it still works, this provokes warnings from some compilers and to avoid them you should use the `_PTR` versions of the macros above. For example, to define an array of pointers to `double` you should use:

```
WX_DEFINE_ARRAY_PTR(double *, MyArrayOfDoublePointers);
```

Note that the above macros are generally only useful for `wxObject` types. There are

separate macros for declaring an array of a simple type, such as an int.

The following simple types are supported:

int  
long  
size\_t  
double

To create an array of a simple type, simply append the type you want in CAPS to the array definition.

For example, for an integer array, you'd use one of the following variants:

*WX\_DEFINE\_ARRAY\_INT* (p. 75)  
*WX\_DEFINE\_EXPORTED\_ARRAY\_INT* (p. 75)  
*WX\_DEFINE\_USER\_EXPORTED\_ARRAY\_INT* (p. 75)  
*WX\_DEFINE\_SORTED\_ARRAY\_INT* (p. 76)  
*WX\_DEFINE\_SORTED\_EXPORTED\_ARRAY\_INT* (p. 76)  
*WX\_DEFINE\_SORTED\_USER\_EXPORTED\_ARRAY\_INT* (p. 76)

## Constructors and destructors

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

*wxArray* default constructor (p. 78)  
*wxArray* copy constructors and assignment operators (p. 78)  
*~wxArray* (p. 78)

## Memory management

Automatic array memory management is quite trivial: the array starts by preallocating some minimal amount of memory (defined by `WX_ARRAY_DEFAULT_INITIAL_SIZE`) and when further new items exhaust already allocated memory it reallocates it adding 50% of the currently allocated amount, but no more than some maximal number which is defined by `ARRAY_MAXSIZE_INCREMENT` constant. Of course, this may lead to some memory being wasted (`ARRAY_MAXSIZE_INCREMENT` in the worst case, i.e. 4Kb in the current implementation), so the *Shrink()* (p. 82) function is provided to deallocate the extra memory. The *Alloc()* (p. 79) function can also be quite useful if you know in advance how many items you are going to put in the array and will prevent the array code from reallocating the memory more times than needed.

*Alloc* (p. 79)  
*Shrink* (p. 82)

## Number of elements and simple item access

Functions in this section return the total number of array elements and allow to retrieve them - possibly using just the C array indexing [] operator which does exactly the same as *Item()* (p. 81) method.

*Count* (p. 79)

*GetCount* (p. 80)

*IsEmpty* (p. 81)

*Item* (p. 81)

*Last* (p. 81)

## Adding items

*Add* (p. 79)

*Insert* (p. 80)

*SetCount* (p. 82)

*WX\_APPEND\_ARRAY* (p. 77)

*WX\_PREPEND\_ARRAY* (p. 78)

## Removing items

*WX\_CLEAR\_ARRAY* (p. 78)

*Empty* (p. 80)

*Clear* (p. 79)

*RemoveAt* (p. 82)

*Remove* (p. 81)

## Searching and sorting

*Index* (p. 80)

*Sort* (p. 82)

## WX\_DEFINE\_ARRAY

**WX\_DEFINE\_ARRAY**(*T*, *name*)

**WX\_DEFINE\_EXPORTED\_ARRAY**(*T*, *name*)

**WX\_DEFINE\_USER\_EXPORTED\_ARRAY**(*T*, *name*, *exportspec*)

This macro defines a new array class named *name* and containing the elements of type *T*. The second form is used when compiling wxWidgets as a DLL under Windows and array needs to be visible outside the DLL. The third is needed for exporting an array from a user DLL.

Example:

```
WX_DEFINE_ARRAY_INT(int, MyArrayInt);
```

```
class MyClass;
WX_DEFINE_ARRAY(MyClass *, ArrayOfMyClass);
```

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

## **WX\_DEFINE\_SORTED\_ARRAY**

**WX\_DEFINE\_SORTED\_ARRAY**(*T*, *name*)

**WX\_DEFINE\_SORTED\_EXPORTED\_ARRAY**(*T*, *name*)

**WX\_DEFINE\_SORTED\_USER\_EXPORTED\_ARRAY**(*T*, *name*)

This macro defines a new sorted array class named *name* and containing the elements of type *T*. The second form is used when compiling wxWidgets as a DLL under Windows and array needs to be visible outside the DLL. The third is needed for exporting an array from a user DLL.

Example:

```
WX_DEFINE_SORTED_ARRAY_INT(int, MySortedArrayInt);

class MyClass;
WX_DEFINE_SORTED_ARRAY(MyClass *, ArrayOfMyClass);
```

You will have to initialize the objects of this class by passing a comparison function to the array object constructor like this:

```
int CompareInts(int n1, int n2)
{
    return n1 - n2;
}

wxSortedArrayInt sorted(CompareInts);

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

wxArrayOfMyClass another(CompareMyClassObjects);
```

## **WX\_DECLARE\_OBJARRAY**

**WX\_DECLARE\_OBJARRAY**(*T*, *name*)

**WX\_DECLARE\_EXPORTED\_OBJARRAY**(*T*, *name*)

**WX\_DECLARE\_USER\_EXPORTED\_OBJARRAY**(*T*, *name*)

This macro declares a new object array class named *name* and containing the elements of

type *T*. The second form is used when compiling wxWidgets as a DLL under Windows and array needs to be visible outside the DLL. The third is needed for exporting an array from a user DLL.

Example:

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

You must use `WX_DEFINE_OBJARRAY()` (p. 77) macro to define the array class - otherwise you would get link errors.

## **WX\_DEFINE\_OBJARRAY**

**WX\_DEFINE\_OBJARRAY**(*name*)

**WX\_DEFINE\_EXPORTED\_OBJARRAY**(*name*)

**WX\_DEFINE\_USER\_EXPORTED\_OBJARRAY**(*name*)

This macro defines the methods of the array class *name* not defined by the `WX_DECLARE_OBJARRAY()` (p. 76) macro. You must include the file `<wx/arrimpl.cpp>` before using this macro and you must have the full declaration of the class of array elements in scope! If you forget to do the first, the error will be caught by the compiler, but, unfortunately, many compilers will not give any warnings if you forget to do the second - but the objects of the class will not be copied correctly and their real destructor will not be called. The latter two forms are merely aliases of the first to satisfy some people's sense of symmetry when using the exported declarations.

Example of usage:

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

    ...

    virtual ~MyClass();
};

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

## **WX\_APPEND\_ARRAY**

**void WX\_APPEND\_ARRAY**(wxArray& *array*, wxArray& *other*)

This macro may be used to append all elements of the *other* array to the *array*. The two arrays must be of the same type.

**WX\_PREPEND\_ARRAY****void WX\_PREPEND\_ARRAY(wxArray& array, wxArray& other)**

This macro may be used to prepend all elements of the *other* array to the *array*. The two arrays must be of the same type.

**WX\_CLEAR\_ARRAY****void WX\_CLEAR\_ARRAY(wxArray& array)**

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

**Default constructors****wxArray()****wxObjArray()**

Default constructor initializes an empty array object.

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

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

**wxArray copy constructor and assignment operator****wxArray(const wxArray& array)****wxSortedArray(const wxSortedArray& array)****wxObjArray(const wxObjArray& array)****wxArray& operator=(const wxArray& array)****wxSortedArray& operator=(const wxSortedArray& array)****wxObjArray& operator=(const wxObjArray& array)**

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

**wxArray::~~wxArray****~wxArray()**

**~wxSortedArray()**

**~wxObjArray()**

The `wxObjArray` destructor deletes all the items owned by the array. This is not done by `wxArray` and `wxSortedArray` versions - you may use `WX_CLEAR_ARRAY` (p. 78) macro for this.

### **wxArray::Add**

**void Add(T item, size\_t copies = 1)**

**void Add(T \*item)**

**void Add(T &item, size\_t copies = 1)**

Appends the given number of *copies* of the *item* to the array consisting of the elements of type *T*.

The first version is used with `wxArray` and `wxSortedArray`. The second and the third are used with `wxObjArray`. There is an important difference between them: if you give a pointer to the array, it will take ownership of it, i.e. will delete it when the item is deleted from the array. If you give a reference to the array, however, the array will make a copy of the item and will not take ownership of the original item. Once again, it only makes sense for `wxObjArrays` because the other array types never take ownership of their elements. Also note that you cannot append more than one pointer as reusing it would lead to deleting it twice (or more) and hence to a crash.

You may also use `WX_APPEND_ARRAY` (p. 77) macro to append all elements of one array to another one but it is more efficient to use *copies* parameter and modify the elements in place later if you plan to append a lot of items.

### **wxArray::Alloc**

**void Alloc(size\_t count)**

Preallocates memory for a given number of array elements. It is worth calling when the number of items which are going to be added to the array is known in advance because it will save unneeded memory reallocation. If the array already has enough memory for the given number of items, nothing happens. In any case, the existing contents of the array is not modified.

### **wxArray::Clear**

**void Clear()**

This function does the same as `Empty()` (p. 80) and additionally frees the memory allocated to the array.

### **wxArray::Count**

**size\_t Count() const**

Same as *GetCount()* (p. 80). This function is deprecated - it exists only for compatibility.

**wxObjArray::Detach****T \* Detach(size\_t index)**

Removes the element from the array, but, unlike, *Remove()* (p. 81) doesn't delete it. The function returns the pointer to the removed element.

**wxArray::Empty****void Empty()**

Empties the array. For *wxObjArray* classes, this destroys all of the array elements. For *wxArray* and *wxSortedArray* this does nothing except marking the array of being empty - this function does not free the allocated memory, use *Clear()* (p. 79) for this.

**wxArray::GetCount****size\_t GetCount() const**

Return the number of items in the array.

**wxArray::Index****int Index(T& item, bool searchFromEnd = false) const****int Index(T& item) const**

The first version of the function is for *wxArray* and *wxObjArray*, the second is for *wxSortedArray* only.

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

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

**NB:** even for *wxObjArray* classes, the operator *==()* of the elements in the array is **not** used by this function. It searches exactly the given element in the array and so will only succeed if this element had been previously added to the array, but fail even if another, identical, element is in the array.

**wxArray::Insert****void Insert(T item, size\_t n, size\_t copies = 1)**



**void Insert(T \*item, size\_t n)**

**void Insert(T &item, size\_t n, size\_t copies = 1)**

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

Please see *Add()* (p. 79) for explanation of the differences between the overloaded versions of this function.

**wxArray::IsEmpty**

**bool IsEmpty() const**

Returns true if the array is empty, false otherwise.

**wxArray::Item**

**T& Item(size\_t index) const**

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

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

**wxArray::Last**

**T& Last() const**

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

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

**wxArray::Remove**

**Remove(T item)**

Removes an element from the array by value: the first item of the array equal to *item* is removed, an assert failure will result from an attempt to remove an item which doesn't exist in the array.

When an element is removed from *wxObjArray* it is deleted by the array - use *Detach()* (p. 80) if you don't want this to happen. On the other hand, when an object is removed from a *wxArray* nothing happens - you should delete it manually if required:

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

See also `WX_CLEAR_ARRAY` (p. 78) macro which deletes all elements of a `wxArray` (supposed to contain pointers).

### **wxArray::RemoveAt**

**RemoveAt**(`size_t index`, `size_t count = 1`)

Removes *count* elements starting at *index* from the array. When an element is removed from `wxObjArray` it is deleted by the array - use `Detach()` (p. 80) if you don't want this to happen. On the other hand, when an object is removed from a `wxArray` nothing happens - you should delete it manually if required:

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

See also `WX_CLEAR_ARRAY` (p. 78) macro which deletes all elements of a `wxArray` (supposed to contain pointers).

### **wxArray::SetCount**

**void SetCount**(`size_t count`, `T defval = T(0)`)

This function ensures that the number of array elements is at least *count*. If the array has already *count* or more items, nothing is done. Otherwise, `count - GetCount()` elements are added and initialized to the value *defval*.

#### **See also**

`GetCount` (p. 80)

### **wxArray::Shrink**

**void Shrink**()

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

### **wxArray::Sort**

**void Sort**(`CMPFUNC<T> compareFunction`)

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

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

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

Sorts the array using the specified compare function: this function should return a negative,

zero or positive value according to whether the first element passed to it is less than, equal to or greater than the second one.

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

## wxArrayString

`wxArrayString` is an efficient container for storing `wxString` (p. 1553) objects. It has the same features as all `wxArray` (p. 71) classes, i.e. it dynamically expands when new items are added to it (so it is as easy to use as a linked list), but the access time to the elements is constant, instead of being linear in number of elements as in the case of linked lists. It is also very size efficient and doesn't take more space than a C array `wxString[]` type (`wxArrayString` uses its knowledge of internals of `wxString` class to achieve this).

This class is used in the same way as other dynamic *arrays* (p. 71), except that no `WX_DEFINE_ARRAY` declaration is needed for it. When a string is added or inserted in the array, a copy of the string is created, so the original string may be safely deleted (e.g. if it was a `wxChar *` pointer the memory it was using can be freed immediately after this). In general, there is no need to worry about string memory deallocation when using this class - it will always free the memory it uses itself.

The references returned by *Item* (p. 86), *Last* (p. 87) or *operator[]* (p. 84) are not constant, so the array elements may be modified in place like this

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

There is also a variant of `wxArrayString` called `wxSortedArrayString` which has exactly the same methods as `wxArrayString`, but which always keeps the string in it in (alphabetical) order. `wxSortedArrayString` uses binary search in its *Index* (p. 86) function (instead of linear search for `wxArrayString::Index`) which makes it much more efficient if you add strings to the array rarely (because, of course, you have to pay for *Index()* efficiency by having *Add()* be slower) but search for them often. Several methods should not be used with sorted array (basically, all which break the order of items) which is mentioned in their description.

Final word: none of the methods of `wxArrayString` is virtual including its destructor, so this class should not be used as a base class.

### Derived from

Although this is not true strictly speaking, this class may be considered as a specialization of `wxArray` (p. 71) class for the `wxString` member data: it is not implemented like this, but it does have all of the `wxArray` functions.

### Include files

<wx/arrstr.h>

### See also

`wxArray` (p. 71), `wxString` (p. 1553), *wxString overview* (p. 2047)

**wxArrayString::wxArrayString****wxArrayString()**

Default constructor.

**wxArrayString(const wxArrayString& array)**

Copy constructor. Note that when an array is assigned to a sorted array, its contents is automatically sorted during construction.

**wxArrayString(size\_t sz, const wxChar\*\* arr)**

Constructor from a C string array. Pass a size *sz* and array *arr*.

**wxArrayString(size\_t sz, const wxString\* arr)**

Constructor from a wxString array. Pass a size *sz* and array *arr*.

**wxArrayString::~~wxArrayString****~wxArrayString()**

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

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

Assignment operator.

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

Compares 2 arrays respecting the case. Returns true only if the arrays have the same number of elements and the same strings in the same order.

**wxArrayString::operator!=****bool operator !=(const wxArrayString& array) const**

Compares 2 arrays respecting the case. Returns true if the arrays have different number of elements or if the elements don't match pairwise.

**wxArrayString::operator[]****wxString& operator[](size\_t nIndex)**

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

This is the operator version of *Item* (p. 86) method.

### **wxArrayString::Add**

**size\_t Add(const wxString& str, size\_t copies = 1)**

Appends the given number of *copies* of the new item *str* to the array and returns the index of the first new item in the array.

**Warning:** For sorted arrays, the index of the inserted item will not be, in general, equal to *GetCount()* (p. 86) - 1 because the item is inserted at the correct position to keep the array sorted and not appended.

See also: *Insert* (p. 86)

### **wxArrayString::Alloc**

**void Alloc(size\_t nCount)**

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

See also: *Dynamic array memory management* (p. 74)

### **wxArrayString::Clear**

**void Clear()**

Clears the array contents and frees memory.

See also: *Empty* (p. 85)

### **wxArrayString::Count**

**size\_t Count() const**

Returns the number of items in the array. This function is deprecated and is for backwards compatibility only, please use *GetCount* (p. 86) instead.

### **wxArrayString::Empty**

**void Empty()**

Empties the array: after a call to this function *GetCount* (p. 86) will return 0. However, this function does not free the memory used by the array and so should be used when the array is going to be reused for storing other strings. Otherwise, you should use *Clear* (p. 85) to empty the array and free memory.

**wxArrayString::GetCount****size\_t GetCount() const**

Returns the number of items in the array.

**wxArrayString::Index****int Index(const wxChar \* sz, bool bCase = true, bool bFromEnd = false)**

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

This function uses linear search for `wxArrayString` and binary search for `wxSortedArrayString`, but it ignores the *bCase* and *bFromEnd* parameters in the latter case.

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

**wxArrayString::Insert****void Insert(const wxString& str, size\_t nIndex, size\_t copies = 1)**

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

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

If *nIndex* is equal to *GetCount()* this function behaves as *Add* (p. 85).

**Warning:** this function should not be used with sorted arrays because it could break the order of items and, for example, subsequent calls to *Index()* (p. 86) would then not work!

**wxArrayString::IsEmpty****bool IsEmpty()**

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

**wxArrayString::Item****wxString& Item(size\_t nIndex) const**

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

See also *operator[]* (p. 84) for the operator version.

**wxArrayString::Last****wxString& Last()**

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

**wxArrayString::Remove****void Remove(const wxChar \* sz)**

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

See also: *Index* (p. 86)

**wxArrayString::RemoveAt****void RemoveAt(size\_t nIndex, size\_t count = 1)**

Removes *count* items starting at position *nIndex* from the array.

**wxArrayString::Shrink****void Shrink()**

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

See also: *Alloc* (p. 85), *Dynamic array memory management* (p. 74)

**wxArrayString::Sort****void Sort(bool reverseOrder = false)**

Sorts the array in alphabetical order or in reverse alphabetical order if *reverseOrder* is true. The sort is case-sensitive.

**Warning:** this function should not be used with sorted array because it could break the order of items and, for example, subsequent calls to *Index()* (p. 86) would then not work!

**void Sort(CompareFunction compareFunction)**

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

**Example**

The following example sorts strings by their length.

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

...

wxArrayString array;

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

array.Sort(CompareStringLen);
```

**Warning:** this function should not be used with sorted array because it could break the order of items and, for example, subsequent calls to *Index()* (p. 86) would then not work!

## wxArtProvider

wxArtProvider class is used to customize the look of wxWidgets application. When wxWidgets needs to display an icon or a bitmap (e.g. in the standard file dialog), it does not use a hard-coded resource but asks wxArtProvider for it instead. This way users can plug in their own wxArtProvider class and easily replace standard art with their own version. All that is needed is to derive a class from wxArtProvider, override its *CreateBitmap* (p. 91) method and register the provider with *wxArtProvider::Push* (p. 93):

```
class MyProvider : public wxArtProvider
{
protected:
    wxBitmap CreateBitmap(const wxArtID& id,
                        const wxArtClient& client,
                        const wxSize size)
    { ... }
};

...
wxArtProvider::Push(new MyProvider);
```

There's another way of taking advantage of this class: you can use it in your code and use platform native icons as provided by *wxArtProvider::GetBitmap* (p. 91) or *wxArtProvider::GetIcon* (p. 92) (NB: this is not yet really possible as of wxWidgets 2.3.3, the set of wxArtProvider bitmaps is too small).

## Identifying art resources

Every bitmap is known to wxArtProvider under an unique ID that is used by when requesting a resource from it. The ID is represented by wxArtID type and can have one of these predefined values (you can see bitmaps represented by these constants in the *artprov* (p. 2031) sample):

- wxART\_ADD\_BOOKMARK



- wxART\_DEL\_BOOKMARK
- wxART\_HELP\_SIDE\_PANEL
- wxART\_HELP\_SETTINGS
- wxART\_HELP\_BOOK
- wxART\_HELP\_FOLDER
- wxART\_HELP\_PAGE
- wxART\_GO\_BACK
- wxART\_GO\_FORWARD
- wxART\_GO\_UP
- wxART\_GO\_DOWN
- wxART\_GO\_TO\_PARENT
- wxART\_GO\_HOME
- wxART\_FILE\_OPEN
- wxART\_PRINT
- wxART\_HELP
- wxART\_TIP
- wxART\_REPORT\_VIEW
- wxART\_LIST\_VIEW
- wxART\_NEW\_DIR
- wxART\_FOLDER
- wxART\_GO\_DIR\_UP
- wxART\_EXECUTABLE\_FILE
- wxART\_NORMAL\_FILE
- wxART\_TICK\_MARK
- wxART\_CROSS\_MARK
- wxART\_ERROR
- wxART\_QUESTION
- wxART\_WARNING

- `wxART_INFORMATION`
- `wxART_MISSING_IMAGE`

Additionally, any string recognized by custom art providers registered using *Push* (p. 93) may be used.

### GTK+ Note

When running under GTK+ 2, GTK+ stock item IDs (e.g. `"gtk-cdrom"`) may be used as well. Additionally, if wxGTK was compiled against GTK+  $\geq 2.4$ , then it is also possible to load icons from current icon theme by specifying their name (without extension and directory components). Icon themes recognized by GTK+ follow thefreedesktop.org Icon Themes specification (<http://freedesktop.org/Standards/icon-theme-spec>). Note that themes are not guaranteed to contain all icons, so `wxArtProvider` may return `wxNullBitmap` or `wxNullIcon`. Default theme is typically installed in `/usr/share/icons/hicolor`.

### Clients

Client is the entity that calls `wxArtProvider`'s `GetBitmap` or `GetIcon` function. It is represented by `wxClietID` type and can have one of these values:

- `wxART_TOOLBAR`
- `wxART_MENU`
- `wxART_BUTTON`
- `wxART_FRAME_ICON`
- `wxART_CMN_DIALOG`
- `wxART_HELP_BROWSER`
- `wxART_MESSAGE_BOX`
- `wxART_OTHER` (used for all requests that don't fit into any of the categories above) Client ID servers as a hint to `wxArtProvider` that is supposed to help it to choose the best looking bitmap. For example it is often desirable to use slightly different icons in menus and toolbars even though they represent the same action (e.g. `wx_ART_FILE_OPEN`). Remember that this is really only a hint for `wxArtProvider` -- it is common that `wxArtProvider::GetBitmap` (p. 91) returns identical bitmap for different *client* values!

### See also

See the *artprov* (p. 2031) sample for an example of `wxArtProvider` usage.

### Derived from

*wxObject* (p. 1148)

**Include files**

<wx/artprov.h>

**wxArtProvider::~~wxArtProvider**

**~wxArtProvider()**

The destructor automatically removes the provider from the provider stack used by *GetBitmap* (p. 91).

**wxArtProvider::CreateBitmap**

**wxBitmap CreateBitmap(const wxArtID& *id*, const wxArtClient& *client*, const wxSize& *size*)**

Derived art provider classes must override this method to create requested art resource. Note that returned bitmaps are cached by wxArtProvider and it is therefore not necessary to optimize CreateBitmap for speed (e.g. you may create wxBitmap objects from XPMs here).

**Parameters**

*id*

wxArtID unique identifier of the bitmap.

*client*

wxArtClient identifier of the client (i.e. who is asking for the bitmap). This only serves as a hint.

*size*

Preferred size of the bitmap. The function may return a bitmap of different dimensions, it will be automatically rescaled to meet client's request.

**Note**

This is **not** part of wxArtProvider's public API, use *wxArtProvider::GetBitmap* (p. 91) or *wxArtProvider::GetIcon* (p. 92) to query wxArtProvider for a resource.

**wxArtProvider::Delete**

**static bool Delete(wxArtProvider\* *provider*)**

Delete the given *provider*.

**wxArtProvider::GetBitmap**

**static wxBitmap GetBitmap(const wxArtID& id, const wxArtClient& client = wxART\_OTHER, const wxSize& size = wxDefaultSize)**

Query registered providers for bitmap with given ID.

#### Parameters

*id*

wxArtID unique identifier of the bitmap.

*client*

wxArtClient identifier of the client (i.e. who is asking for the bitmap).

*size*

Size of the returned bitmap or wxDefaultSize if size doesn't matter.

#### Return value

The bitmap if one of registered providers recognizes the ID or wxNullBitmap otherwise.

#### wxArtProvider::GetIcon

**static wxIcon GetIcon(const wxArtID& id, const wxArtClient& client = wxART\_OTHER, const wxSize& size = wxDefaultSize)**

Same as *wxArtProvider::GetBitmap* (p. 91), but return a wxIcon object (or wxNullIcon on failure).

**static wxSize GetSizeHint(const wxArtClient& client, bool platform\_default = false)**

Returns a suitable size hint for the given *wxArtClient*. If *platform\_default* is `true`, return a size based on the current platform, otherwise return the size from the topmost *wxArtProvider*. *wxDefaultSize* may be returned if the client doesn't have a specified size, like *wxART\_OTHER* for example.

#### wxArtProvider::Insert

**static void Insert(wxArtProvider\* provider)**

Register new art provider and add it to the bottom of providers stack (i.e. it will be queried as the last one).

#### See also

*Push* (p. 93)

#### wxArtProvider::Pop

**static bool Pop()**

Remove latest added provider and delete it.

### **wxArtProvider::Push**

**static void Push**(wxArtProvider\* provider)

Register new art provider and add it to the top of providers stack (i.e. it will be queried as the first provider).

#### **See also**

*Insert* (p. 92)

### **wxArtProvider::Remove**

**static bool Remove**(wxArtProvider\* provider)

Remove a provider from the stack if it is on it. The provider is *not* deleted, unlike when using *Delete()* (p. 91).

## **wxAuiDockArt**

wxAuiDockArt is part of the wxAUI class framework. See also *wxAUI overview* (p. 2197).

Dock art provider code - a dock provider provides all drawing functionality to the wxAui dock manager. This allows the dock manager to have a plugable look-and-feel.

By default, a *wxAuiManager* (p. 99) uses an instance of this class called **wxAuiDefaultDockArt** which provides bitmap art and a colour scheme that is adapted to the major platforms' look. You can either derive from that class to alter its behaviour or write a completely new dock art class. Call *wxAuiManager::SetArtProvider* (p. 103) to make use this new dock art.

#### **Derived from**

No base class

#### **Include files**

<wx/au/dockart.h>

#### **See also**

*wxAuiManager* (p. 99), *wxAuiPanelInfo* (p. 110)

#### **Data structures**

```
enum wxAuiPaneDockArtSetting
{
    wxAUI_DOCKART_SASH_SIZE = 0,
    wxAUI_DOCKART_CAPTION_SIZE = 1,
    wxAUI_DOCKART_GRIPPER_SIZE = 2,
    wxAUI_DOCKART_PANE_BORDER_SIZE = 3,
```

```
    wxAUI_DOCKART_PANE_BUTTON_SIZE = 4,
    wxAUI_DOCKART_BACKGROUND_COLOUR = 5,
    wxAUI_DOCKART_SASH_COLOUR = 6,
    wxAUI_DOCKART_ACTIVE_CAPTION_COLOUR = 7,
    wxAUI_DOCKART_ACTIVE_CAPTION_GRADIENT_COLOUR = 8,
    wxAUI_DOCKART_INACTIVE_CAPTION_COLOUR = 9,
    wxAUI_DOCKART_INACTIVE_CAPTION_GRADIENT_COLOUR = 10,
    wxAUI_DOCKART_ACTIVE_CAPTION_TEXT_COLOUR = 11,
    wxAUI_DOCKART_INACTIVE_CAPTION_TEXT_COLOUR = 12,
    wxAUI_DOCKART_BORDER_COLOUR = 13,
    wxAUI_DOCKART_GRIPPER_COLOUR = 14,
    wxAUI_DOCKART_CAPTION_FONT = 15,
    wxAUI_DOCKART_GRADIENT_TYPE = 16
}

enum wxAuiPaneDockArtGradients

    wxAUI_GRADIENT_NONE = 0,
    wxAUI_GRADIENT_VERTICAL = 1,
    wxAUI_GRADIENT_HORIZONTAL = 2

enum wxAuiPaneButtonState
{
    wxAUI_BUTTON_STATE_NORMAL = 0,
    wxAUI_BUTTON_STATE_HOVER = 1,
    wxAUI_BUTTON_STATE_PRESSED = 2
}

enum wxAuiButtonId

    wxAUI_BUTTON_CLOSE = 101,
    wxAUI_BUTTON_MAXIMIZE_RESTORE = 102,
    wxAUI_BUTTON_MINIMIZE = 103,
    wxAUI_BUTTON_PIN = 104,
    wxAUI_BUTTON_OPTIONS = 105,
    wxAUI_BUTTON_WINDOWLIST = 106,
    wxAUI_BUTTON_LEFT = 107,
    wxAUI_BUTTON_RIGHT = 108,
    wxAUI_BUTTON_UP = 109,
    wxAUI_BUTTON_DOWN = 110,
    wxAUI_BUTTON_CUSTOM1 = 201,
    wxAUI_BUTTON_CUSTOM2 = 202,
    wxAUI_BUTTON_CUSTOM3 = 203
;
```

## **wxAuiDockArt::wxAuiDockArt**

### **wxAuiDockArt()**

Constructor.

**wxAuiDockArt::~~wxAuiDockArt****~wxAuiDockArt()**

Destructor.

**wxAuiDockArt::DrawBackground****virtual void DrawBackground(wxDC& dc, wxWindow\* window, int orientation, const wxRect& rect)**

Draws a background.

**wxAuiDockArt::DrawBorder****virtual void DrawBorder(wxDC& dc, wxWindow\* window, const wxRect& rect, wxAuiPanelInfo& pane)**

Draws a border.

**wxAuiDockArt::DrawCaption****virtual void DrawCaption(wxDC& dc, wxWindow\* window, const wxString& text, const wxRect& rect, wxAuiPanelInfo& pane)**

Draws a caption.

**wxAuiDockArt::DrawGripper****virtual void DrawGripper(wxDC& dc, wxWindow\* window, const wxRect& rect, wxAuiPanelInfo& pane)**

Draws a gripper.

**wxAuiDockArt::DrawPaneButton****virtual void DrawPaneButton(wxDC& dc, wxWindow\* window, int button, int button\_state, const wxRect& rect, wxAuiPanelInfo& pane)**

Draws a button in the pane's title bar.

*button* can be one of the values of **wxAuiButtonId**.*button\_state* can be one of the values of **wxAuiPaneButtonState**.**wxAuiDockArt::DrawSash****virtual void DrawSash(wxDC& dc, wxWindow\* window, int orientation, const wxRect& rect)**

Draws a sash between two windows.

**wxAuiDockArt::GetColor****virtual wxColour GetColor(int id)**

The same as *GetColour* (p. 96).

**wxAuiDockArt::GetColour****virtual wxColour GetColour(int id)**

Get the colour of a certain setting.

*id* can be one of the colour values of **wxAuiPaneDockArtSetting**.

**wxAuiDockArt::GetFont****virtual wxFont GetFont(int id)**

Get a font setting.

**wxAuiDockArt::GetMetric****virtual int GetMetric(int id)**

Get the value of a certain setting.

*id* can be one of the size values of **wxAuiPaneDockArtSetting**.

**wxAuiDockArt::SetColor****virtual void SetColor(int id, const wxColour& color)**

The same as *SetColour* (p. 96).

**wxAuiDockArt::SetColour****virtual void SetColour(int id, const wxColor& colour)**

Set a certain setting with the value *colour*.

*id* can be one of the colour values of **wxAuiPaneDockArtSetting**.

**wxAuiDockArt::SetFont****virtual void SetFont(int id, const wxFont& font)**

Set a font setting.

**wxAuiDockArt::SetMetric****virtual void SetMetric(int id, int new\_val)**



Set a certain setting with the value *new\_val*.

*id* can be one of the size values of **wxAuiPaneDockArtSetting**.

## **wxAuiTabArt**

Tab art class.

### **Derived from**

No base class

### **Include files**

<wx/auibook.h>

### **Data structures**

### **wxAuiTabArt::wxAuiTabArt**

**wxAuiTabArt()**

Constructor.

### **wxAuiTabArt::Clone**

**wxAuiTabArt\* Clone()**

Clones the art object.

### **wxAuiTabArt::DrawBackground**

**void DrawBackground(wxDC& dc, wxWindow\* wnd, const wxRect& rect)**

Draws a background on the given area.

### **wxAuiTabArt::DrawButton**

**void DrawButton(wxDC& dc, wxWindow\* wnd, const wxRect& in\_rect, int bitmap\_id, int button\_state, int orientation, const wxBitmap& bitmap\_override, wxRect\* out\_rect)**

Draws a button.

### **wxAuiTabArt::DrawTab**

**void DrawTab(wxDC& dc, wxWindow\* wnd, const wxRect& in\_rect, const wxString& caption, const wxBitmap& bitmap, bool active, int close\_button\_state, wxRect\* out\_tab\_rect, wxRect\* out\_button\_rect, int\* x\_extent)**

Draws a tab.

### **wxAuiTabArt::GetBestTabCtrlSize**

**int GetBestTabCtrlSize(wxWindow\* *wnd*, wxAuiNotebookPageArray& *pages*)**

Returns the tab control size.

### **wxAuiTabArt::GetIndentSize**

**int GetIndentSize()**

Returns the indent size.

### **wxAuiTabArt::GetTabSize**

**wxSize GetTabSize(wxDC& *dc*, wxWindow\* *wnd*, const wxString& *caption*, const wxBitmap& *bitmap*, bool *active*, int *close\_button\_state*, int\* *x\_extent*)**

Returns the tab size for the given caption, bitmap and state.

### **wxAuiTabArt::SetFlags**

**void SetFlags(unsigned int *flags*)**

Sets flags.

### **wxAuiTabArt::SetMeasuringFont**

**void SetMeasuringFont(const wxFont& *font*)**

Sets the font used for calculating measurements.

### **wxAuiTabArt::SetNormalFont**

**void SetNormalFont(const wxFont& *font*)**

Sets the normal font for drawing labels.

### **wxAuiTabArt::SetSelectedFont**

**void SetSelectedFont(const wxFont& *font*)**

Sets the font for drawing text for selected UI elements.

### **wxAuiTabArt::SetSizingInfo**

**void SetSizingInfo(const wxSize& *tab\_ctrl\_size*, size\_t *tab\_count*)**

Sets sizing information.

## **wxAuiTabArt::ShowWindowList**

**int ShowWindowList**(wxWindow\* *wnd*, const wxArrayString& *items*, int *active\_idx*)

Pops up a menu to show the list of windows managed by wxAui.

## **wxAuiManager**

wxAuiManager is the central class of the wxAUI class framework.

See also *wxAUI overview* (p. 2197).

wxAuiManager manages the panes associated with it for a particular wxFrame, using a pane's wxAuiPaneInfo information to determine each pane's docking and floating behavior. wxAuiManager uses wxWidgets' sizer mechanism to plan the layout of each frame. It uses a replaceable dock art class to do all drawing, so all drawing is localized in one area, and may be customized depending on an application's specific needs.

wxAuiManager works as follows: the programmer adds panes to the class, or makes changes to existing pane properties (dock position, floating state, show state, etc.). To apply these changes, wxAuiManager's Update() function is called. This batch processing can be used to avoid flicker, by modifying more than one pane at a time, and then "committing" all of the changes at once by calling Update().

Panes can be added quite easily:

```
wxTextCtrl* text1 = new wxTextCtrl(this, -1);
wxTextCtrl* text2 = new wxTextCtrl(this, -1);
m_mgr.AddPane(text1, wxLEFT, wxT("Pane Caption"));
m_mgr.AddPane(text2, wxBOTTOM, wxT("Pane Caption"));
m_mgr.Update();
```

Later on, the positions can be modified easily. The following will float an existing pane in a tool window:

```
m_mgr.GetPane(text1).Float();
```

## **Layers, Rows and Directions, Positions**

Inside wxAUI, the docking layout is figured out by checking several pane parameters. Four of these are important for determining where a pane will end up:

**Direction:**Each docked pane has a direction, Top, Bottom, Left, Right, or Center. This is fairly self-explanatory. The pane will be placed in the location specified by this variable.

**Position:**More than one pane can be placed inside of a dock. Imagine two panes being docked on the left side of a window. One pane can be placed over another. In proportionally managed docks, the pane position indicates its sequential position, starting with zero. So, in our scenario with two panes docked on the left side, the top pane in the dock would have position 0, and the second one would occupy position 1.

**Row:**A row can allow for two docks to be placed next to each other. One of the most common places for this to happen is in the toolbar. Multiple toolbar rows are allowed, the

first row being row 0, and the second row 1. Rows can also be used on vertically docked panes.

**Layer:** A layer is akin to an onion. Layer 0 is the very center of the managed pane. Thus, if a pane is in layer 0, it will be closest to the center window (also sometimes known as the "content window"). Increasing layers "swallow up" all layers of a lower value. This can look very similar to multiple rows, but is different because all panes in a lower level yield to panes in higher levels. The best way to understand layers is by running the wxAUI sample.

### Derived from

*wxEvtHandler* (p. 576)

### Include files

<wx/au/au.h>

### See also

*wxAuiPaneInfo* (p. 110), *wxAuiDockArt* (p. 93)

### Data structures

```
enum wxAuiManagerDock
{
    wxAUI_DOCK_NONE = 0,
    wxAUI_DOCK_TOP = 1,
    wxAUI_DOCK_RIGHT = 2,
    wxAUI_DOCK_BOTTOM = 3,
    wxAUI_DOCK_LEFT = 4,
    wxAUI_DOCK_CENTER = 5,
    wxAUI_DOCK_CENTRE = wxAUI_DOCK_CENTER
}
```

```
enum wxAuiManagerOption

    wxAUI_MGR_ALLOW_FLOATING           = 1 << 0,
    wxAUI_MGR_ALLOW_ACTIVE_PANE        = 1 << 1,
    wxAUI_MGR_TRANSPARENT_DRAG         = 1 << 2,
    wxAUI_MGR_TRANSPARENT_HINT         = 1 << 3,
    wxAUI_MGR_VENETIAN_BLINDS_HINT     = 1 << 4,
    wxAUI_MGR_RECTANGLE_HINT           = 1 << 5,
    wxAUI_MGR_HINT_FADE                 = 1 << 6,
    wxAUI_MGR_NO_VENETIAN_BLINDS_FADE  = 1 << 7,

    wxAUI_MGR_DEFAULT = wxAUI_MGR_ALLOW_FLOATING |
                        wxAUI_MGR_TRANSPARENT_HINT |
                        wxAUI_MGR_HINT_FADE |
                        wxAUI_MGR_NO_VENETIAN_BLINDS_FADE
```

**wxAuiManager::wxAuiManager**

**wxAuiManager**(wxWindow\* *managed\_wnd* = NULL, unsigned int *flags* = wxAUI\_MGR\_DEFAULT)

Constructor. *managed\_wnd* specifies the wxFrame which should be managed. *flags* specifies options which allow the frame management behavior to be modified.

**wxAuiManager::~~wxAuiManager**

**~wxAuiManager**()

**wxAuiManager::AddPane**

**bool** AddPane(wxWindow\* *window*, const wxAuiPanelInfo& *pane\_info*)

**bool** AddPane(wxWindow\* *window*, int *direction* = wxLEFT, const wxString& *caption* = wxEmptyString)

**bool** AddPane(wxWindow\* *window*, const wxAuiPanelInfo& *pane\_info*, const wxPoint& *drop\_pos*)

AddPane() tells the frame manager to start managing a child window. There are several versions of this function. The first version allows the full spectrum of pane parameter possibilities. The second version is used for simpler user interfaces which do not require as much configuration. The last version allows a drop position to be specified, which will determine where the pane will be added.

**wxAuiManager::DetachPane**

**bool** DetachPane(wxWindow\* *window*)

Tells the wxAuiManager to stop managing the pane specified by *window*. The window, if in a floated frame, is reparented to the frame managed by wxAuiManager.

**wxAuiManager::GetAllPanels**

**wxAuiPanelInfoArray&** GetAllPanels()

Returns an array of all panes managed by the frame manager.

**wxAuiManager::GetArtProvider**

**wxAuiDockArt\*** GetArtProvider() const

Returns the current art provider being used.

See also: *wxAuiDockArt* (p. 93).

**wxAuiManager::GetDockSizeConstraint**

**void GetDockSizeConstraint(double\* widthpct, double\* heightpct)**

Returns the current dock constraint values. See *SetDockSizeConstraint()* (p. 104) for more information.

**wxAuiManager::GetFlags**

**unsigned int GetFlags() const**

Returns the current manager's flags.

**wxAuiManager::GetManagedWindow**

**wxWindow\* GetManagedWindow() const**

Returns the frame currently being managed by wxAuiManager.

**wxAuiManager::GetManager**

**static wxAuiManager\* GetManager(wxWindow\* window)**

Calling this method will return the wxAuiManager for a given window. The *window* parameter should specify any child window or sub-child window of the frame or window managed by wxAuiManager. The *window* parameter need not be managed by the manager itself, nor does it even need to be a child or sub-child of a managed window. It must however be inside the window hierarchy underneath the managed window.

**wxAuiManager::GetPane**

**wxAuiPanelInfo& GetPane(wxWindow\* window)**

**wxAuiPanelInfo& GetPane(const wxString& name)**

*GetPane* is used to lookup a wxAuiPanelInfo object either by window pointer or by pane name, which acts as a unique id for a window pane. The returned wxAuiPanelInfo object may then be modified to change a pane's look, state or position. After one or more modifications to wxAuiPanelInfo, wxAuiManager::Update() should be called to commit the changes to the user interface. If the lookup failed (meaning the pane could not be found in the manager), a call to the returned wxAuiPanelInfo's *IsOk()* method will return false.

**wxAuiManager::HideHint**

**void HideHint()**

HideHint() hides any docking hint that may be visible.

**wxAuiManager::InsertPane**

**bool InsertPane(wxWindow\* window, const wxAuiPanelInfo& insert\_location, int insert\_level = wxAUI\_INSERT\_PANE)**

This method is used to insert either a previously unmanaged pane window into the frame manager, or to insert a currently managed pane somewhere else. *InsertPane* will push all panes, rows, or docks aside and insert the window into the position specified by *insert\_location*. Because *insert\_location* can specify either a pane, dock row, or dock layer, the *insert\_level* parameter is used to disambiguate this. The parameter *insert\_level* can take a value of `wxAUI_INSERT_PANE`, `wxAUI_INSERT_ROW` or `wxAUI_INSERT_DOCK`.

### **wxAuiManager::LoadPanelInfo**

**void LoadPanelInfo(wxString pane\_part, wxAuiPanelInfo& pane)**

LoadPanelInfo() is similar to LoadPerspective, with the exception that it only loads information about a single pane. It is used in combination with SavePanelInfo().

### **wxAuiManager::LoadPerspective**

**bool LoadPerspective(const wxString& perspective, bool update = true)**

Loads a saved perspective. If update is true, wxAuiManager::Update() is automatically invoked, thus realizing the saved perspective on screen.

### **wxAuiManager::ProcessDockResult**

**bool ProcessDockResult(wxAuiPanelInfo& target, const wxAuiPanelInfo& new\_pos)**

ProcessDockResult() is a protected member of the wxAUI layout manager. It can be overridden by derived classes to provide custom docking calculations.

### **wxAuiManager::SavePanelInfo**

**wxString SavePanelInfo(wxAuiPanelInfo& pane)**

SavePanelInfo() is similar to SavePerspective, with the exception that it only saves information about a single pane. It is used in combination with LoadPanelInfo().

### **wxAuiManager::SavePerspective**

**wxString SavePerspective()**

Saves the entire user interface layout into an encoded wxString, which can then be stored by the application (probably using wxConfig). When a perspective is restored using LoadPerspective(), the entire user interface will return to the state it was when the perspective was saved.

### **wxAuiManager::SetArtProvider**

**void SetArtProvider(wxAuiDockArt\* art\_provider)**

Instructs wxAuiManager to use art provider specified by parameter *art\_provider* for all

drawing calls. This allows plugable look-and-feel features. The previous art provider object, if any, will be deleted by `wxAuiManager`.

See also: *wxAuiDockArt* (p. 93).

### **wxAuiManager::SetDockSizeConstraint**

**void SetDockSizeConstraint(double widthpct, double heightpct)**

When a user creates a new dock by dragging a window into a docked position, often times the large size of the window will create a dock that is unwieldly large. `wxAuiManager` by default limits the size of any new dock to 1/3 of the window size. For horizontal docks, this would be 1/3 of the window height. For vertical docks, 1/3 of the width. Calling this function will adjust this constraint value. The numbers must be between 0.0 and 1.0. For instance, calling `SetDockSizeConstraint` with 0.5, 0.5 will cause new docks to be limited to half of the size of the entire managed window.

### **wxAuiManager::SetFlags**

**void SetFlags(unsigned int flags)**

This method is used to specify `wxAuiManager`'s settings flags. *flags* specifies options which allow the frame management behavior to be modified.

### **wxAuiManager::SetManagedWindow**

**void SetManagedWindow(wxWindow\* managed\_wnd)**

Called to specify the frame or window which is to be managed by `wxAuiManager`. Frame management is not restricted to just frames. Child windows or custom controls are also allowed.

### **wxAuiManager::ShowHint**

**void ShowHint(const wxRect& rect)**

This function is used by controls to explicitly show a hint window at the specified rectangle. It is rarely called, and is mostly used by controls implementing custom pane drag/drop behaviour. The specified rectangle should be in screen coordinates.

### **wxAuiManager::UnInit**

**void UnInit()**

Uninitializes the framework and should be called before a managed frame or window is destroyed. `UnInit()` is usually called in the managed `wxFrame`'s destructor. It is necessary to call this function before the managed frame or window is destroyed, otherwise the manager cannot remove its custom event handlers from a window.

### **wxAuiManager::Update**



**void Update()**

This method is called after any number of changes are made to any of the managed panes. Update() must be invoked after AddPane() or InsertPane() are called in order to "realize" or "commit" the changes. In addition, any number of changes may be made to wxAuiPaneInfo structures (retrieved with wxAuiManager::GetPane), but to realize the changes, Update() must be called. This construction allows pane flicker to be avoided by updating the whole layout at one time.

**wxAuiNotebook**

wxAuiNotebook is part of the wxAUI class framework. See also *wxAUI overview* (p. 2197).

wxAuiNotebook is a notebook control which implements many features common in applications with dockable panes. Specifically, wxAuiNotebook implements functionality which allows the user to rearrange tab order via drag-and-drop, split the tab window into many different splitter configurations, and toggle through different themes to customize the control's look and feel.

An effort has been made to try to maintain an API as similar to that of wxNotebook.

The default theme that is used is wxAuiDefaultTabArt, which provides a modern, glossy look and feel. The theme can be changed by calling *wxAuiNotebook::SetArtProvider* (p. 108).

**Derived from**

*wxControl* (p. 285)

**Include files**

<wx/auui/auibook.h>

**Window styles**

**wxAUI\_NB\_DEFAULT\_STYLE** Defined as wxAUI\_NB\_TOP |  
wxAUI\_NB\_TAB\_SPLIT | wxAUI\_NB\_TAB\_MOVE |  
wxAUI\_NB\_SCROLL\_BUTTONS |  
wxAUI\_NB\_CLOSE\_ON\_ACTIVE\_TAB .

**wxAUI\_NB\_TAB\_SPLIT** Allows the tab control to be split by dragging a tab.

**wxAUI\_NB\_TAB\_MOVE** Allows a tab to be moved horizontally by dragging.

**wxAUI\_NB\_TAB\_EXTERNAL\_MOVE** Allows a tab to be moved to another tab  
control.

**wxAUI\_NB\_TAB\_FIXED\_WIDTH** With this style, all tabs have the same width.

**wxAUI\_NB\_SCROLL\_BUTTONS** With this style, left and right scroll buttons are  
displayed.

**wxAUI\_NB\_WINDOWLIST\_BUTTON** With this style, a drop-down list of windows is

available.

**wxAUI\_NB\_CLOSE\_BUTTON** With this style, a close button is available on the tab bar.

**wxAUI\_NB\_CLOSE\_ON\_ACTIVE\_TAB** With this style, the close button is visible on the active tab.

**wxAUI\_NB\_CLOSE\_ON\_ALL\_TABS** With this style, the close button is visible on all tabs.

**wxAUI\_NB\_TOP** With this style, tabs are drawn along the top of the notebook.

**wxAUI\_NB\_BOTTOM** With this style, tabs are drawn along the bottom of the notebook.

## Data structures

### **wxAuiNotebook::wxAuiNotebook**

**wxAuiNotebook()**

**wxAuiNotebook**(wxWindow\* *parent*, wxWindowID *id* = wxID\_ANY, const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size* = wxDefaultSize, long *style* = wxAUI\_NB\_DEFAULT\_STYLE)

Constructor. Creates a wxAuiNotebok control.

### **wxAuiNotebook::AddPage**

**bool AddPage**(wxWindow\* *page*, const wxString& *caption*, bool *select* = false, const wxBitmap& *bitmap* = wxNullBitmap)

Adds a page. If the *select* parameter is true, calling this will generate a page change event.

### **wxAuiNotebook::AdvanceSelection**

**void AdvanceSelection**(bool *forward* = true)

Sets the selection to the next or previous page.

### **wxAuiNotebook::Create**

**bool Create**(wxWindow\* *parent*, wxWindowID *id* = wxID\_ANY, const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size* = wxDefaultSize, long *style* = 0)

Creates the notebook window.

### **wxAuiNotebook::DeletePage**

**bool DeletePage(size\_t page)**

Deletes a page at the given index. Calling this method will generate a page change event.

**wxAuiNotebook::GetArtProvider**

**wxAuiTabArt\* GetArtProvider() const**

Returns the associated art provider.

**wxAuiNotebook::GetHeightForPageHeight**

**int GetHeightForPageHeight(int pageHeight)**

Returns the desired height of the notebook for the given page height. Use this to fit the notebook to a given page size.

**wxAuiNotebook::GetPage**

**wxWindow\* GetPage(size\_t page\_idx) const**

Returns the page specified by the given index.

**wxAuiNotebook::GetPageBitmap**

**wxBitmap GetPageBitmap(size\_t page) const**

Returns the tab bitmap for the page.

**wxAuiNotebook::GetPageCount**

**size\_t GetPageCount() const**

Returns the number of pages in the notebook.

**wxAuiNotebook::GetPageIndex**

**int GetPageIndex(wxWindow\* page\_wnd) const**

Returns the page index for the specified window. If the window is not found in the notebook, wxNOT\_FOUND is returned.

**wxAuiNotebook::GetPageText**

**wxString GetPageText(size\_t page) const**

Returns the tab label for the page.

**wxAuiNotebook::GetSelection**

**int GetSelection() const**

Returns the currently selected page.

**wxAuiNotebook::GetTabCtrlHeight****int GetTabCtrlHeight() const**

Returns the height of the tab control.

**wxAuiNotebook::InsertPage**

**bool InsertPage(size\_t page\_idx, wxWindow\* page, const wxString& caption, bool select = false, const wxBitmap& bitmap = wxNullBitmap)**

InsertPage() is similar to AddPage, but allows the ability to specify the insert location. If the *select* parameter is true, calling this will generate a page change event.

**wxAuiNotebook::RemovePage**

**bool RemovePage(size\_t page)**

Removes a page, without deleting the window pointer.

**wxAuiNotebook::SetArtProvider**

**void SetArtProvider(wxAuiTabArt\* art)**

Sets the art provider to be used by the notebook.

**wxAuiNotebook::SetFont**

**bool SetFont(const wxFont& font)**

Sets the font for drawing the tab labels, using a bold version of the font for selected tab labels.

**wxAuiNotebook::SetNormalFont**

**void SetNormalFont(const wxFont& font)**

Sets the font for drawing unselected tab labels.

**wxAuiNotebook::SetSelectedFont**

**void SetSelectedFont(const wxFont& font)**

Sets the font for drawing selected tab labels.

**wxAuiNotebook::SetMeasuringFont**

**void SetMeasuringFont(const wxFont& font)**

Sets the font for measuring tab labels.

**wxAuiNotebook::SetPageBitmap**

**bool SetPageBitmap(size\_t page, const wxBitmap& bitmap)**

Sets the bitmap for the page. To remove a bitmap from the tab caption, pass wxNullBitmap.

**wxAuiNotebook::SetPageText**

**bool SetPageText(size\_t page, const wxString& text)**

Sets the tab label for the page.

**wxAuiNotebook::SetSelection**

**size\_t SetSelection(size\_t new\_page)**

Sets the page selection. Calling this method will generate a page change event.

**wxAuiNotebook::SetTabCtrlHeight**

**void SetTabCtrlHeight(int height)**

Sets the tab height. By default, the tab control height is calculated by measuring the text height and bitmap sizes on the tab captions. Calling this method will override that calculation and set the tab control to the specified height parameter. A call to this method will override any call to SetUniformBitmapSize(). Specifying -1 as the height will return the control to its default auto-sizing behaviour.

**wxAuiNotebook::SetUniformBitmapSize**

**void SetUniformBitmapSize(const wxSize& size)**

SetUniformBitmapSize() ensures that all tabs will have the same height, even if some tabs don't have bitmaps. Passing wxDefaultSize to this function will instruct the control to use dynamic tab height, which is the default behaviour. Under the default behaviour, when a tab with a large bitmap is added, the tab control's height will automatically increase to accommodate the larger bitmap.

**void Split(size\_t page, int direction)**

Split performs a split operation programmatically. The argument *page* indicates the page that will be split off. This page will also become the active page after the split. The *direction* argument specifies where the pane should go, it should be one of the following: wxTOP, wxBOTTOM, wxLEFT, or wxRIGHT.

**wxAuiNotebook::ShowWindowMenu**

**bool ShowWindowMenu()**

Shows the window menu for the active tab control associated with this notebook, and returns `true` if a selection was made.

**wxAuiPanelInfo**

`wxAuiPanelInfo` is part of the `wxAUI` class framework. See also *wxAUI overview* (p. 2197).

`wxAuiPanelInfo` specifies all the parameters for a pane. These parameters specify where the pane is on the screen, whether it is docked or floating, or hidden. In addition, these parameters specify the pane's docked position, floating position, preferred size, minimum size, caption text among many other parameters.

**Derived from**

No base class

**Include files**

<wx/au/au.h>

**See also**

*wxAuiManager* (p. 99), *wxAuiDockArt* (p. 93)

**Data structures**

```
enum wxAuiPaneInsertLevel
{
    wxAUI_INSERT_PANE = 0,
    wxAUI_INSERT_ROW = 1,
    wxAUI_INSERT_DOCK = 2
}
```

**wxAuiPanelInfo::wxAuiPanelInfo****wxAuiPanelInfo()**

Default constructor.

**wxAuiPanelInfo(const wxAuiPanelInfo& c)**

Copy constructor.

**wxAuiPanelInfo::~wxAuiPanelInfo****~wxAuiPanelInfo()****wxAuiPanelInfo::BestSize**

**wxAuiPanelInfo& BestSize(const wxSize& size)**

**wxAuiPanelInfo& BestSize(int x, int y)**

BestSize() sets the ideal size for the pane. The docking manager will attempt to use this size as much as possible when docking or floating the pane.

**wxAuiPanelInfo::Bottom**

**wxAuiPanelInfo& Bottom()**

Bottom() sets the pane dock position to the bottom side of the frame. This is the same thing as calling Direction(wxAUI\_DOCK\_BOTTOM).

**wxAuiPanelInfo::BottomDockable**

**wxAuiPanelInfo& BottomDockable(bool b = true)**

BottomDockable() indicates whether a pane can be docked at the bottom of the frame.

**wxAuiPanelInfo::Caption**

**wxAuiPanelInfo& Caption(const wxString& c)**

Caption() sets the caption of the pane.

**wxAuiPanelInfo::CaptionVisible**

**wxAuiPanelInfo& CaptionVisible(bool visible = true)**

CaptionVisible indicates that a pane caption should be visible. If false, no pane caption is drawn.

**wxAuiPanelInfo::Centre**

**wxAuiPanelInfo& Centre()**

**wxAuiPanelInfo& Center()**

Center() sets the pane dock position to the left side of the frame. The centre pane is the space in the middle after all border panes (left, top, right, bottom) are subtracted from the layout.

This is the same thing as calling Direction(wxAUI\_DOCK\_CENTRE).

**wxAuiPanelInfo::CentrePane**

**wxAuiPanelInfo& CentrePane()**

**wxAuiPanelInfo& CenterPane()**

`CentrePane()` specifies that the pane should adopt the default center pane settings. Centre panes usually do not have caption bars. This function provides an easy way of preparing a pane to be displayed in the center dock position.

### **`wxAuiPanelInfo::CloseButton`**

**`wxAuiPanelInfo& CloseButton(bool visible = true)`**

`CloseButton()` indicates that a close button should be drawn for the pane.

### **`wxAuiPanelInfo::DefaultPane`**

**`wxAuiPanelInfo& DefaultPane()`**

`DefaultPane()` specifies that the pane should adopt the default pane settings.

### **`wxAuiPanelInfo::DestroyOnClose`**

**`wxAuiPanelInfo& DestroyOnClose(bool b = true)`**

`DestroyOnClose()` indicates whether a pane should be destroyed when it is closed. Normally a pane is simply hidden when the close button is clicked. Setting `DestroyOnClose` to true will cause the window to be destroyed when the user clicks the pane's close button.

### **`wxAuiPanelInfo::Direction`**

**`wxAuiPanelInfo& Direction(int direction)`**

`Direction()` determines the direction of the docked pane. It is functionally the same as calling `Left()`, `Right()`, `Top()` or `Bottom()`, except that docking direction may be specified programmatically via the parameter.

### **`wxAuiPanelInfo::Dock`**

**`wxAuiPanelInfo& Dock()`**

`Dock()` indicates that a pane should be docked. It is the opposite of `Float()`.

### **`wxAuiPanelInfo::DockFixed`**

**`wxAuiPanelInfo& DockFixed(bool b = true)`**

`DockFixed()` causes the containing dock to have no resize sash. This is useful for creating panes that span the entire width or height of a dock, but should not be resizable in the other direction.

### **`wxAuiPanelInfo::Dockable`**

**`wxAuiPanelInfo& Dockable(bool b = true)`**



`Dockable()` specifies whether a frame can be docked or not. It is the same as specifying `TopDockable(b).BottomDockable(b).LeftDockable(b).RightDockable(b)`.

### **`wxAuiPanelInfo::Fixed`**

#### **`wxAuiPanelInfo& Fixed()`**

`Fixed()` forces a pane to be fixed size so that it cannot be resized. After calling `Fixed()`, `IsFixed()` will return true.

### **`wxAuiPanelInfo::Float`**

#### **`wxAuiPanelInfo& Float()`**

`Float()` indicates that a pane should be floated. It is the opposite of `Dock()`.

### **`wxAuiPanelInfo::Floatable`**

#### **`wxAuiPanelInfo& Floatable(bool b = true)`**

`Floatable()` sets whether the user will be able to undock a pane and turn it into a floating window.

### **`wxAuiPanelInfo::FloatingPosition`**

#### **`wxAuiPanelInfo& FloatingPosition(const wxPoint& pos)`**

#### **`wxAuiPanelInfo& FloatingPosition(int x, int y)`**

`FloatingPosition()` sets the position of the floating pane.

### **`wxAuiPanelInfo::FloatingSize`**

#### **`wxAuiPanelInfo& FloatingSize(const wxSize& size)`**

#### **`wxAuiPanelInfo& FloatingSize(int x, int y)`**

`FloatingSize()` sets the size of the floating pane.

### **`wxAuiPanelInfo::Gripper`**

#### **`wxAuiPanelInfo& Gripper(bool visible = true)`**

`Gripper()` indicates that a gripper should be drawn for the pane.

### **`wxAuiPanelInfo::GripperTop`**

#### **`wxAuiPanelInfo& GripperTop(bool attop = true)`**

`GripperTop()` indicates that a gripper should be drawn at the top of the pane.

**wxAuiPanelInfo::HasBorder****bool HasBorder() const**

HasBorder() returns true if the pane displays a border.

**wxAuiPanelInfo::HasCaption****bool HasCaption() const**

HasCaption() returns true if the pane displays a caption.

**wxAuiPanelInfo::HasCloseButton****bool HasCloseButton() const**

HasCloseButton() returns true if the pane displays a button to close the pane.

**wxAuiPanelInfo::HasFlag****bool HasFlag(unsigned int *flag*) const**

HasFlag() returns true if the the property specified by flag is active for the pane.

**wxAuiPanelInfo::HasGripper****bool HasGripper() const**

HasGripper() returns true if the pane displays a gripper.

**wxAuiPanelInfo::HasGripperTop****bool HasGripperTop() const**

HasGripper() returns true if the pane displays a gripper at the top.

**wxAuiPanelInfo::HasMaximizeButton****bool HasMaximizeButton() const**

HasMaximizeButton() returns true if the pane displays a button to maximize the pane.

**wxAuiPanelInfo::HasMinimizeButton****bool HasMinimizeButton() const**

HasMinimizeButton() returns true if the pane displays a button to minimize the pane.

**wxAuiPanelInfo::HasPinButton**

**bool HasPinButton() const**

HasPinButton() returns true if the pane displays a button to float the pane.

**wxAuiPanelInfo::Hide****wxAuiPanelInfo& Hide()**

Hide() indicates that a pane should be hidden.

**wxAuiPanelInfo::IsBottomDockable****bool IsBottomDockable() const**

IsBottomDockable() returns true if the pane can be docked at the bottom of the managed frame.

**wxAuiPanelInfo::IsDocked****bool IsDocked() const**

IsDocked() returns true if the pane is docked.

**wxAuiPanelInfo::IsFixed****bool IsFixed() const**

IsFixed() returns true if the pane cannot be resized.

**wxAuiPanelInfo::IsFloatable****bool IsFloatable() const**

IsFloatable() returns true if the pane can be undocked and displayed as a floating window.

**wxAuiPanelInfo::IsFloating****bool IsFloating() const**

IsFloating() returns true if the pane is floating.

**wxAuiPanelInfo::IsLeftDockable****bool IsLeftDockable() const**

IsLeftDockable() returns true if the pane can be docked on the left of the managed frame.

**wxAuiPanelInfo::IsMovable****bool IsMovable() const**

`IsMoveable()` returns true if the docked frame can be undocked or moved to another dock position.

### **`wxAuiPanelInfo::IsOk`**

#### **`bool IsOk() const`**

`IsOk()` returns true if the `wxAuiPanelInfo` structure is valid. A pane structure is valid if it has an associated window.

### **`wxAuiPanelInfo::IsResizable`**

#### **`bool IsResizable() const`**

`IsResizable()` returns true if the pane can be resized.

### **`wxAuiPanelInfo::IsRightDockable`**

#### **`bool IsRightDockable() const`**

`IsRightDockable()` returns true if the pane can be docked on the right of the managed frame.

### **`wxAuiPanelInfo::IsShown`**

#### **`bool IsShown() const`**

`IsShown()` returns true if the pane is currently shown.

### **`wxAuiPanelInfo::IsToolbar`**

#### **`bool IsToolbar() const`**

`IsToolbar()` returns true if the pane contains a toolbar.

### **`wxAuiPanelInfo::IsTopDockable`**

#### **`bool IsTopDockable() const`**

`IsTopDockable()` returns true if the pane can be docked at the top of the managed frame.

### **`wxAuiPanelInfo::Layer`**

#### **`wxAuiPanelInfo& Layer(int layer)`**

`Layer()` determines the layer of the docked pane. The dock layer is similar to an onion, the inner-most layer being layer 0. Each shell moving in the outward direction has a higher layer number. This allows for more complex docking layout formation.

### **`wxAuiPanelInfo::Left`**

**wxAuiPanelInfo& Left()**

Left() sets the pane dock position to the left side of the frame. This is the same thing as calling Direction(wxAUI\_DOCK\_LEFT).

**wxAuiPanelInfo::LeftDockable****wxAuiPanelInfo& LeftDockable(bool b = true)**

LeftDockable() indicates whether a pane can be docked on the left of the frame.

**wxAuiPanelInfo::MaxSize****wxAuiPanelInfo& MaxSize(const wxSize& size)****wxAuiPanelInfo& MaxSize(int x, int y)**

MaxSize() sets the maximum size of the pane.

**wxAuiPanelInfo::MaximizeButton****wxAuiPanelInfo& MaximizeButton(bool visible = true)**

MaximizeButton() indicates that a maximize button should be drawn for the pane.

**wxAuiPanelInfo::MinSize****wxAuiPanelInfo& MinSize(const wxSize& size)****wxAuiPanelInfo& MinSize(int x, int y)**

MinSize() sets the minimum size of the pane. Please note that this is only partially supported as of this writing.

**wxAuiPanelInfo::MinimizeButton****wxAuiPanelInfo& MinimizeButton(bool visible = true)**

MinimizeButton() indicates that a minimize button should be drawn for the pane.

**wxAuiPanelInfo::Movable****wxAuiPanelInfo& Movable(bool b = true)**

Movable indicates whether a frame can be moved.

**wxAuiPanelInfo::Name****wxAuiPanelInfo& Name(const wxString& n)**

Name() sets the name of the pane so it can be referenced in lookup functions. If a name is

not specified by the user, a random name is assigned to the pane when it is added to the manager.

### **wxAuiPanelInfo::PaneBorder**

**wxAuiPanelInfo& PaneBorder**(bool *visible = true*)

PaneBorder indicates that a border should be drawn for the pane.

### **wxAuiPanelInfo::PinButton**

**wxAuiPanelInfo& PinButton**(bool *visible = true*)

PinButton() indicates that a pin button should be drawn for the pane.

### **wxAuiPanelInfo::Position**

**wxAuiPanelInfo& Position**(int *pos*)

Position() determines the position of the docked pane.

### **wxAuiPanelInfo::Resizable**

**wxAuiPanelInfo& Resizable**(bool *resizable = true*)

Resizable() allows a pane to be resized if the parameter is true, and forces it to be a fixed size if the parameter is false. This is simply an antonym for Fixed().

### **wxAuiPanelInfo::Right**

**wxAuiPanelInfo& Right**()

Right() sets the pane dock position to the right side of the frame.

### **wxAuiPanelInfo::RightDockable**

**wxAuiPanelInfo& RightDockable**(bool *b = true*)

RightDockable() indicates whether a pane can be docked on the right of the frame.

### **wxAuiPanelInfo::Row**

**wxAuiPanelInfo& Row**(int *row*)

Row() determines the row of the docked pane.

### **wxAuiPanelInfo::SafeSet**

**void SafeSet**(wxAuiPanelInfo *source*)

Write the safe parts of a newly loaded `PanelInfo` structure "source" into "this" used on loading perspectives etc.

### **`wxAuiPanelInfo::SetFlag`**

**`wxAuiPanelInfo& SetFlag(unsigned int flag, bool option_state)`**

`SetFlag()` turns the property given by `flag` on or off with the `option_state` parameter.

### **`wxAuiPanelInfo::Show`**

**`wxAuiPanelInfo& Show(bool show = true)`**

`Show()` indicates that a pane should be shown.

### **`wxAuiPanelInfo::ToolbarPane`**

**`wxAuiPanelInfo& ToolbarPane()`**

`ToolbarPane()` specifies that the pane should adopt the default toolbar pane settings.

### **`wxAuiPanelInfo::Top`**

**`wxAuiPanelInfo& Top()`**

`Top()` sets the pane dock position to the top of the frame.

### **`wxAuiPanelInfo::TopDockable`**

**`wxAuiPanelInfo& TopDockable(bool b = true)`**

`TopDockable()` indicates whether a pane can be docked at the top of the frame.

### **`wxAuiPanelInfo::Window`**

**`wxAuiPanelInfo& Window(wxWindow* w)`**

`Window()` assigns the window pointer that the `wxAuiPanelInfo` should use. This normally does not need to be specified, as the window pointer is automatically assigned to the `wxAuiPanelInfo` structure as soon as it is added to the manager.

### **`wxAuiPanelInfo::operator=`**

**`wxAuiPanelInfo& operator operator=(const wxAuiPanelInfo& c)`**

Makes a copy of the `wxAuiPanelInfo` object.

## **`wxAutomationObject`**

The **wxAutomationObject** class represents an OLE automation object containing a single data member, an IDispatch pointer. It contains a number of functions that make it easy to perform automation operations, and set and get properties. The class makes heavy use of the *wxVariant* (p. 1769) class.

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

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

Note that this class obviously works under Windows only.

#### **Derived from**

*wxObject* (p. 1148)

#### **Include files**

<wx/msw/ole/automtn.h>

#### **See also**

*wxVariant* (p. 1769)

### **wxAutomationObject::wxAutomationObject**

**wxAutomationObject(WXIDISPATCH\* dispatchPtr = NULL)**

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

### **wxAutomationObject::~~wxAutomationObject**

**~wxAutomationObject()**

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

### **wxAutomationObject::CallMethod**

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

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

Calls an automation method for this object. The first form takes a method name, number of arguments, and an array of variants. The second form takes a method name and zero to



six constant references to variants. Since the variant class has constructors for the basic data types, and C++ provides temporary objects automatically, both of the following lines are syntactically valid:

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

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

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

### **wxAutomationObject::CreateInstance**

**bool CreateInstance(const wxString& classId) const**

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

### **wxAutomationObject::GetDispatchPtr**

**IDispatch\* GetDispatchPtr() const**

Gets the IDispatch pointer.

### **wxAutomationObject::GetInstance**

**bool GetInstance(const wxString& classId) const**

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

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

### **wxAutomationObject::GetObject**

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

Retrieves a property from this object, assumed to be a dispatch pointer, and initialises *obj* with it. To avoid having to deal with IDispatch pointers directly, use this function in preference to `wxAutomationObject::GetProperty` (p. 122) when retrieving objects from other objects.

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

**See also**

*wxAutomationObject::GetProperty* (p. 122)

**wxAutomationObject::GetProperty**

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

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

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

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

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

**wxAutomationObject::Invoke**

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

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

**Parameters**

*member*

The member function or property name.

*action*

Bitlist: may contain DISPATCH\_PROPERTYPUT, DISPATCH\_PROPERTYPUTREF, DISPATCH\_METHOD.

*retValue*

Return value (ignored if there is no return value)

.

*noArgs*

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

*args*

If non-null, contains an array of variants.

*ptrArgs*

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

### Return value

true if the operation was successful, false otherwise.

### Remarks

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

## **wxAutomationObject::PutProperty**

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

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

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

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

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

## **wxAutomationObject::SetDispatchPtr**

**void SetDispatchPtr(WXIDISPATCH\* *dispatchPtr*)**

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

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

## **wxBitmap**

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

### Derived from

*wxGDIObject* (p. 709)

*wxObject* (p. 1148)

### Include files

<wx/bitmap.h>

### Predefined objects

Objects:

### **wxNullBitmap**

### See also

*wxBitmap* overview (p. 2117), *supported bitmap file formats* (p. 2118), *wxDC::Blit* (p. 457), *wxIcon* (p. 894), *wxCursor* (p. 297), *wxBitmap* (p. 123), *wxMemoryDC* (p. 1069)

## **wxBitmap::wxBitmap**

### **wxBitmap()**

Default constructor.

### **wxBitmap(const wxBitmap& bitmap)**

Copy constructor, uses *reference counting* (p. 2046). To make a real copy, you can use:

```
wxBitmap newBitmap = oldBitmap.GetSubBitmap(  
                                                    wxRect(0, 0, oldBitmap.GetWidth(),  
oldBitmap.GetHeight()));
```

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

Creates a bitmap from the given data which is interpreted in platform-dependent manner.

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

Creates a bitmap from an array of bits.

You should only use this function for monochrome bitmaps (*depth* 1) in portable programs: in this case the *bits* parameter should contain an XBM image.

For other bit depths, the behaviour is platform dependent: under Windows, the data is passed without any changes to the underlying `CreateBitmap()` API. Under other platforms, only monochrome bitmaps may be created using this constructor and *wxImage* (p. 906) should be used for creating colour bitmaps from static data.

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

Creates a new bitmap. A depth of -1 indicates the depth of the current screen or visual. Some platforms only support 1 for monochrome and -1 for the current colour setting.

Beginning with version 2.5.4 of wxWidgets a depth of 32 including an alpha channel is supported under MSW, Mac and GTK+.

**wxBitmap(const char\* const\* bits)**

Creates a bitmap from XPM data.

**wxBitmap(const wxString& name, long type)**

Loads a bitmap from a file or resource.

**wxBitmap(const wxImage& img, int depth = -1)**

Creates bitmap object from the image. This has to be done to actually display an image as you cannot draw an image directly on a window. The resulting bitmap will use the provided colour depth (or that of the current system if depth is -1) which entails that a colour reduction has to take place.

When in 8-bit mode (PseudoColour mode), the GTK port will use a color cube created on program start-up to look up colors. This ensures a very fast conversion, but the image quality won't be perfect (and could be better for photo images using more sophisticated dithering algorithms).

On Windows, if there is a palette present (set with SetPalette), it will be used when creating the wxBitmap (most useful in 8-bit display mode). On other platforms, the palette is currently ignored.

### Parameters

*bits*

Specifies an array of pixel values.

*width*

Specifies the width of the bitmap.

*height*

Specifies the height of the bitmap.

*depth*

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

*name*

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

*type*

May be one of the following:

<code>wxBITMAP_TYPE_BMP</code>	Load a Windows bitmap file.
<code>wxBITMAP_TYPE_BMP_RESOURCE</code>	Load a Windows bitmap resource from the executable. Windows only.
<code>wxBITMAP_TYPE_PICT_RESOURCE</code>	Load a PICT image resource from the executable. Mac OS only.
<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.

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

In addition, wxBitmap can read all formats that *wxImage* (p. 906) can, which currently include `wxBITMAP_TYPE_JPEG`, `wxBITMAP_TYPE_TIF`, `wxBITMAP_TYPE_PNG`, `wxBITMAP_TYPE_GIF`, `wxBITMAP_TYPE_PCX`, and `wxBITMAP_TYPE_PNM`. Of course, you must have wxImage handlers loaded.

*img*

Platform-independent wxImage object.

### Remarks

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

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

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

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

The sixth form constructs a new bitmap.

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

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

```
#include "mybitmap.xpm"

...
```

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

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

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

### See also

`wxBitmap::LoadFile` (p. 131)

**wxPython note:** Constructors supported by wxPython are:

- |  |  |
|--|--|
| <b>wxBitmap(name, flag)</b>                            | Loads a bitmap from a file   |
| <b>wxEmptyBitmap(width, height, depth = -1)</b>        | Creates an empty bitmap with the given specifications                      |
| <b>wxBitmapFromXPMData(listOfStrings)</b>              | Create a bitmap from a Python list of strings whose contents are XPM data. |
| <b>wxBitmapFromBits(bits, width, height, depth=-1)</b> | Create a bitmap from an array of bits contained in a string.               |
| <b>wxBitmapFromImage(image, depth=-1)</b>              | Convert a wxImage to a wxBitmap.   |

**wxPerl note:** Constructors supported by wxPerl are:

- `Bitmap->new( width, height, depth = -1 )`
- `Bitmap->new( name, type )`
- `Bitmap->new( icon )`
- `Bitmap->newFromBits( bits, width, height, depth = 1 )`
- `Bitmap->newFromXPM( data )`

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

**~wxBitmap()**

Destructor. See *reference-counted object destruction* (p. 2046) for more info.

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

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

**wxBitmap::AddHandler****static void AddHandler(wxBitmapHandler\* handler)**

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

*handler*

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

**See also**

*wxBitmapHandler* (p. 146)

**wxBitmap::CleanUpHandlers****static void CleanUpHandlers()**

Deletes all bitmap handlers.

This function is called by wxWidgets on exit.

**wxBitmap::ConvertToImage****wxImage ConvertToImage()**

Creates an image from a platform-dependent bitmap. This preserves mask information so that bitmaps and images can be converted back and forth without loss in that respect.

**wxBitmap::CopyFromIcon****bool CopyFromIcon(const wxIcon& icon)**

Creates the bitmap from an icon.

**wxBitmap::Create****virtual bool Create(int width, int height, int depth = -1)**

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

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

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

**Parameters**

*width*

The width of the bitmap in pixels.



*height*

The height of the bitmap in pixels.

*depth*

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

*data*

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

*type*

A bitmap type identifier - see *wxBitmap::wxBitmap* (p. 124) for a list of possible values.

### **Return value**

true if the call succeeded, false otherwise.

### **Remarks**

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

### **See also**

*wxBitmap::wxBitmap* (p. 124)

### **wxBitmap::FindHandler**

**static wxBitmapHandler\* FindHandler(const wxString& *name*)**

Finds the handler with the given name.

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

Finds the handler associated with the given extension and type.

**static wxBitmapHandler\* FindHandler(wxBitmapType *bitmapType*)**

Finds the handler associated with the given bitmap type.

*name*

The handler name.

*extension*

The file extension, such as "bmp".

*bitmapType*

The bitmap type, such as `wxBITMAP_TYPE_BMP`.

**Return value**

A pointer to the handler if found, `NULL` otherwise.

**See also**

*wxBitmapHandler* (p. 146)

**wxBitmap::GetDepth**

**int GetDepth() const**

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

**wxBitmap::GetHandlers**

**static wxList& GetHandlers()**

Returns the static list of bitmap format handlers.

**See also**

*wxBitmapHandler* (p. 146)

**wxBitmap::GetHeight**

**int GetHeight() const**

Gets the height of the bitmap in pixels.

**wxBitmap::GetPalette**

**wxPalette\* GetPalette() const**

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

**See also**

*wxPalette* (p. 1166)

**wxBitmap::GetMask**

**wxMask\* GetMask() const**

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

**See also**

*wxBitmap::SetMask* (p. 134), *wxMask* (p. 1036)

### **wxBitmap::GetWidth**

**int GetWidth() const**

Gets the width of the bitmap in pixels.

#### **See also**

*wxBitmap::GetHeight* (p. 130)

### **wxBitmap::GetSubBitmap**

**wxBitmap GetSubBitmap(const wxRect&rect) const**

Returns a sub bitmap of the current one as long as the rect belongs entirely to the bitmap. This function preserves bit depth and mask information.

### **wxBitmap::InitStandardHandlers**

**static void InitStandardHandlers()**

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

This function is called by wxWidgets on startup.

#### **See also**

*wxBitmapHandler* (p. 146)

### **wxBitmap::InsertHandler**

**static void InsertHandler(wxBitmapHandler\* handler)**

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

*handler*

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

#### **See also**

*wxBitmapHandler* (p. 146)

### **wxBitmap::LoadFile**

**bool LoadFile(const wxString& name, wxBitmapType type)**

Loads a bitmap from a file or resource.

**Parameters***name*

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

*type*

One of the following values:

**wxBITMAP\_TYPE\_BMP**      Load a Windows bitmap file.

**wxBITMAP\_TYPE\_BMP\_RESOURCE**      Load a Windows bitmap resource from the executable.

**wxBITMAP\_TYPE\_PICT\_RESOURCE**      Load a PICT image resource from the executable. Mac OS only.

**wxBITMAP\_TYPE\_GIF**      Load a GIF bitmap file.

**wxBITMAP\_TYPE\_XBM**      Load an X bitmap file.

**wxBITMAP\_TYPE\_XPM**      Load an XPM bitmap file.

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

In addition, wxBitmap can read all formats that *wxImage* (p. 906) can (wxBITMAP\_TYPE\_JPEG, wxBITMAP\_TYPE\_PNG, wxBITMAP\_TYPE\_GIF, wxBITMAP\_TYPE\_PCX, wxBITMAP\_TYPE\_PNM). (Of course you must have wxImage handlers loaded.)

**Return value**

true if the operation succeeded, false otherwise.

**Remarks**

A palette may be associated with the bitmap if one exists (especially for colour Windows bitmaps), and if the code supports it. You can check if one has been created by using the *GetPalette* (p. 130) member.

**See also**

*wxBitmap::SaveFile* (p. 133)

**wxBitmap::IsOk**

**bool IsOk() const**

Returns true if bitmap data is present.

**wxBitmap::RemoveHandler**

**static bool RemoveHandler(const wxString& name)**

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

*name*

The handler name.

**Return value**

true if the handler was found and removed, false otherwise.

**See also**

*wxBitmapHandler* (p. 146)

**wxBitmap::SaveFile**

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

Saves a bitmap in the named file.

**Parameters**

*name*

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

*type*

One of the following values:

**wxBITMAP\_TYPE\_BMP**     Save a Windows bitmap file.

**wxBITMAP\_TYPE\_GIF**     Save a GIF bitmap file.

**wxBITMAP\_TYPE\_XBM**     Save an X bitmap file.

**wxBITMAP\_TYPE\_XPM**     Save an XPM bitmap file.

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

In addition, wxBitmap can save all formats that *wxImage* (p. 906) can (wxBITMAP\_TYPE\_JPEG, wxBITMAP\_TYPE\_PNG). (Of course you must have wxImage handlers loaded.)

*palette*

An optional palette used for saving the bitmap.

**Return value**

true if the operation succeeded, false otherwise.

**Remarks**

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

**See also**

*wxBitmap::LoadFile* (p. 131)

**wxBitmap::SetDepth**

**void SetDepth(int *depth*)**

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

**Parameters**

*depth*

Bitmap depth.

**wxBitmap::SetHeight**

**void SetHeight(int *height*)**

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

**Parameters**

*height*

Bitmap height in pixels.

**wxBitmap::SetMask**

**void SetMask(wxMask\* *mask*)**

Sets the mask for this bitmap.

**Remarks**

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

**See also**

*wxBitmap::GetMask* (p. 130), *wxMask* (p. 1036)

**wxBitmap::SetPalette**

**void SetPalette(const wxPalette& *palette*)**

Sets the associated palette. (Not implemented under GTK+).

**Parameters**

*palette*

The palette to set.

**See also**

*wxPalette* (p. 1166)

**wxBitmap::SetWidth**

**void SetWidth(int *width*)**

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

**Parameters**

*width*

Bitmap width in pixels.

**wxBitmap::operator =**

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

Assignment operator, using *reference counting* (p. 2046).

**Parameters**

*bitmap*

Bitmap to assign.

**Return value**

Returns 'this' object.

**wxBitmapComboBox**

A combobox that displays bitmap in front of the list items. It currently only allows using bitmaps of one size, and resizes itself so that a bitmap can be shown next to the text field.

**Derived from**

*wxComboBox* (p. 225)

*wxControlWithItems* (p. 286)

*wxControl* (p. 285)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

**Remarks**

While wxBitmapComboBox contains the *wxComboBox* (p. 225) API, but it might not

actually be derived from that class. In fact, if the platform does not have a native implementation, `wxBitmapComboBox` will inherit from `wxOwnerDrawnComboBox` (p. 1152). You can determine if the implementation is generic by checking whether `wxGENERIC_BITMAPCOMBOBOX` is defined.

#### Include files

`<wx/bmpcbox.h>`

#### Window styles

- |                           |  |
|---------------------------|--|
| <b>wxCB_READONLY</b>      | Creates a combobox without a text editor. On some platforms the control may appear very different when this style is used.   |
| <b>wxCB_SORT</b>          | Sorts the entries in the list alphabetically.  |
| <b>wxTE_PROCESS_ENTER</b> | The control will generate the event <code>wxEVT_COMMAND_TEXT_ENTER</code> (otherwise pressing Enter key is either processed internally by the control or used for navigation between dialog controls). Windows only. |

See also *window styles overview* (p. 2089).

#### Event handling

- |                                 |   |
|---------------------------------|---|
| <b>EVT_COMBOBOX(id, func)</b>   | Process a <code>wxEVT_COMMAND_COMBOBOX_SELECTED</code> event, when an item on the list is selected.                 |
| <b>EVT_TEXT(id, func)</b>       | Process a <code>wxEVT_COMMAND_TEXT_UPDATED</code> event, when the combobox text changes.                            |
| <b>EVT_TEXT_ENTER(id, func)</b> | Process a <code>wxEVT_COMMAND_TEXT_ENTER</code> event, when <code>&lt;RETURN&gt;</code> is pressed in the combobox. |

#### See also

`wxComboBox` (p. 225), `wxChoice` (p. 186), `wxOwnerDrawnComboBox` (p. 1152), `wxCommandEvent` (p. 250)

### **wxBitmapComboBox::wxBitmapComboBox**

#### **wxBitmapComboBox()**

Default constructor.

**wxBitmapComboBox(wxWindow\* parent, wxWindowID id, const wxString& value = "", const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, int n = 0, const wxString choices[] = NULL, long style = 0, const wxValidator& validator =**



*wxDefaultValidator*, **const wxString& name** = "comboBox")

**wxBitmapComboBox**(*wxWindow\* parent*, **wxWindowID id**, **const wxString& value**, **const wxPoint& pos**, **const wxSize& size**, **const wxString& choices**, **long style** = 0, **const wxValidator& validator** = *wxDefaultValidator*, **const wxString& name** = "comboBox")

Constructor, creating and showing a combobox.

### Parameters

*parent*

Parent window. Must not be NULL.

*id*

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

*value*

Initial selection string. An empty string indicates no selection.

*pos*

Window position.

*size*

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

*n*

Number of strings with which to initialise the control.

*choices*

An array of strings with which to initialise the control.

*style*

Window style. See *wxBitmapComboBox* (p. 135).

*validator*

Window validator.

*name*

Window name.

### See also

*wxBitmapComboBox::Create* (p. 138), *wxValidator* (p. 1767)

**wxBitmapComboBox::~~wxBitmapComboBox****~wxBitmapComboBox()**

Destructor, destroying the combobox.

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

Creates the combobox for two-step construction. Derived classes should call or replace this function. See *wxBitmapComboBox::wxBitmapComboBox* (p. 136) for further details.

**wxBitmapComboBox::Append****int Append(const wxString& item, const wxBitmap& bitmap = wxNullBitmap)**

Adds the item to the end of the combo box.

**int Append(const wxString& item, const wxBitmap& bitmap, void \*clientData)****int Append(const wxString& item, const wxBitmap& bitmap, wxClientData \*clientData)**

Adds the item to the end of the combo box, associating the given, typed or untyped, client data pointer with the item.

**wxBitmapComboBox::GetBitmapSize****wxSize GetBitmapSize() const**

Returns size of bitmaps used in the list.

**wxBitmapComboBox::GetItemBitmap****wxBitmap GetItemBitmap(unsigned int n) const**

Returns the bitmap of the item with the given index.

**wxBitmapComboBox::Insert****int Insert(const wxString& item, const wxBitmap& bitmap, unsigned int pos)**

Inserts the item into the list before pos. Not valid for `wxCB_SORT` style, use `Append` instead.

**int Insert(const wxString& item, const wxBitmap& bitmap, unsigned int pos, void \*clientData)**

**int Insert(const wxString& item, const wxBitmap& bitmap, unsigned int pos, wxClientData \*clientData)**

Inserts the item into the list before pos, associating the given, typed or untyped, client data pointer with the item. Not valid for `wxCB_SORT` style, use `Append` instead.

### **wxBitmapComboBox::SetItemBitmap**

**void SetItemBitmap(unsigned int n, const wxBitmap& bitmap)**

Sets the bitmap for the given item.

## **wxBitmapButton**

A bitmap button is a control that contains a bitmap. It may be placed on a *dialog box* (p. 496) or *panel* (p. 1170), or indeed almost any other window.

### **Derived from**

*wxButton* (p. 164)  
*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### **Include files**

<wx/bmpbuttn.h>

### **Remarks**

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

### **Button states**

This class supports bitmaps for several different states:

<b>normal</b>	This is the bitmap shown in the default state, it must be always valid while all the other bitmaps are optional and don't have to be set.
<b>disabled</b>	Bitmap shown when the button is disabled.
<b>selected</b>	Bitmap shown when the button is pushed (e.g. while the

user keeps the mouse button pressed on it)

<b>focus</b>	Bitmap shown when the button has keyboard focus but is not pressed.
<b>hover</b>	Bitmap shown when the mouse is over the button (but it is not pressed). Notice that if hover bitmap is not specified but the current platform UI uses hover images for the buttons (such as Windows XP or GTK+), then the focus bitmap is used for hover state as well. This makes it possible to set focus bitmap only to get reasonably good behaviour on all platforms.

### Window styles

<b>wxBU_AUTODRAW</b>	If this is specified, the button will be drawn automatically using the label bitmap only, providing a 3D-look border. If this style is not specified, the button will be drawn without borders and using all provided bitmaps. WIN32 only.
<b>wxBU_LEFT</b>	Left-justifies the bitmap label. WIN32 only.
<b>wxBU_TOP</b>	Aligns the bitmap label to the top of the button. WIN32 only.
<b>wxBU_RIGHT</b>	Right-justifies the bitmap label. WIN32 only.
<b>wxBU_BOTTOM</b>	Aligns the bitmap label to the bottom of the button. WIN32 only.

Note that **wxBU\_EXACTFIT** supported by *wxButton* (p. 164) is *not* used by this class as bitmap buttons don't have any minimal standard size by default.

See also *window styles overview* (p. 2089).

### Event handling

<b>EVT_BUTTON(id, func)</b>	Process a <code>wxEVT_COMMAND_BUTTON_CLICKED</code> event, when the button is clicked.
-----------------------------	--

### See also

*wxButton* (p. 164)

## **wxBitmapButton::wxBitmapButton**

### **wxBitmapButton()**

Default constructor.

**wxBitmapButton(wxWindow\* parent, wxWindowID id, const wxBitmap& bitmap, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long**

```
style = wxBU_AUTODRAW, const wxValidator& validator = wxDefaultValidator, const wxString& name = "button")
```

Constructor, creating and showing a button.

### Parameters

*parent*

Parent window. Must not be NULL.

*id*

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

*bitmap*

Bitmap to be displayed.

*pos*

Button position.

*size*

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

*style*

Window style. See *wxBitmapButton* (p. 139).

*validator*

Window validator.

*name*

Window name.

### Remarks

The *bitmap* parameter is normally the only bitmap you need to provide, and wxWidgets will draw the button correctly in its different states. If you want more control, call any of the functions *wxBitmapButton::SetBitmapSelected* (p. 144), *wxBitmapButton::SetBitmapFocus* (p. 143), *wxBitmapButton::SetBitmapDisabled* (p. 143).

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

### See also

*wxBitmapButton::Create* (p. 142), *wxValidator* (p. 1767)

### **wxBitmapButton::~wxBitmapButton**

**~wxBitmapButton()**

Destructor, destroying the button.

**wxBitmapButton::Create**

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

Button creation function for two-step creation. For more details, see *wxBitmapButton::wxBitmapButton* (p. 140).

**wxBitmapButton::GetBitmapDisabled**

**const wxBitmap& GetBitmapDisabled() const wxBitmap& GetBitmapDisabled()**

Returns the bitmap for the disabled state, may be invalid.

**Return value**

A reference to the disabled state bitmap.

**See also**

*wxBitmapButton::SetBitmapDisabled* (p. 143)

**wxBitmapButton::GetBitmapFocus**

**const wxBitmap& GetBitmapFocus() const wxBitmap& GetBitmapFocus()**

Returns the bitmap for the focused state, may be invalid.

**Return value**

A reference to the focused state bitmap.

**See also**

*wxBitmapButton::SetBitmapFocus* (p. 143)

**wxBitmapButton::GetBitmapHover**

**const wxBitmap& GetBitmapHover() const wxBitmap& GetBitmapHover()**

Returns the bitmap used when the mouse is over the button, may be invalid.

**See also**

*wxBitmapButton::SetBitmapHover* (p. 144)

**wxBitmapButton::GetBitmapLabel**

**const wxBitmap& GetBitmapLabel() const** wxBitmap& GetBitmapLabel()

Returns the label bitmap (the one passed to the constructor), always valid.

**Return value**

A reference to the button's label bitmap.

**See also**

*wxBitmapButton::SetBitmapLabel* (p. 144)

**wxBitmapButton::GetBitmapSelected**

**wxBitmap& GetBitmapSelected() const** wxBitmap& GetBitmapSelected()

Returns the bitmap for the pushed button state, may be invalid.

**Return value**

A reference to the selected state bitmap.

**See also**

*wxBitmapButton::SetBitmapSelected* (p. 144)

**wxBitmapButton::SetBitmapDisabled**

**void SetBitmapDisabled(const wxBitmap& *bitmap*)**

Sets the bitmap for the disabled button appearance.

**Parameters**

*bitmap*

The bitmap to set.

**See also**

*wxBitmapButton::GetBitmapDisabled* (p. 142), *wxBitmapButton::SetBitmapLabel* (p. 144), *wxBitmapButton::SetBitmapSelected* (p. 144), *wxBitmapButton::SetBitmapFocus* (p. 143)

**wxBitmapButton::SetBitmapFocus**

**void SetBitmapFocus(const wxBitmap& *bitmap*)**

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

**Parameters**

*bitmap*

The bitmap to set.

**See also**

*wxBitmapButton::GetBitmapFocus* (p. 142), *wxBitmapButton::SetBitmapLabel* (p. 144), *wxBitmapButton::SetBitmapSelected* (p. 144), *wxBitmapButton::SetBitmapDisabled* (p. 143)

**wxBitmapButton::SetBitmapHover**

**void SetBitmapHover(const wxBitmap& *bitmap*)**

Sets the bitmap to be shown when the mouse is over the button.

This function is new since wxWidgets version 2.7.0 and the hover bitmap is currently only supported in wxMSW.

**See also**

*wxBitmapButton::GetBitmapHover* (p. 142)

**wxBitmapButton::SetBitmapLabel**

**void SetBitmapLabel(const wxBitmap& *bitmap*)**

Sets the bitmap label for the button.

**Parameters**

*bitmap*

The bitmap label to set.

**Remarks**

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

**See also**

*wxBitmapButton::GetBitmapLabel* (p. 142)

**wxBitmapButton::SetBitmapSelected**

**void SetBitmapSelected(const wxBitmap& *bitmap*)**

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

**Parameters**

*bitmap*

The bitmap to set.

**See also**



*wxBitmapButton::GetBitmapSelected* (p. 143), *wxBitmapButton::SetBitmapLabel* (p. 144), *wxBitmapButton::SetBitmapFocus* (p. 143), *wxBitmapButton::SetBitmapDisabled* (p. 143)

## wxBitmapDataObject

`wxBitmapDataObject` is a specialization of `wxDataObject` for bitmap data. It can be used without change to paste data into the *wxClipboard* (p. 195) or a *wxDropSource* (p. 558). A user may wish to derive a new class from this class for providing a bitmap on-demand in order to minimize memory consumption when offering data in several formats, such as a bitmap and GIF.

**wxPython note:** If you wish to create a derived `wxBitmapDataObject` class in wxPython you should derive the class from `wxPyBitmapDataObject` in order to get Python-aware capabilities for the various virtual methods.

### Virtual functions to override

This class may be used as is, but *GetBitmap* (p. 145) may be overridden to increase efficiency.

### Derived from

*wxDataObjectSimple* (p. 335)  
*wxDataObject* (p. 311)

### Include files

<wx/dataobj.h>

### See also

*Clipboard and drag and drop overview* (p. 2150), *wxDataObject* (p. 311), *wxDataObjectSimple* (p. 335), *wxFileDataObject* (p. 599), *wxTextDataObject* (p. 1653), *wxDataObject* (p. 311)

**wxBitmapDataObject(const wxBitmap& bitmap = wxNullBitmap)**

Constructor, optionally passing a bitmap (otherwise use *SetBitmap* (p. 145) later).

### wxBitmapDataObject::GetBitmap

**virtual wxBitmap GetBitmap() const**

Returns the bitmap associated with the data object. You may wish to override this method when offering data on-demand, but this is not required by wxWidgets' internals. Use this method to get data in bitmap form from the *wxClipboard* (p. 195).

### wxBitmapDataObject::SetBitmap

**virtual void SetBitmap(const wxBitmap& bitmap)**

Sets the bitmap associated with the data object. This method is called when the data

object receives data. Usually there will be no reason to override this function.

## **wxBitmapHandler**

*Overview* (p. 2117)

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

If you wish to extend the capabilities of `wxBitmap`, derive a class from `wxBitmapHandler` and add the handler using `wxBitmap::AddHandler` (p. 128) in your application initialisation.

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/bitmap.h>

### **See also**

*wxBitmap* (p. 123), *wxIcon* (p. 894), *wxCursor* (p. 297)

## **wxBitmapHandler::wxBitmapHandler**

**wxBitmapHandler()**

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

## **wxBitmapHandler::~~wxBitmapHandler**

**~wxBitmapHandler()**

Destroys the `wxBitmapHandler` object.

## **wxBitmapHandler::Create**

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

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

### **Parameters**

*bitmap*

The `wxBitmap` object.

*width*

The width of the bitmap in pixels.

*height*

The height of the bitmap in pixels.

*depth*

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

*data*

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

*type*

A bitmap type identifier - see *wxBitmapHandler::wxBitmapHandler* (p. 124) for a list of possible values.

#### **Return value**

true if the call succeeded, false otherwise (the default).

#### **wxBitmapHandler::GetName**

**const wxString& GetName() const**

Gets the name of this handler.

#### **wxBitmapHandler::GetExtension**

**const wxString& GetExtension() const**

Gets the file extension associated with this handler.

#### **wxBitmapHandler::GetType**

**long GetType() const**

Gets the bitmap type associated with this handler.

#### **wxBitmapHandler::LoadFile**

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

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

#### **Parameters**

*bitmap*

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

*name*

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

*type*

See *wxBitmap::wxBitmap* (p. 124) for values this can take.

### Return value

true if the operation succeeded, false otherwise.

### See also

*wxBitmap::LoadFile* (p. 131)

*wxBitmap::SaveFile* (p. 133)

*wxBitmapHandler::SaveFile* (p. 148)

### **wxBitmapHandler::SaveFile**

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

Saves a bitmap in the named file.

### Parameters

*bitmap*

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

*name*

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

*type*

See *wxBitmap::wxBitmap* (p. 124) for values this can take.

*palette*

An optional palette used for saving the bitmap.

### Return value

true if the operation succeeded, false otherwise.

### See also

*wxBitmap::LoadFile* (p. 131)

*wxBitmap::SaveFile* (p. 133)

*wxBitmapHandler::LoadFile* (p. 147)

**wxBitmapHandler::SetName****void SetName(const wxString& name)**

Sets the handler name.

**Parameters***name*

Handler name.

**wxBitmapHandler::SetExtension****void SetExtension(const wxString& extension)**

Sets the handler extension.

**Parameters***extension*

Handler extension.

**wxBitmapHandler::SetType****void SetType(long type)**

Sets the handler type.

**Parameters***name*

Handler type.

**wxBoxSizer**

The basic idea behind a box sizer is that windows will most often be laid out in rather simple basic geometry, typically in a row or a column or several hierarchies of either.

For more information, please see *Programming with wxBoxSizer* (p. 2102).

**Derived from***wxSizer* (p. 1444)*wxObject* (p. 1148)**Include files**`<wx/sizer.h>`**See also**

*wxSizer* (p. 1444), *Sizer overview* (p. 2098)

### **wxBoxSizer::wxBoxSizer**

**wxBoxSizer**(int *orient*)

Constructor for a *wxBoxSizer*. *orient* may be either of *wxVERTICAL* or *wxHORIZONTAL* for creating either a column sizer or a row sizer.

### **wxBoxSizer::RecalcSizes**

**void RecalcSizes()**

Implements the calculation of a box sizer's dimensions and then sets the size of its children (calling *wxWindow::SetSize* (p. 1845) if the child is a window). It is used internally only and must not be called by the user. Documented for information.

### **wxBoxSizer::CalcMin**

**wxSize CalcMin()**

Implements the calculation of a box sizer's minimal. It is used internally only and must not be called by the user. Documented for information.

### **wxBoxSizer::GetOrientation**

**int GetOrientation()**

Returns the orientation of the box sizer, either *wxVERTICAL* or *wxHORIZONTAL*.

## **wxBrush**

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

### **Derived from**

*wxGDIObject* (p. 709)

*wxObject* (p. 1148)

### **Include files**

<wx/brush.h>

### **Predefined objects**

Objects:

**wxNullBrush**

Pointers:

**wxBLUE\_BRUSH**  
**wxGREEN\_BRUSH**  
**wxWHITE\_BRUSH**  
**wxBLACK\_BRUSH**  
**wxGREY\_BRUSH**  
**wxMEDIUM\_GREY\_BRUSH**  
**wxLIGHT\_GREY\_BRUSH**  
**wxTRANSPARENT\_BRUSH**  
**wxCYAN\_BRUSH**  
**wxRED\_BRUSH**

#### Remarks

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

Do not initialize objects on the stack before the program commences, since other required structures may not have been set up yet. Instead, define global pointers to objects and create them in *wxApp::OnInit* (p. 52) 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**.

This class uses *reference counting and copy-on-write* (p. 2046) internally so that assignments between two instances of this class are very cheap. You can therefore use actual objects instead of pointers without efficiency problems. If an instance of this class is changed it will create its own data internally so that other instances, which previously shared the data using the reference counting, are not affected.

#### See also

*wxBrushList* (p. 156), *wxDC* (p. 456), *wxDC::SetBrush* (p. 472)

### **wxBrush::wxBrush**

#### **wxBrush()**

Default constructor. The brush will be uninitialised, and *wxBrush::IsOk* (p. 154) will return false.

#### **wxBrush(const wxColour& colour, int style = wxSOLID)**

Constructs a brush from a colour object and style.

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

Constructs a brush from a colour name and style.

#### **wxBrush(const wxBitmap& stippleBitmap)**

Constructs a stippled brush using a bitmap.

**wxBrush(const wxBrush& brush)**

Copy constructor, uses *reference counting* (p. 2046).

### Parameters

*colour*

Colour object.

*colourName*

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

*style*

One of:

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

*brush*

Pointer or reference to a brush to copy.

*stippleBitmap*

A bitmap to use for stippling.

### Remarks

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

### See also

*wxBrushList* (p. 156), *wxColour* (p. 214), *wxColourDatabase* (p. 219)

**wxBrush::~wxBrush**



**~wxBrush()**

Destructor. See *reference-counted object destruction* (p. 2046) for more info.

**Remarks**

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

**wxBrush::GetColour****wxColour& GetColour() const**

Returns a reference to the brush colour.

**See also**

*wxBrush::SetColour* (p. 154)

**wxBrush::GetStipple****wxBitmap \* GetStipple() const**

Gets a pointer to the stipple bitmap. If the brush does not have a wxSTIPPLE style, this bitmap may be non-NULL but uninitialised (*wxBitmap::IsOk* (p. 132) returns false).

**See also**

*wxBrush::SetStipple* (p. 154)

**wxBrush::GetStyle****int GetStyle() const**

Returns the brush style, one of:

<b>wxTRANSPARENT</b>	Transparent (no fill).
<b>wxSOLID</b>	Solid.
<b>wxBDIAGONAL_HATCH</b>	Backward diagonal hatch.
<b>wxCROSSDIAG_HATCH</b>	Cross-diagonal hatch.
<b>wxFDIAGONAL_HATCH</b>	Forward diagonal hatch.
<b>wxCROSS_HATCH</b>	Cross hatch.
<b>wxHORIZONTAL_HATCH</b>	Horizontal hatch.
<b>wxVERTICAL_HATCH</b>	Vertical hatch.

**wxSTIPPLE**

Stippled using a bitmap.

**wxSTIPPLE\_MASK\_OPAQUE**

Stippled using a bitmap's mask.

**See also**

*wxBrush::SetStyle* (p. 155), *wxBrush::SetColour* (p. 154), *wxBrush::SetStipple* (p. 154)

**wxBrush::IsHatch****bool IsHatch() const**

Returns true if the style of the brush is any of hatched fills.

**See also**

*wxBrush::GetStyle* (p. 153)

**wxBrush::IsOk****bool IsOk() const**

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

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

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

**void SetColour(const wxString& colourName)**

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

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

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

**See also**

*wxBrush::GetColour* (p. 153)

**wxBrush::SetStipple****void SetStipple(const wxBitmap& bitmap)**

Sets the stipple bitmap.

**Parameters**

*bitmap*

The bitmap to use for stippling.

### Remarks

The style will be set to `wxSTIPPLE`, unless the bitmap has a mask associated to it, in which case the style will be set to `wxSTIPPLE_MASK_OPAQUE`.

If the `wxSTIPPLE` variant is used, the bitmap will be used to fill out the area to be drawn. If the `wxSTIPPLE_MASK_OPAQUE` is used, the current text foreground and text background determine what colours are used for displaying and the bits in the mask (which is a mono-bitmap actually) determine where to draw what.

Note that under Windows 95, only 8x8 pixel large stipple bitmaps are supported, Windows 98 and NT as well as GTK support arbitrary bitmaps.

### See also

*wxBitmap* (p. 123)

### **wxBrush::SetStyle**

**void SetStyle(int style)**

Sets the brush style.

*style*

One of:

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

### See also

*wxBrush::GetStyle* (p. 153)

**wxBrush::operator =****wxBrush& operator =(const wxBrush& brush)**Assignment operator, using *reference counting* (p. 2046).**wxBrush::operator ==****bool operator ==(const wxBrush& brush)**Equality operator. See *reference-counted object comparison* (p. 2046) for more info.**wxBrush::operator !=****bool operator !=(const wxBrush& brush)**Inequality operator. See *reference-counted object comparison* (p. 2046) for more info.**wxBrushList**

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

**Derived from***wxList* (p. 966)*wxObject* (p. 1148)**Include files**

&lt;wx/gdicmn.h&gt;

**Remarks**

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

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

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

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

**See also**

`wxBrush` (p. 150)

**wxBrushList::wxBrushList****void wxBrushList()**

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

**wxBrushList::FindOrCreateBrush****wxBrush \* FindOrCreateBrush(const wxColour& colour, int style = wxSOLID)**

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

**Parameters**

*colour*

Colour object.

*style*

Brush style. See `wxBrush::SetStyle` (p. 155) for a list of styles.

**wxBufferedDC**

This class provides a simple way to avoid flicker: when drawing on it, everything is in fact first drawn on an in-memory buffer (`awxBitmap` (p. 123)) and then copied to the screen, using the associated `wxDC`, only once, when this object is destroyed. `wxBufferedDC` itself is typically associated with `wxClientDC` (p. 193), if you want to use it in your `EVT_PAINT` handler, you should look at `wxBufferedPaintDC` (p. 159) instead.

When used like this, a valid *dc* must be specified in the constructor while the *buffer* bitmap doesn't have to be explicitly provided, by default this class will allocate the bitmap of required size itself. However using a dedicated bitmap can speed up the redrawing process by eliminating the repeated creation and destruction of a possibly big bitmap. Otherwise, `wxBufferedDC` can be used in the same way as any other device context.

There is another possible use for `wxBufferedDC` is to use it to maintain a backing store for the window contents. In this case, the associated *dc* may be `NULL` but a valid backing store bitmap should be specified.

Finally, please note that GTK+ 2.0 as well as OS X provide double buffering themselves natively. You can either use *wxWindow::IsDoubleBuffered* (p. 1823) to determine whether you need to use buffering or not, or use *wxAutoBufferedPaintDC* (p. 160) to avoid needless double buffering on the systems which already do it automatically.

### Derived from

*wxMemoryDC* (p. 1069)

*wxDC* (p. 456)

*wxObject* (p. 1148)

### Include files

<wx/dcbuffer.h>

### See also

*wxDC* (p. 456), *wxMemoryDC* (p. 1069), *wxBufferedPaintDC* (p. 159),  
*wxAutoBufferedPaintDC* (p. 160)

## wxBufferedDC::wxBufferedDC

### wxBufferedDC()

**wxBufferedDC(wxDC \*dc, const wxSize& area, int style = wxBUFFER\_CLIENT\_AREA)**

**wxBufferedDC(wxDC \*dc, wxBitmap& buffer, int style = wxBUFFER\_CLIENT\_AREA)**

If you use the first, default, constructor, you must call one of the *Init* (p. 159) methods later in order to use the object.

The other constructors initialize the object immediately and *Init()* must not be called after using them.

### Parameters

*dc*

The underlying DC: everything drawn to this object will be flushed to this DC when this object is destroyed. You may pass NULL in order to just initialize the buffer, and not flush it.

*area*

The size of the bitmap to be used for buffering (this bitmap is created internally when it is not given explicitly).

*buffer*

Explicitly provided bitmap to be used for buffering: this is the most efficient solution as the bitmap doesn't have to be recreated each time but it also requires more

memory as the bitmap is never freed. The bitmap should have appropriate size, anything drawn outside of its bounds is clipped.

*style*

wxBUFFER\_CLIENT\_AREA to indicate that just the client area of the window is buffered, or wxBUFFER\_VIRTUAL\_AREA to indicate that the buffer bitmap covers the virtual area (in which case PrepareDC is automatically called for the actual window device context).

### **wxBufferedDC::Init**

**void Init(wxDC \*dc, const wxSize& area, int style = wxBUFFER\_CLIENT\_AREA)**

**void Init(wxDC \*dc, wxBitmap& buffer, int style = wxBUFFER\_CLIENT\_AREA)**

These functions initialize the object created using the default constructor. Please see *constructors documentation* (p. 158) for details.

### **wxBufferedDC::~~wxBufferedDC**

Copies everything drawn on the DC so far to the underlying DC associated with this object, if any.

## **wxBufferedPaintDC**

This is a subclass of *wxBufferedDC* (p. 157) which can be used inside of an *OnPaint()* event handler. Just create an object of this class instead of *wxPaintDC* (p. 1164) and make sure *wxWindow::SetBackgroundStyle* (p. 1809) is called with *wxBG\_STYLE\_CUSTOM* somewhere in the class initialization code, and that's all you have to do to (mostly) avoid flicker. The only thing to watch out for is that if you are using this class together with *wxScrolledWindow* (p. 1414), you probably do **not** want to call *PrepareDC* (p. 1421) on it as it already does this internally for the real underlying *wxPaintDC*.

### **Derived from**

*wxBufferedDC* (p. 157)  
*wxMemoryDC* (p. 1069)  
*wxDC* (p. 456)  
*wxObject* (p. 1148)

### **Include files**

<wx/dcbuffer.h>

### **See also**

*wxDC* (p. 456), *wxBufferedDC* (p. 157), *wxAutoBufferedPaintDC* (p. 160)

## **wxBufferedPaintDC::wxBufferedPaintDC**

**wxBufferedPaintDC**(wxWindow \*window, wxBitmap& buffer, int style = wxBUFFER\_CLIENT\_AREA)

**wxBufferedPaintDC**(wxWindow \*window, int style = wxBUFFER\_CLIENT\_AREA)

As with *wxBufferedDC* (p. 158), you may either provide the bitmap to be used for buffering or let this object create one internally (in the latter case, the size of the client part of the window is used).

Pass `wxBUFFER_CLIENT_AREA` for the *style* parameter to indicate that just the client area of the window is buffered, or `wxBUFFER_VIRTUAL_AREA` to indicate that the buffer bitmap covers the virtual area (in which case `PrepareDC` is automatically called for the actual window device context).

## **wxBufferedPaintDC::~wxBufferedPaintDC**

Copies everything drawn on the DC so far to the window associated with this object, using a *wxPaintDC* (p. 1164).

## **wxAutoBufferedPaintDC**

This wxDC derivative can be used inside of an `OnPaint()` event handler to achieve double-buffered drawing. Just create an object of this class instead of *wxPaintDC* (p. 1164) and make sure *wxWindow::SetBackgroundStyle* (p. 1809) is called with `wxBG_STYLE_CUSTOM` somewhere in the class initialization code, and that's all you have to do to (mostly) avoid flicker.

The difference between *wxBufferedPaintDC* (p. 159) and this class, is the lightweighthness - on platforms which have native double-buffering, *wxAutoBufferedPaintDC* is simply a typedef of *wxPaintDC*. Otherwise, it is a typedef of *wxBufferedPaintDC*.

### **Derived from**

*wxBufferedPaintDC* (p. 159)  
*wxPaintDC* (p. 1164)  
*wxDC* (p. 456)  
*wxObject* (p. 1148)

### **Include files**

<wx/dcbuffer.h>

### **See also**

*wxDC* (p. 456), *wxBufferedPaintDC* (p. 159)

## **wxAutoBufferedPaintDC::wxAutoBufferedPaintDC**



**wxAutoBufferedPaintDC**(wxWindow \*window)

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

## wxBufferedInputStream

This stream acts as a cache. It caches the bytes read from the specified input stream (See *wxFilterInputStream* (p. 646)). It uses *wxStreamBuffer* and sets the default in-buffer size to 1024 bytes. This class may not be used without some other stream to read the data from (such as a file stream or a memory stream).

### Derived from

*wxFilterInputStream* (p. 646)

### Include files

<wx/stream.h>

### See also

*wxStreamBuffer* (p. 1547), *wxInputStream* (p. 941), *wxBufferedOutputStream* (p. 161)

## wxBufferedOutputStream

This stream acts as a cache. It caches the bytes to be written to the specified output stream (See *wxFilterOutputStream* (p. 647)). The data is only written when the cache is full, when the buffered stream is destroyed or when calling *SeekO()*.

This class may not be used without some other stream to write the data to (such as a file stream or a memory stream).

### Derived from

*wxFilterOutputStream* (p. 647)

### Include files

<wx/stream.h>

### See also

*wxStreamBuffer* (p. 1547), *wxOutputStream* (p. 1156)

## wxBufferedOutputStream::wxBufferedOutputStream

**wxBufferedOutputStream**(const wxOutputStream& parent)

Creates a buffered stream using a buffer of a default size of 1024 bytes for caching the stream *parent*.

**wxBufferedOutputStream::~~wxBufferedOutputStream****~wxBufferedOutputStream()**

Destructor. Calls `Sync()` and destroys the internal buffer.

**wxBufferedOutputStream::SeekO****off\_t SeekO(off\_t pos, wxSeekMode mode)**

Calls `Sync()` and changes the stream position.

**wxBufferedOutputStream::Sync****void Sync()**

Flushes the buffer and calls `Sync()` on the parent stream.

**wxBusyCursor**

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

For example:

```
wxBusyCursor wait;

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

It works by calling `wxBeginBusyCursor` (p. 1934) in the constructor, and `wxEndBusyCursor` (p. 1937) in the destructor.

**Derived from**

None

**Include files**

<wx/utils.h>

**See also**

`wxBeginBusyCursor` (p. 1934), `wxEndBusyCursor` (p. 1937), `wxWindowDisabler` (p. 1857)

**wxBusyCursor::wxBusyCursor****wxBusyCursor(wxCursor\* cursor = wxHOURLASS\_CURSOR)**

Constructs a busy cursor object, calling *wxBeginBusyCursor* (p. 1934).

### **wxBusyCursor::~~wxBusyCursor**

#### **~wxBusyCursor()**

Destroys the busy cursor object, calling *wxEndBusyCursor* (p. 1937).

## **wxBusyInfo**

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

For example:

```
wxBusyInfo wait("Please wait, working...");

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

It works by creating a window in the constructor, and deleting it in the destructor.

You may also want to call *wxTheApp->Yield()* to refresh the window periodically (in case it had been obscured by other windows, for example) like this:

```
wxWindowDisabler disableAll;

wxBusyInfo wait("Please wait, working...");

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

    if ( !(i % 1000) )
        wxTheApp->Yield();
}
```

but take care to not cause undesirable reentrancies when doing it (see *wxApp::Yield()* (p. 56) for more details). The simplest way to do it is to use *wxWindowDisabler* (p. 1857) class as illustrated in the above example.

### **Derived from**

None

### **Include files**

<wx/busyinfo.h>

**wxBusyInfo::wxBusyInfo****wxBusyInfo**(const wxString& msg, wxWindow\* parent = NULL)

Constructs a busy info window as child of *parent* and displays *msg* in it.

**NB:** If *parent* is not `NULL` you must ensure that it is not closed while the busy info is shown.

**wxBusyInfo::~~wxBusyInfo****~wxBusyInfo**()

Hides and closes the window containing the information text.

**wxButton**

A button is a control that contains a text string, and is one of the most common elements of a GUI. It may be placed on a *dialog box* (p. 496) or *panel* (p. 1170), or indeed almost any other window.

**Derived from***wxControl* (p. 285)*wxWindow* (p. 1795)*wxEvtHandler* (p. 576)*wxObject* (p. 1148)**Include files**

&lt;wx/button.h&gt;

**Window styles**

<b>wxBU_LEFT</b>	Left-justifies the label. Windows and GTK+ only.
<b>wxBU_TOP</b>	Aligns the label to the top of the button. Windows and GTK+ only.
<b>wxBU_RIGHT</b>	Right-justifies the bitmap label. Windows and GTK+ only.
<b>wxBU_BOTTOM</b>	Aligns the label to the bottom of the button. Windows and GTK+ only.
<b>wxBU_EXACTFIT</b>	Creates the button as small as possible instead of making it of the standard size (which is the default behaviour).
<b>wxNO_BORDER</b>	Creates a flat button. Windows and GTK+ only.

See also *window styles overview* (p. 2089).

**Event handling****EVT\_BUTTON**(id, func)                      Process a

`wxEVT_COMMAND_BUTTON_CLICKED`  
event, when the button is clicked.

### See also

*wxBitmapButton* (p. 139)

## **wxButton::wxButton**

### **wxButton()**

Default constructor.

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

Constructor, creating and showing a button.

The preferred way to create standard buttons is to use default value of *label*. If no label is supplied and *id* is one of standard IDs from *this list* (p. 2004), standard label will be used. In addition to that, the button will be decorated with stock icons under GTK+ 2.

### Parameters

*parent*

Parent window. Must not be NULL.

*id*

Button identifier. A value of `wxID_ANY` indicates a default value.

*label*

Text to be displayed on the button.

*pos*

Button position.

*size*

Button size. If the default size is specified then the button is sized appropriately for the text.

*style*

Window style. See *wxButton* (p. 164).

*validator*

Window validator.

*name*

Window name.

### See also

*wxButton::Create* (p. 166), *wxValidator* (p. 1767)

### **wxButton::~~wxButton**

**~wxButton()**

Destructor, destroying the button.

### **wxButton::Create**

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

Button creation function for two-step creation. For more details, see *wxButton::wxButton* (p. 165).

### **wxButton::GetLabel**

**wxString GetLabel() const**

Returns the string label for the button.

### Return value

The button's label.

### See also

*wxButton::SetLabel* (p. 167)

### **wxButton::GetDefaultSize**

**wxSize GetDefaultSize()**

Returns the default size for the buttons. It is advised to make all the dialog buttons of the same size and this function allows to retrieve the (platform and current font dependent size) which should be the best suited for this.

### **wxButton::SetDefault**

**void SetDefault()**

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

### Remarks

Under Windows, only dialog box buttons respond to this function. As normal under Windows and Motif, pressing return causes the default button to be depressed when the return key is pressed. See also *wxWindow::SetFocus* (p. 1839) which sets the keyboard focus for windows and text panel items, and *wxTopLevelWindow::SetDefaultItem* (p. 1716).

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: *wxWidgets* will then set the size of all buttons currently on the panel to the same size.

### wxButton::SetLabel

**void SetLabel(const wxString& label)**

Sets the string label for the button.

### Parameters

*label*

The label to set.

### See also

*wxButton::GetLabel* (p. 166)

## wxCalculateLayoutEvent

This event is sent by *wxLayoutAlgorithm* (p. 961) to calculate the amount of the remaining client area that the window should occupy.

### Derived from

*wxEvent* (p. 572)

*wxObject* (p. 1148)

### Include files

<wx/laywin.h>

### Event table macros

**EVT\_CALCULATE\_LAYOUT(func)**

Process a *wxEVT\_CALCULATE\_LAYOUT* event, which asks the window to take a 'bite' out of a rectangle provided by the algorithm.

### See also

*wxQueryLayoutInfoEvent* (p. 1238), *wxSashLayoutWindow* (p. 1394), *wxLayoutAlgorithm* (p. 961).

### **wxCalculateLayoutEvent::wxCalculateLayoutEvent**

**wxCalculateLayoutEvent(wxWindowID id = 0)**

Constructor.

### **wxCalculateLayoutEvent::GetFlags**

**int GetFlags() const**

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

### **wxCalculateLayoutEvent::GetRect**

**wxRect GetRect() const**

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

### **wxCalculateLayoutEvent::SetFlags**

**void SetFlags(int flags)**

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

### **wxCalculateLayoutEvent::SetRect**

**void SetRect(const wxRect& rect)**

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

## **wxCalendarCtrl**

The calendar control allows the user to pick a date. For this, it displays a window containing several parts: a control at the top to pick the month and the year (either or both of them may be disabled), and a month area below them which shows all the days in the month. The user can move the current selection using the keyboard and select the date (generating `EVT_CALENDAR` event) by pressing `<Return>` or double clicking it.

It has advanced possibilities for the customization of its display. All global settings (such as colours and fonts used) can, of course, be changed. But also, the display style for each day in the month can be set independently using *wxCalendarDateAttr* (p. 174) class.



An item without custom attributes is drawn with the default colours and font and without border, but setting custom attributes with *SetAttr* (p. 173) allows to modify its appearance. Just create a custom attribute object and set it for the day you want to be displayed specially (note that the control will take ownership of the pointer, i.e. it will delete it itself). A day may be marked as being a holiday, even if it is not recognized as one by *wxDatetime* (p. 2056) using *SetHoliday* (p. 175) method.

As the attributes are specified for each day, they may change when the month is changed, so you will often want to update them in `EVT_CALENDAR_MONTH` event handler.

### Derived from

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/calctrl.h>

### Window styles

**wxCAL\_SUNDAY\_FIRST** Show Sunday as the first day in the week

**wxCAL\_MONDAY\_FIRST** Show Monday as the first day in the week

**wxCAL\_SHOW\_HOLIDAYS** Highlight holidays in the calendar

**wxCAL\_NO\_YEAR\_CHANGE** Disable the year changing

**wxCAL\_NO\_MONTH\_CHANGE** Disable the month (and, implicitly, the year) changing

**wxCAL\_SHOW\_SURROUNDING\_WEEKS** Show the neighbouring weeks in the previous and next months

**wxCAL\_SEQUENTIAL\_MONTH\_SELECTION** Use alternative, more compact, style for the month and year selection controls.

The default calendar style is `wxCAL_SHOW_HOLIDAYS`.

### Event table macros

To process input from a calendar control, use these event handler macros to direct input to member functions that take a *wxCalendarEvent* (p. 176) argument.

**EVT\_CALENDAR(id, func)** A day was double clicked in the calendar.

**EVT\_CALENDAR\_SEL\_CHANGED(id, func)** The selected date changed.

**EVT\_CALENDAR\_DAY(id, func)** The selected day changed.

**EVT\_CALENDAR\_MONTH(id, func)** The selected month changed.

<b>EVT_CALENDAR_YEAR(id, func)</b>	The selected year changed.
<b>EVT_CALENDAR_WEEKDAY_CLICKED(id, func)</b>	User clicked on the week day header

Note that changing the selected date will result in either of `EVT_CALENDAR_DAY`, `MONTH` or `YEAR` events and `EVT_CALENDAR_SEL_CHANGED` one.

### Constants

The following are the possible return values for *HitTest* (p. 173) method:

```
enum wxCalendarHitTestResult
{
    wxCAL_HITTEST_NOWHERE,      // outside of anything
    wxCAL_HITTEST_HEADER,      // on the header (weekdays)
    wxCAL_HITTEST_DAY           // on a day in the calendar
}
```

### See also

*Calendar sample* (p. 2031)  
*wxCalendarDateAttr* (p. 174)  
*wxCalendarEvent* (p. 176)

## wxCalendarCtrl::wxCalendarCtrl

### wxCalendarCtrl()

Default constructor, use *Create* (p. 170) after it.

**wxCalendarCtrl(wxWindow\* parent, wxWindowID id, const wxDateTime& date = wxDefaultDateTime, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxCAL\_SHOW\_HOLIDAYS, const wxString& name = wxCalendarNameStr)**

Does the same as *Create* (p. 170) method.

### wxCalendarCtrl::Create

**bool Create(wxWindow\* parent, wxWindowID id, const wxDateTime& date = wxDefaultDateTime, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxCAL\_SHOW\_HOLIDAYS, const wxString& name = wxCalendarNameStr)**

Creates the control. See *wxWindow* (p. 1798) for the meaning of the parameters and the control overview for the possible styles.

## wxCalendarCtrl::~wxCalendarCtrl

**~wxCalendarCtrl()**

Destroys the control.

**wxCalendarCtrl::SetDate**

**void SetDate(const wxDateTime& date)**

Sets the current date.

**wxCalendarCtrl::GetDate**

**const wxDateTime& GetDate() const**

Gets the currently selected date.

**wxCalendarCtrl::EnableYearChange**

**void EnableYearChange(bool enable = true)**

This function should be used instead of changing `wxCAL_NO_YEAR_CHANGE` style bit directly. It allows or disallows the user to change the year interactively.

**wxCalendarCtrl::EnableMonthChange**

**void EnableMonthChange(bool enable = true)**

This function should be used instead of changing `wxCAL_NO_MONTH_CHANGE` style bit. It allows or disallows the user to change the month interactively. Note that if the month can not be changed, the year can not be changed neither.

**wxCalendarCtrl::EnableHolidayDisplay**

**void EnableHolidayDisplay(bool display = true)**

This function should be used instead of changing `wxCAL_SHOW_HOLIDAYS` style bit directly. It enables or disables the special highlighting of the holidays.

**wxCalendarCtrl::SetHeaderColours**

**void SetHeaderColours(const wxColour& colFg, const wxColour& colBg)**

Set the colours used for painting the weekdays at the top of the control.

**wxCalendarCtrl::GetHeaderColourFg**

**const wxColour& GetHeaderColourFg() const**

Gets the foreground colour of the header part of the calendar window.

**See also**

*SetHeaderColours* (p. 171)

**wxCalendarCtrl::GetHeaderColourBg**

**const wxColour& GetHeaderColourBg() const**

Gets the background colour of the header part of the calendar window.

**See also**

*SetHeaderColours* (p. 171)

**wxCalendarCtrl::SetHighlightColours**

**void SetHighlightColours(const wxColour& colFg, const wxColour& colBg)**

Set the colours to be used for highlighting the currently selected date.

**wxCalendarCtrl::GetHighlightColourFg**

**const wxColour& GetHighlightColourFg() const**

Gets the foreground highlight colour.

**See also**

*SetHighlightColours* (p. 172)

**wxCalendarCtrl::GetHighlightColourBg**

**const wxColour& GetHighlightColourBg() const**

Gets the background highlight colour.

**See also**

*SetHighlightColours* (p. 172)

**wxCalendarCtrl::SetHolidayColours**

**void SetHolidayColours(const wxColour& colFg, const wxColour& colBg)**

Sets the colours to be used for the holidays highlighting (only used if the window style includes `wxCAL_SHOW_HOLIDAYS` flag).

**wxCalendarCtrl::GetHolidayColourFg**

**const wxColour& GetHolidayColourFg() const**

Return the foreground colour currently used for holiday highlighting.

**See also**

*SetHolidayColours* (p. 172)

**wxCalendarCtrl::GetHolidayColourBg**

**const wxColour& GetHolidayColourBg() const**

Return the background colour currently used for holiday highlighting.

**See also**

*SetHolidayColours* (p. 172)

**wxCalendarCtrl::GetAttr**

**wxCalendarDateAttr \* GetAttr(size\_t day) const**

Returns the attribute for the given date (should be in the range 1...31).

The returned pointer may be `NULL`.

**wxCalendarCtrl::SetAttr**

**void SetAttr(size\_t day, wxCalendarDateAttr\* attr)**

Associates the attribute with the specified date (in the range 1...31).

If the pointer is `NULL`, the items attribute is cleared.

**wxCalendarCtrl::SetHoliday**

**void SetHoliday(size\_t day)**

Marks the specified day as being a holiday in the current month.

**wxCalendarCtrl::ResetAttr**

**void ResetAttr(size\_t day)**

Clears any attributes associated with the given day (in the range 1...31).

**wxCalendarCtrl::HitTest**

**wxCalendarHitTestResult HitTest(const wxPoint& pos, wxDateTime\* date = NULL, wxDateTime::WeekDay\* wd = NULL)**

Returns one of `wxCAL_HITTEST_XXX` constants (p. 168) and fills either *date* or *wd* pointer with the corresponding value depending on the hit test code.

## wxCalendarDateAttr

wxCalendarDateAttr is a custom attributes for a calendar date. The objects of this class are used with *wxCalendarCtrl* (p. 168).

### Derived from

No base class

### Constants

Here are the possible kinds of borders which may be used to decorate a date:

```
enum wxCalendarDateBorder
{
    wxCAL_BORDER_NONE,           // no border (default)
    wxCAL_BORDER_SQUARE,        // a rectangular border
    wxCAL_BORDER_ROUND          // a round border
}
```

### See also

*wxCalendarCtrl* (p. 168)

### Include files

<wx/calctrl.h>

## wxCalendarDateAttr::wxCalendarDateAttr

### wxCalendarDateAttr()

**wxCalendarDateAttr**(const wxColour& colText, const wxColour& colBack = wxNullColour, const wxColour& colBorder = wxNullColour, const wxFont& font = wxNullFont, wxCalendarDateBorder border = wxCAL\_BORDER\_NONE)

**wxCalendarDateAttr**(wxCalendarDateBorder border, const wxColour& colBorder = wxNullColour)

The constructors.

## wxCalendarDateAttr::SetTextColour

**void SetTextColour**(const wxColour& colText)

Sets the text (foreground) colour to use.

## wxCalendarDateAttr::SetBackgroundColour

**void SetBackgroundColour**(const wxColour& colBack)

Sets the text background colour to use.

**wxCalendarDateAttr::SetBorderColour****void SetBorderColour(const wxColour& col)**

Sets the border colour to use.

**wxCalendarDateAttr::SetFont****void SetFont(const wxFont& font)**

Sets the font to use.

**wxCalendarDateAttr::SetBorder****void SetBorder(wxCalendarDateBorder border)**

Sets the *border kind* (p. 174)

**wxCalendarDateAttr::SetHoliday****void SetHoliday(bool holiday)**

Display the date with this attribute as a holiday.

**wxCalendarDateAttr::HasTextColour****bool HasTextColour() const**

Returns `true` if this item has a non-default text foreground colour.

**wxCalendarDateAttr::HasBackgroundColour****bool HasBackgroundColour() const**

Returns `true` if this attribute specifies a non-default text background colour.

**wxCalendarDateAttr::HasBorderColour****bool HasBorderColour() const**

Returns `true` if this attribute specifies a non-default border colour.

**wxCalendarDateAttr::HasFont****bool HasFont() const**

Returns `true` if this attribute specifies a non-default font.

**wxCalendarDateAttr::HasBorder****bool HasBorder() const**

Returns `true` if this attribute specifies a non-default (i.e. any) border.

**wxCalendarDateAttr::IsHoliday****bool IsHoliday() const**

Returns `true` if this attribute specifies that this item should be displayed as a holiday.

**wxCalendarDateAttr::GetTextColour****const wxColour& GetTextColour() const**

Returns the text colour to use for the item with this attribute.

**wxCalendarDateAttr::GetBackgroundColour****const wxColour& GetBackgroundColour() const**

Returns the background colour to use for the item with this attribute.

**wxCalendarDateAttr::GetBorderColour****const wxColour& GetBorderColour() const**

Returns the border colour to use for the item with this attribute.

**wxCalendarDateAttr::GetFont****const wxFont& GetFont() const**

Returns the font to use for the item with this attribute.

**wxCalendarDateAttr::GetBorder****wxCalendarDateBorder GetBorder() const**

Returns the *border* (p. 174) to use for the item with this attribute.

**wxCalendarEvent**

The `wxCalendarEvent` class is used together with `wxCalendarCtrl` (p. 168).

**Derived from**

`wxDateEvent` (p. 339)

`wxCommandEvent` (p. 250)



*wxEvt* (p. 572)  
*wxObject* (p. 1148)

**Include files**

<wx/calctrl.h>

**See also**

*wxCalendarCtrl* (p. 168)

**wxCalendarEvent::GetWeekDay**

**wxDateTime::WeekDay GetWeekDay() const**

Returns the week day on which the user clicked in `EVT_CALENDAR_WEEKDAY_CLICKED` handler. It doesn't make sense to call this function in other handlers.

**wxCalendarEvent::SetWeekDay**

**void SetWeekDay(wxDateTime::WeekDay day)**

Sets the week day carried by the event, normally only used by the library internally.

**wxCaret**

A caret is a blinking cursor showing the position where the typed text will appear. The text controls usually have a caret but `wxCaret` class also allows to use a caret in other windows.

Currently, the caret appears as a rectangle of the given size. In the future, it will be possible to specify a bitmap to be used for the caret shape.

A caret is always associated with a window and the current caret can be retrieved using *wxWindow::GetCaret* (p. 1810). The same caret can't be reused in two different windows.

**Derived from**

No base class

**Include files**

<wx/caret.h>

**Data structures****wxCaret::wxCaret**

**wxCaret()**

Default constructor: you must use one of Create() functions later.

**wxCaret(wxWindow\* window, int width, int height)****wxCaret(wxWindowBase\* window, const wxSize& size)**

Create the caret of given (in pixels) width and height and associates it with the given window.

**wxCaret::Create****bool Create(wxWindowBase\* window, int width, int height)****bool Create(wxWindowBase\* window, const wxSize& size)**

Create the caret of given (in pixels) width and height and associates it with the given window (same as constructor).

**wxCaret::GetBlinkTime****static int GetBlinkTime()**

Returns the blink time which is measured in milliseconds and is the time elapsed between 2 inversions of the caret (blink time of the caret is the same for all carets, so this functions is static).

**wxCaret::GetPosition****void GetPosition(int\* x, int\* y) const****wxPoint GetPosition() const**

Get the caret position (in pixels).

**wxPerl note:** In wxPerl there are two methods instead of a single overloaded method:

**GetPosition()**Returns a `Wx::Point`**GetPositionXY()**Returns a 2-element list ( `x`, `y` )**wxCaret::GetSize****void GetSize(int\* width, int\* height) const****wxSize GetSize() const**

Get the caret size.

**wxPerl note:** In wxPerl there are two methods instead of a single overloaded method:

**GetSize()**Returns a `Wx::Size`**GetSizeWH()**Returns a 2-element list ( `width`,  
`height` )**wxCaret::GetWindow****wxWindow\* GetWindow() const**

Get the window the caret is associated with.

**wxCaret::Hide****void Hide()**Same as `wxCaret::Show(false)` (p. 180).**wxCaret::IsOk****bool IsOk() const**

Returns true if the caret was created successfully.

**wxCaret::IsVisible****bool IsVisible() const**

Returns true if the caret is visible and false if it is permanently hidden (if it is blinking and not shown currently but will be after the next blink, this method still returns true).

**wxCaret::Move****void Move(int x, int y)****void Move(const wxPoint& pt)**

Move the caret to given position (in logical coordinates).

**wxCaret::SetBlinkTime****static void SetBlinkTime(int milliseconds)**

Sets the blink time for all the carets.

**Remarks**Under Windows, this function will change the blink time for **all** carets permanently (until the next time it is called), even for the carets in other applications.**See also**

*GetBlinkTime* (p. 178)

### **wxCaret::SetSize**

**void SetSize**(int *width*, int *height*)

**void SetSize**(const wxSize& *size*)

Changes the size of the caret.

### **wxCaret::Show**

**void Show**(bool *show* = true)

Shows or hides the caret. Notice that if the caret was hidden N times, it must be shown N times as well to reappear on the screen.

## **wxCheckBox**

A checkbox is a labelled box which by default is either on (checkmark is visible) or off (no checkmark). Optionally (when the wxCHK\_3STATE style flag is set) it can have a third state, called the mixed or undetermined state. Often this is used as a "Does Not Apply" state.

### **Derived from**

*wxControl* (p. 285)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### **Include files**

<wx/checkbox.h>

### **Window styles**

#### **wxCHK\_2STATE**

Create a 2-state checkbox. This is the default.

#### **wxCHK\_3STATE**

Create a 3-state checkbox. Not implemented in wxMGL, wxOS2 and wxGTK built against GTK+ 1.2.

#### **wxCHK\_ALLOW\_3RD\_STATE\_FOR\_USER**

By default a user can't set a 3-state checkbox to the third state. It can only be done from code. Using this flags allows the user to set the checkbox to the third state by clicking.

#### **wxALIGN\_RIGHT**

Makes the text appear on the left of the checkbox.

See also *window styles overview* (p. 2089).

**Event handling****EVT\_CHECKBOX(id, func)**

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

**See also**

`wxRadioButton` (p. 1249), `wxCommandEvent` (p. 250)

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

Default constructor.

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

Constructor, creating and showing a checkbox.

**Parameters**

*parent*

Parent window. Must not be NULL.

*id*

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

*label*

Text to be displayed next to the checkbox.

*pos*

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

*size*

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

*style*

Window style. See `wxCheckBox` (p. 180).

*validator*

Window validator.

*name*

Window name.

**See also**

*wxCheckBox::Create* (p. 182), *wxValidator* (p. 1767)

**wxCheckBox::~~wxCheckBox**

**~wxCheckBox()**

Destructor, destroying the checkbox.

**wxCheckBox::Create**

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

Creates the checkbox for two-step construction. See *wxCheckBox::wxCheckBox* (p. 181) for details.

**wxCheckBox::GetValue**

**bool GetValue()** **const**

Gets the state of a 2-state checkbox.

**Return value**

Returns `true` if it is checked, `false` otherwise.

**wxCheckBox::Get3StateValue**

**wxCheckBoxState Get3StateValue()** **const**

Gets the state of a 3-state checkbox.

**Return value**

Returns `wxCHK_UNCHECKED` when the checkbox is unchecked, `wxCHK_CHECKED` when it is checked and `wxCHK_UNDETERMINED` when it's in the undetermined state. Asserts when the function is used with a 2-state checkbox.

**wxCheckBox::Is3rdStateAllowedForUser**

**bool Is3rdStateAllowedForUser()** **const**

Returns whether or not the user can set the checkbox to the third state.

**Return value**

Returns `true` if the user can set the third state of this checkbox, `false` if it can only be set programmatically or if it's a 2-state checkbox.

### **wxCheckBox::Is3State**

**bool Is3State() const**

Returns whether or not the checkbox is a 3-state checkbox.

#### **Return value**

Returns `true` if this checkbox is a 3-state checkbox, `false` if it's a 2-state checkbox.

### **wxCheckBox::IsChecked**

**bool IsChecked() const**

This is just a maybe more readable synonym for *GetValue* (p. 182): just as the latter, it returns `true` if the checkbox is checked and `false` otherwise.

### **wxCheckBox::SetValue**

**void SetValue(bool state)**

Sets the checkbox to the given state. This does not cause a `wxEVT_COMMAND_CHECKBOX_CLICKED` event to get emitted.

#### **Parameters**

*state*

If `true`, the check is on, otherwise it is off.

### **wxCheckBox::Set3StateValue**

**void Set3StateValue(const wxCheckBoxState state)**

Sets the checkbox to the given state. This does not cause a `wxEVT_COMMAND_CHECKBOX_CLICKED` event to get emitted.

#### **Parameters**

*state*

Can be one of: `wxCHK_UNCHECKED` (Check is off), `wxCHK_CHECKED` (Check is on) or `wxCHK_UNDETERMINED` (Check is mixed). Asserts when the checkbox is a 2-state checkbox and setting the state to `wxCHK_UNDETERMINED`.

## **wxCheckListBox**

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

When using this class under Windows `wxWidgets` must be compiled with `USE_OWNER_DRAWN` set to 1.

Only the new functions for this class are documented; see also `wxListBox` (p. 974).

Please note that `wxCheckListBox` uses client data in its implementation, and therefore this is not available to the application.

### Derived from

`wxListBox` (p. 974)  
`wxControl` (p. 285)  
`wxWindow` (p. 1795)  
`wxEvtHandler` (p. 576)  
`wxObject` (p. 1148)

### Include files

`<wx/checklst.h>`

### Window styles

See `wxListBox` (p. 974).

### Event handling

<b>EVT_CHECKLISTBOX(id, func)</b>	Process a <code>wxEVT_COMMAND_CHECKLISTBOX_TOGGLED</code> event, when an item in the check list box is checked or unchecked.
-----------------------------------	--

### See also

`wxListBox` (p. 974), `wxChoice` (p. 186), `wxComboBox` (p. 225), `wxListCtrl` (p. 980), `wxCommandEvent` (p. 250)

## **wxCheckListBox::wxCheckListBox**

### **wxCheckListBox()**

Default constructor.

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

**wxCheckListBox(wxWindow\* parent, wxWindowID id, const wxPoint& pos, const wxSize& size, const wxString& choices, long style = 0, const wxValidator& validator = wxDefaultValidator, const wxString& name = "listBox")**

Constructor, creating and showing a list box.



**Parameters***parent*

Parent window. Must not be NULL.

*id*

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

*pos*

Window position.

*size*

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

*n*

Number of strings with which to initialise the control.

*choices*

An array of strings with which to initialise the control.

*style*

Window style. See *wxCheckListBox* (p. 183).

*validator*

Window validator.

*name*

Window name.

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

**wxPerl note:** In wxPerl there is just an array reference in place of *nand choices*.

**wxCheckListBox::~wxCheckListBox****void ~wxCheckListBox()**

Destructor, destroying the list box.

**wxCheckListBox::Check**

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

Checks the given item. Note that calling this method doesn't result in `wxEVT_COMMAND_CHECKLISTBOX_TOGGLE` being emitted.

### Parameters

*item*

Index of item to check.

*check*

true if the item is to be checked, false otherwise.

### **wxCheckListBox::IsChecked**

**bool IsChecked(unsigned int *item*) const**

Returns true if the given item is checked, false otherwise.

### Parameters

*item*

Index of item whose check status is to be returned.

## **wxChoice**

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

### Derived from

*wxControlWithItems* (p. 286)

*wxControl* (p. 285)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### Include files

<wx/choice.h>

### Window styles

There are no special styles for `wxChoice`.

See also *window styles overview* (p. 2089).

### Event handling

**EVT\_CHOICE(id, func)**

Process a `wxEVT_COMMAND_CHOICE_SELECTED` event, when an item on the list is selected.

**See also**

*wxListBox* (p. 974), *wxComboBox* (p. 225), *wxCommandEvent* (p. 250)

**wxChoice::wxChoice****wxChoice()**

Default constructor.

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

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

Constructor, creating and showing a choice.

**Parameters**

*parent*

Parent window. Must not be NULL.

*id*

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

*pos*

Window position.

*size*

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

*n*

Number of strings with which to initialise the choice control.

*choices*

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

*style*

Window style. See *wxChoice* (p. 186).

*validator*

Window validator.

*name*

Window name.

### See also

*wxChoice::Create* (p. 188), *wxValidator* (p. 1767)

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

**wxPerl note:** In *wxPerl* there is just an array reference in place of *nand choices*.

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

**~wxChoice()**

Destructor, destroying the choice item.

### **wxChoice::Create**

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

**bool Create(wxWindow \*parent, wxWindowID id, const wxPoint& pos, const wxSize& size, const wxString& choices, long style = 0, const wxValidator& validator = wxDefaultValidator, const wxString& name = "choice")**

Creates the choice for two-step construction. See *wxChoice::wxChoice* (p. 187).

### **wxChoice::GetColumns**

**int GetColumns() const**

Gets the number of columns in this choice item.

### Remarks

This is implemented for Motif only and always returns 1 for the other platforms.

### **wxChoice::GetCurrentSelection**

**int GetCurrentSelection() const**

Unlike *GetSelection* (p. 289) which only returns the accepted selection value, i.e. the selection in the control once the user closes the dropdown list, this function returns the current selection. That is, while the dropdown list is shown, it returns the currently selected item in it. When it is not shown, its result is the same as for the other function.

This function is new since wxWidgets version 2.6.2 (before this version *GetSelection* (p. 289) itself behaved like this).

### **wxChoice::SetColumns**

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

Sets the number of columns in this choice item.

#### **Parameters**

*n*

Number of columns.

#### **Remarks**

This is implemented for Motif only and doesn't do anything under other platforms.

## **wxChoicebook**

wxChoicebook is a class similar to *wxNotebook* (p. 1135) but which uses a *wxChoice* (p. 186) to show the labels instead of the tabs.

There is no documentation for this class yet but its usage is identical to *wxNotebook* (except for the features clearly related to tabs only), so please refer to that class documentation for now. You can also use the *notebook sample* (p. 2036) to see wxChoicebook in action.

wxChoicebook allows the use of *wxBookCtrl::GetControlSizer*, allowing a program to add other controls next to the choice control. This is particularly useful when screen space is restricted, as it often is when wxChoicebook is being employed.

#### **Derived from**

wxBookCtrlBase  
*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

#### **Include files**

<wx/choicebk.h>

#### **Window styles**

- |                      |  |
|----------------------|--|
| <b>wxCHB_DEFAULT</b> | Choose the default location for the labels depending on the current platform (left everywhere except Mac where it is top). |
| <b>wxCHB_TOP</b>     | Place labels above the page area.  |

<b>wxCHB_LEFT</b>	Place labels on the left side.
<b>wxCHB_RIGHT</b>	Place labels on the right side.
<b>wxCHB_BOTTOM</b>	Place labels below the page area.

**See also**

*wxBookCtrl* (p. 2127), *wxNotebook* (p. 1135), *notebook sample* (p. 2036)

## wxClassInfo

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

**Derived from**

No parent class.

**Include files**

<wx/object.h>

**See also**

*Overview* (p. 2045), *wxObject* (p. 1148)

### wxClassInfo::wxClassInfo

**wxClassInfo(const wxChar \* className, const wxClassInfo \* baseClass1, const wxClassInfo \* 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() const**

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

### wxClassInfo::FindClass

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

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

**wxClassInfo::GetBaseClassName1****wxChar \* GetBaseClassName1() const**

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

**wxClassInfo::GetBaseClassName2****wxChar \* GetBaseClassName2() const**

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

**wxClassInfo::GetClassName****wxChar \* GetClassName() const**

Returns the string form of the class name.

**wxClassInfo::GetSize****int GetSize() const**

Returns the size of the class.

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

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

**wxClassInfo::IsDynamic****bool IsDynamic() const**

Returns true if this class info can create objects of the associated class.

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

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

**wxClient**

A wxClient object represents the client part of a client-server DDE-like (Dynamic Data Exchange) conversation. The actual DDE-based implementation using wxDDEClient is available on Windows only, but a platform-independent, socket-based version of this API is available using wxTCPClient, which has the same API.

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 `wxClient` is required so that a user-overridden `wxClient::OnMakeConnection` (p. 192) member can return a `wxConnection` of the required class, when a connection is made. Look at the IPC sample and the *Interprocess communications overview* (p. 2175) for an example of how to do this.

### Derived from

`wxClientBase`  
*wxObject* (p. 1148)

### Include files

`<wx/ipc.h>`

### See also

*wxServer* (p. 1431), *wxConnection* (p. 276), *Interprocess communications overview* (p. 2175)

## **wxClient::wxClient**

### **wxClient()**

Constructs a client object.

## **wxClient::MakeConnection**

**wxConnectionBase \* MakeConnection(const wxString& host, const wxString& service, const wxString& topic)**

Tries to make a connection with a server by host (machine name under UNIX - use 'localhost' for same machine; ignored when using native DDE in Windows), service name 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` (p. 192) member to return your own derived connection object.

Under Unix, the service name may be either an integer port identifier in which case an Internet domain socket will be used for the communications, or a valid file name (which shouldn't exist and will be deleted afterwards) in which case a Unix domain socket is created.

**SECURITY NOTE:** Using Internet domain sockets is extremely insecure for IPC as there is absolutely no access control for them, use Unix domain sockets whenever possible!

## **wxClient::OnMakeConnection**



**wxConnectionBase \* OnMakeConnection()**

Called by *wxClient::MakeConnection* (p. 192), by default this simply returns a new *wxConnection* object. Override this method to return a *wxConnection* descendant customised for the application.

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* (p. 278). You may also want to store application-specific data in instances of the new class.

**wxClient::ValidHost****bool ValidHost(const wxString& host)**

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

**wxClientDC**

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

To draw on a window from within **OnPaint**, construct a *wxPaintDC* (p. 1164) object.

To draw on the whole window including decorations, construct a *wxWindowDC* (p. 1855) object (Windows only).

**Derived from**

*wxWindowDC* (p. 1855)  
*wxDC* (p. 456)*wxObject* (p. 1148)

**Include files**

<wx/dcclient.h>

**See also**

*wxDC* (p. 456), *wxMemoryDC* (p. 1069), *wxPaintDC* (p. 1164), *wxWindowDC* (p. 1855), *wxScreenDC* (p. 1407)

**wxClientDC::wxClientDC****wxClientDC(wxWindow\* window)**

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

**wxClientData**

All classes deriving from *wxEvtHandler* (p. 576) (such as all controls and *wxApp* (p. 45)) can hold arbitrary data which is here referred to as "client data". This is useful e.g. for scripting languages which need to handle shadow objects for most of *wxWidgets'* classes and which store a handle to such a shadow class as client data in that class. This data can either be of type *void* - in which case the *datacontainer* does not take care of freeing the data again or it is of type *wxClientData* or its derivatives. In that case the container (e.g. a control) will free the memory itself later. Note that you *must not* assign both *void* data and data derived from the *wxClientData* class to a container.

Some controls can hold various items and these controls can additionally hold client data for each item. This is the case for *wxChoice* (p. 186), *wxComboBox* (p. 225) and *wxListBox* (p. 974). *wxTreeCtrl* (p. 1728) has a specialized class *wxTreeItemData* (p. 1751) for each item in the tree.

If you want to add client data to your own classes, you may use the mix-in class *wxClientDataContainer* (p. 194).

#### **Include files**

<wx/clntdata.h>

#### **See also**

*wxEvtHandler* (p. 576), *wxTreeItemData* (p. 1751), *wxStringClientData* (p. 1581), *wxClientDataContainer* (p. 194)

### **wxClientData::wxClientData**

**wxClientData()**

Constructor.

### **wxClientData::~~wxClientData**

**~wxClientData()**

Virtual destructor.

## **wxClientDataContainer**

This class is a mixin that provides storage and management of "client data." This data can either be of type *void* - in which case the *datacontainer* does not take care of freeing the data again or it is of type *wxClientData* or its derivatives. In that case the container will free the memory itself later. Note that you *must not* assign both *void* data and data derived from the *wxClientData* class to a container.

NOTE: This functionality is currently duplicated in *wxEvtHandler* in order to avoid having more than one vtable in that class hierarchy.

**See also**

*wxEvtHandler* (p. 576), *wxClientData* (p. 193)

**Derived from**

No base class

**Include files**

<wx/clntdata.h>

**Data structures****wxClientDataContainer::wxClientDataContainer**

**wxClientDataContainer()**

**wxClientDataContainer::~~wxClientDataContainer**

**~wxClientDataContainer()**

**wxClientDataContainer::GetClientData**

**void\* GetClientData() const**

Get the untyped client data.

**wxClientDataContainer::GetClientObject**

**wxClientData\* GetClientObject() const**

Get a pointer to the client data object.

**wxClientDataContainer::SetClientData**

**void SetClientData(void\* data)**

Set the untyped client data.

**wxClientDataContainer::SetClientObject**

**void SetClientObject(wxClientData\* data)**

Set the client data object. Any previous object will be deleted.

**wxClipboard**

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

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

See also the *`wxDataObject` overview* (p. 2151) for further information.

Call `wxClipboard::Open` (p. 198) to get ownership of the clipboard. If this operation returns true, you now own the clipboard. Call `wxClipboard::SetData` (p. 198) to put data on the clipboard, or `wxClipboard::GetData` (p. 197) to retrieve data from the clipboard. Call `wxClipboard::Close` (p. 197) to close the clipboard and relinquish ownership. You should keep the clipboard open only momentarily.

For example:

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

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

### Derived from

*`wxObject`* (p. 1148)

### Include files

`<wx/clipbrd.h>`

### See also

*Drag and drop overview* (p. 2150), *`wxDataObject`* (p. 311)

## **`wxClipboard::wxClipboard`**

### **`wxClipboard()`**

Constructor.

**wxClipboard::~wxClipboard****~wxClipboard()**

Destructor.

**wxClipboard::AddData****bool AddData(wxDataObject\* data)**

Call this function to add the data object to the clipboard. You may call this function repeatedly after having cleared the clipboard using *wxClipboard::Clear* (p. 197).

After this function has been called, the clipboard owns the data, so do not delete the data explicitly.

**See also**

*wxClipboard::SetData* (p. 198)

**wxClipboard::Clear****void Clear()**

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

**wxClipboard::Close****void Close()**

Call this function to close the clipboard, having opened it with *wxClipboard::Open* (p. 198).

**wxClipboard::Flush****bool Flush()**

Flushes the clipboard: this means that the data which is currently on clipboard will stay available even after the application exits (possibly eating memory), otherwise the clipboard will be emptied on exit. Returns false if the operation is unsuccessful for any reason.

**wxClipboard::GetData****bool GetData(wxDataObject& data)**

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

**wxClipboard::IsOpened****bool IsOpened() const**

Returns true if the clipboard has been opened.

### **wxClipboard::IsSupported**

**bool IsSupported(const wxDataFormat& *format*)**

Returns true if there is data which matches the data format of the given data object currently **available** (IsSupported sounds like a misnomer, FIXME: better deprecate this name?) on the clipboard.

### **wxClipboard::Open**

**bool Open()**

Call this function to open the clipboard before calling *wxClipboard::SetData* (p. 198) and *wxClipboard::GetData* (p. 197).

Call *wxClipboard::Close* (p. 197) when you have finished with the clipboard. You should keep the clipboard open for only a very short time.

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

### **wxClipboard::SetData**

**bool SetData(wxDataObject\* *data*)**

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

After this function has been called, the clipboard owns the data, so do not delete the data explicitly.

### **See also**

*wxClipboard::AddData* (p. 197)

### **wxClipboard::UsePrimarySelection**

**void UsePrimarySelection(bool *primary* = true)**

On platforms supporting it (currently only GTK), selects the so called PRIMARY SELECTION as the clipboard as opposed to the normal clipboard, if *primary* is true.

## **wxClipboardTextEvent**

This class represents the events generated by a control (typically a *wxTextCtrl* (p. 1633) but other windows can generate these events as well) when its content gets copied or cut to, or pasted from the clipboard. There are three types of corresponding events *wxEVT\_COMMAND\_TEXT\_COPY*, *wxEVT\_COMMAND\_TEXT\_CUT* and *wxEVT\_COMMAND\_TEXT\_PASTE*.

If any of these events is processed (without being skipped) by an event handler, the corresponding operation doesn't take place which allows to prevent the text from being copied from or pasted to a control. It is also possible to examine the clipboard contents in the PASTE event handler and transform it in some way before inserting in a control -- for example, changing its case or removing invalid characters.

Finally notice that a CUT event is always preceded by the COPY event which makes it possible to only process the latter if it doesn't matter if the text was copied or cut.

### Remarks

These events are currently only generated by *wxTextCtrl* (p. 1633) under GTK. They are generated by all controls under Windows. (Prior to version 2.8.8, only *wxComboBox* and *wxTextCtrl* without *wxTE\_RICH* style generated them under Windows.)

### Derived from

*wxCommandEvent* (p. 250)

*wxEvent* (p. 572)

*wxObject* (p. 1148)

### Include files

<wx/event.h>

### Event handling

To process this type of events use the following event handling macros. The *func* parameter must be a member functions that takes an argument of type `wxClipboardTextEvent &`.

#### **EVT\_TEXT\_COPY(id, func)**

Some or all of the controls content was copied to the clipboard.

#### **EVT\_TEXT\_CUT(id, func)**

Some or all of the controls content was cut (i.e. copied and deleted).

#### **EVT\_TEXT\_PASTE(id, func)**

Clipboard content was pasted into the control.

### See also

*wxClipboard* (p. 195)

### **wxClipboardTextEvent::wxClipboardTextEvent**

**wxClipboardTextEvent(wxEvtType *commandType* = wxEVT\_NULL, int *id* = 0)**

## wxCloseEvent

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

The handler function for `EVT_CLOSE` is called when the user has tried to close a frame or dialog box using the window manager (X) or system menu (Windows). It can also be invoked by the application itself programmatically, for example by calling the `wxWindow::Close` (p. 1802) function.

You should check whether the application is forcing the deletion of the window using `wxCloseEvent::CanVeto` (p. 201). If this is `false`, you *must* destroy the window using `wxWindow::Destroy` (p. 1804). If the return value is `true`, it is up to you whether you respond by destroying the window.

If you don't destroy the window, you should call `wxCloseEvent::Veto` (p. 201) to let the calling code know that you did not destroy the window. This allows the `wxWindow::Close` (p. 1802) function to return `true` or `false` depending on whether the close instruction was honoured or not.

### Derived from

`wxEvent` (p. 572)

### Include files

`<wx/event.h>`

### Event table macros

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

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

### See also

`wxWindow::Close` (p. 1802), *Window deletion overview* (p. 2089)

## wxCloseEvent::wxCloseEvent

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



Constructor.

### **wxCloseEvent::CanVeto**

#### **bool CanVeto()**

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

### **wxCloseEvent::GetLoggingOff**

#### **bool GetLoggingOff() const**

Returns true if the user is just logging off or false if the system is shutting down. This method can only be called for end session and query end session events, it doesn't make sense for close window event.

### **wxCloseEvent::SetCanVeto**

#### **void SetCanVeto(bool *canVeto*)**

Sets the 'can veto' flag.

### **wxCloseEvent::SetForce**

#### **void SetForce(bool *force*) const**

Sets the 'force' flag.

### **wxCloseEvent::SetLoggingOff**

#### **void SetLoggingOff(bool *loggingOff*) const**

Sets the 'logging off' flag.

### **wxCloseEvent::Veto**

#### **void Veto(bool *veto* = true)**

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

You can only veto a shutdown if *wxCloseEvent::CanVeto* (p. 201) returns true.

## **wxCmdLineParser**

*wxCmdLineParser* is a class for parsing the command line.

It has the following features:

1. distinguishes options, switches and parameters; allows option grouping
2. allows both short and long options
3. automatically generates the usage message from the command line description
4. does type checks on the options values (number, date, ...).

To use it you should follow these steps:

1. *construct* (p. 204) an object of this class giving it the command line to parse and optionally its description or use `AddXXX()` functions later
2. call `Parse()`
3. use `Found()` to retrieve the results

In the documentation below the following terminology is used:

switch	This is a boolean option which can be given or not, but which doesn't have any value. We use the word switch to distinguish such boolean options from more generic options like those described below. For example, <code>-v</code> might be a switch meaning "enable verbose mode".
option	Option for us here is something which comes with a value 0 unlike a switch. For example, <code>-o:filename</code> might be an option which allows to specify the name of the output file.
parameter	This is a required program argument.

### **Derived from**

No base class

### **Include files**

<wx/cmdline.h>

### **Constants**

The structure `wxCmdLineEntryDesc` is used to describe the one command line switch, option or parameter. An array of such structures should be passed to *SetDesc()* (p. 208). Also, the meanings of parameters of the `AddXXX()` functions are the same as of the corresponding fields in this structure:

```
struct wxCmdLineEntryDesc
{
    wxCmdLineEntryType kind;
    const wxChar *shortName;
    const wxChar *longName;
    const wxChar *description;
    wxCmdLineParamType type;
```

```
        int flags;
    };
```

The type of a command line entity is in the `kind` field and may be one of the following constants:

```
enum wxCmdLineEntryType
{
    wxCMD_LINE_SWITCH,
    wxCMD_LINE_OPTION,
    wxCMD_LINE_PARAM,
    wxCMD_LINE_NONE           // use this to terminate the list
}
```

The field `shortName` is the usual, short, name of the switch or the option. `longName` is the corresponding long name or `NULL` if the option has no long name. Both of these fields are unused for the parameters. Both the short and long option names can contain only letters, digits and the underscores.

`description` is used by the `Usage()` (p. 209) method to construct a help message explaining the syntax of the program.

The possible values of `type` which specifies the type of the value accepted by an option or parameter are:

```
enum wxCmdLineParamType
{
    wxCMD_LINE_VAL_STRING,    // default
    wxCMD_LINE_VAL_NUMBER,
    wxCMD_LINE_VAL_DATE,
    wxCMD_LINE_VAL_NONE
}
```

Finally, the `flags` field is a combination of the following bit masks:

```
enum
{
    wxCMD_LINE_OPTION_MANDATORY = 0x01, // this option must be given
    wxCMD_LINE_PARAM_OPTIONAL   = 0x02, // the parameter may be omitted
    wxCMD_LINE_PARAM_MULTIPLE   = 0x04, // the parameter may be
repeated
    wxCMD_LINE_OPTION_HELP      = 0x08, // this option is a help request
    wxCMD_LINE_NEEDS_SEPARATOR = 0x10, // must have sep before the
value
}
```

Notice that by default (i.e. if flags are just 0), options are optional (sic) and each call to `AddParam()` (p. 209) allows one more parameter - this may be changed by giving non-default flags to it, i.e. use `wxCMD_LINE_OPTION_MANDATORY` to require that the

option is given and `wxCMD_LINE_PARAM_OPTIONAL` to make a parameter optional. Also, `wxCMD_LINE_PARAM_MULTIPLE` may be specified if the programs accepts a variable number of parameters - but it only can be given for the last parameter in the command line description. If you use this flag, you will probably need to use *GetParamCount* (p. 210) to retrieve the number of parameters effectively specified after calling *Parse* (p. 209).

The last flag `wxCMD_LINE_NEEDS_SEPARATOR` can be specified to require a separator (either a colon, an equal sign or white space) between the option name and its value. By default, no separator is required.

### See also

*wxApp::argc* (p. 46) and *wxApp::argv* (p. 46)  
console sample

## Construction

Before *Parse* (p. 209) can be called, the command line parser object must have the command line to parse and also the rules saying which switches, options and parameters are valid - this is called command line description in what follows.

You have complete freedom of choice as to when specify the required information, the only restriction is that it must be done before calling *Parse* (p. 209).

To specify the command line to parse you may use either one of constructors accepting it (*wxCmdLineParser(argc, argv)* (p. 205) or *wxCmdLineParser* (p. 206) usually) or, if you use *the default constructor* (p. 205), you can do it later by calling *SetCmdLine* (p. 206).

The same holds for command line description: it can be specified either in the constructor (*without command line* (p. 206) or *together with it* (p. 206)) or constructed later using either *SetDesc* (p. 208) or combination of *AddSwitch* (p. 208), *AddOption* (p. 208) and *AddParam* (p. 209) methods.

Using constructors or *SetDesc* (p. 208) uses a (usually `const static`) table containing the command line description. If you want to decide which options to accept during the run-time, using one of the *AddXXX()* functions above might be preferable.

## Customization

*wxCmdLineParser* has several global options which may be changed by the application. All of the functions described in this section should be called before *Parse* (p. 209).

First global option is the support for long (also known as GNU-style) options. The long options are the ones which start with two dashes ("--") and look like this: `--verbose`, i.e. they generally are complete words and not some abbreviations of them. As long options are used by more and more applications, they are enabled by default, but may be disabled with *DisableLongOptions* (p. 207).

Another global option is the set of characters which may be used to start an option (otherwise, the word on the command line is assumed to be a parameter). Under Unix,

' - ' is always used, but Windows has at least two common choices for this: ' - ' and ' / '. Some programs also use ' + '. The default is to use what suits most the current platform, but may be changed with *SetSwitchChars* (p. 207) method.

Finally, *SetLogo* (p. 207) can be used to show some application-specific text before the explanation given by *Usage* (p. 209) function.

### Parsing command line

After the command line description was constructed and the desired options were set, you can finally call *Parse* (p. 209) method. It returns 0 if the command line was correct and was parsed, -1 if the help option was specified (this is a separate case as, normally, the program will terminate after this) or a positive number if there was an error during the command line parsing.

In the latter case, the appropriate error message and usage information are logged by *wxCmdLineParser* itself using the standard *wxWidgets* logging functions.

### Getting results

After calling *Parse* (p. 209) (and if it returned 0), you may access the results of parsing using one of overloaded *Found( )* methods.

For a simple switch, you will simply call *Found* (p. 209) to determine if the switch was given or not, for an option or a parameter, you will call a version of *Found( )* which also returns the associated value in the provided variable. All *Found( )* functions return true if the switch or option were found in the command line or false if they were not specified.

### **wxCmdLineParser::wxCmdLineParser**

**wxCmdLineParser()**

Default constructor. You must use *SetCmdLine* (p. 206) later.

### **wxCmdLineParser::wxCmdLineParser**

**wxCmdLineParser(int argc, char\*\* argv)**

**wxCmdLineParser(int argc, wchar\_t\*\* argv)**

Constructor specifies the command line to parse. This is the traditional (Unix) command line format. The parameters *argc* and *argv* have the same meaning as for *main( )* function.

The second overloaded constructor is only available in Unicode build. The first one is available in both ANSI and Unicode modes because under some platforms the command line arguments are passed as ASCII strings even to Unicode programs.

**wxCmdLineParser::wxCmdLineParser****wxCmdLineParser(const wxString& cmdline)**

Constructor specifies the command line to parse in Windows format. The parameter *cmdline* has the same meaning as the corresponding parameter of `WinMain()`.

**wxCmdLineParser::wxCmdLineParser****wxCmdLineParser(const wxCmdLineEntryDesc\* desc)**

Same as *wxCmdLineParser* (p. 205), but also specifies the *command line description* (p. 208).

**wxCmdLineParser::wxCmdLineParser****wxCmdLineParser(const wxCmdLineEntryDesc\* desc, int argc, char\*\* argv)**

Same as *wxCmdLineParser* (p. 205), but also specifies the *command line description* (p. 208).

**wxCmdLineParser::wxCmdLineParser****wxCmdLineParser(const wxCmdLineEntryDesc\* desc, const wxString& cmdline)**

Same as *wxCmdLineParser* (p. 206), but also specifies the *command line description* (p. 208).

**wxCmdLineParser::ConvertStringToArgs****static wxArrayString ConvertStringToArgs(const wxChar \*cmdline)**

Breaks down the string containing the full command line in words. The words are separated by whitespace. The quotes can be used in the input string to quote the white space and the back slashes can be used to quote the quotes.

**wxCmdLineParser::SetCmdLine****void SetCmdLine(int argc, char\*\* argv)****void SetCmdLine(int argc, wchar\_t\*\* argv)**

Set command line to parse after using one of the constructors which don't do it. The second overload of this function is only available in Unicode build.

**See also***wxCmdLineParser* (p. 205)**wxCmdLineParser::SetCmdLine**

**void SetCmdLine(const wxString& cmdline)**

Set command line to parse after using one of the constructors which don't do it.

**See also**

*wxCmdLineParser* (p. 206)

**wxCmdLineParser::~~wxCmdLineParser**

**~wxCmdLineParser()**

Frees resources allocated by the object.

**NB:** destructor is not virtual, don't use this class polymorphically.

**wxCmdLineParser::SetSwitchChars**

**void SetSwitchChars(const wxString& switchChars)**

*switchChars* contains all characters with which an option or switch may start. Default is " - " for Unix, " - / " for Windows.

**wxCmdLineParser::EnableLongOptions**

**void EnableLongOptions(bool enable = true)**

Enable or disable support for the long options.

As long options are not (yet) POSIX-compliant, this option allows to disable them.

**See also**

*Customization* (p. 204) and *AreLongOptionsEnabled* (p. 207)

**wxCmdLineParser::DisableLongOptions**

**void DisableLongOptions()**

Identical to *EnableLongOptions(false)* (p. 207).

**wxCmdLineParser::AreLongOptionsEnabled**

**bool AreLongOptionsEnabled()**

Returns true if long options are enabled, otherwise false.

**See also**

*EnableLongOptions* (p. 207)

**wxCmdLineParser::SetLogo**

**void SetLogo(const wxString& logo)**

*logo* is some extra text which will be shown by *Usage* (p. 209) method.

**wxCmdLineParser::SetDesc**

**void SetDesc(const wxCmdLineEntryDesc\* desc)**

Construct the command line description

Take the command line description from the wxCMD\_LINE\_NONE terminated table.

Example of usage:

```
static const wxCmdLineEntryDesc cmdLineDesc[] =
{
    { wxCMD_LINE_SWITCH, "v", "verbose", "be verbose" },
    { wxCMD_LINE_SWITCH, "q", "quiet", "be quiet" },

    { wxCMD_LINE_OPTION, "o", "output", "output file" },
    { wxCMD_LINE_OPTION, "i", "input", "input dir" },
    { wxCMD_LINE_OPTION, "s", "size", "output block size",
wxCMD_LINE_VAL_NUMBER },
    { wxCMD_LINE_OPTION, "d", "date", "output file date",
wxCMD_LINE_VAL_DATE },

    { wxCMD_LINE_PARAM, NULL, NULL, "input file",
wxCMD_LINE_VAL_STRING, wxCMD_LINE_PARAM_MULTIPLE },

    { wxCMD_LINE_NONE }
};

wxCmdLineParser parser;

parser.SetDesc(cmdLineDesc);
```

**wxCmdLineParser::AddSwitch**

**void AddSwitch(const wxString& name, const wxString& lng = wxEmptyString, const wxString& desc = wxEmptyString, int flags = 0)**

Add a switch *name* with an optional long name *lng* (no long name if it is empty, which is default), description *desc* and flags *flags* to the command line description.

**wxCmdLineParser::AddOption**

**void AddOption(const wxString& name, const wxString& lng = wxEmptyString, const wxString& desc = wxEmptyString, wxCmdLineParamType type = wxCMD\_LINE\_VAL\_STRING, int flags = 0)**

Add an option *name* with an optional long name *lng* (no long name if it is empty, which is default) taking a value of the given type (string by default) to the command line description.



**wxCmdLineParser::AddParam**

**void AddParam(const wxString& desc = wxEmptyString, wxCmdLineParamType type = wxCMD\_LINE\_VAL\_STRING, int flags = 0)**

Add a parameter of the given *type* to the command line description.

**wxCmdLineParser::Parse**

**int Parse(bool giveUsage = true)**

Parse the command line, return 0 if ok, -1 if "-h" or "--help" option was encountered and the help message was given or a positive value if a syntax error occurred.

**Parameters**

*giveUsage*

If *true* (default), the usage message is given if a syntax error was encountered while parsing the command line or if help was requested. If *false*, only error messages about possible syntax errors are given, use *Usage* (p. 209) to show the usage message from the caller if needed.

**wxCmdLineParser::Usage**

**void Usage()**

Give the standard usage message describing all program options. It will use the options and parameters descriptions specified earlier, so the resulting message will not be helpful to the user unless the descriptions were indeed specified.

**See also**

*SetLogo* (p. 207)

**wxCmdLineParser::Found**

**bool Found(const wxString& name) const**

Returns true if the given switch was found, false otherwise.

**wxCmdLineParser::Found**

**bool Found(const wxString& name, wxString\* value) const**

Returns true if an option taking a string value was found and stores the value in the provided pointer (which should not be NULL).

**wxCmdLineParser::Found**

**bool Found(const wxString& name, long\* value) const**

Returns true if an option taking an integer value was found and stores the value in the provided pointer (which should not be NULL).

### **wxCmdLineParser::Found**

**bool Found(const wxString& name, wxDateTime\* value) const**

Returns true if an option taking a date value was found and stores the value in the provided pointer (which should not be NULL).

### **wxCmdLineParser::GetParamCount**

**size\_t GetParamCount() const**

Returns the number of parameters found. This function makes sense mostly if you had used `wxCMD_LINE_PARAM_MULTIPLE` flag.

### **wxCmdLineParser::GetParam**

**wxString GetParam(size\_t n = 0u) const**

Returns the value of Nth parameter (as string only for now).

### **See also**

*GetParamCount* (p. 210)

## **wxCollapsiblePane**

A collapsible pane is a container with an embedded button-like control which can be used by the user to collapse or expand the pane's contents.

Once constructed you should use the *GetPane* (p. 213) function to access the pane and add your controls inside it (i.e. use the *GetPane* (p. 213)'s returned pointer as parent for the controls which must go in the pane, NOT the `wxCollapsiblePane` itself!).

Note that because of its nature of control which can dynamically (and drastically) change its size at run-time under user-input, when putting `wxCollapsiblePane` inside a `wxSizer` (p. 1444) you should be careful to add it with a proportion value of zero; this is because otherwise all other windows with non-null proportion values would automatically get resized each time the user expands or collapse the pane window resulting usually in a weird, flickering effect.

Usage sample:

```
wxCollapsiblePane *collpane = new wxCollapsiblePane(this,
wxID_ANY, wxT("Details:"));

// add the pane with a zero proportion value to the 'sz' sizer which
contains it
sz->Add(collpane, 0, wxGROW|wxALL, 5);
```

```
// now add a test label in the collapsible pane using a sizer to
layout it:
wxWindow *win = collpane->GetPane();
wxSizer *paneSz = new wxBoxSizer(wxVERTICAL);
paneSz->Add(new wxStaticText(win, wxID_ANY, wxT("test!")), 1,
wxGROW|wxALL, 2);
win->SetSizer(paneSz);
paneSz->SetSizeHints(win);
```

It is only available if `wxUSE_COLLPANE` is set to 1 (the default).

### Derived from

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/collpane.h>

### Window styles

**wxCP\_DEFAULT\_STYLE** The default style: 0.

### Event handling

To process a collapsible pane event, use these event handler macros to direct input to member functions that take a *wxCollapsiblePaneEvent* (p. 213) argument.

**EVT\_COLLAPSIBLEPANE\_CHANGED(id, func)** The user showed or hidden the collapsible pane.

### See also

*wxPanel* (p. 1170),  
*wxCollapsiblePaneEvent* (p. 213)

### **wxCollapsiblePane::wxCollapsiblePane**

**wxCollapsiblePane(wxWindow \*parent, wxWindowID id, const wxString& label, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxCP\_DEFAULT\_STYLE, const wxValidator& validator = wxDefaultValidator, const wxString& name = "collapsiblePane")**

Initializes the object and calls *Create* (p. 211) with all the parameters.

### **wxCollapsiblePane::Create**

```
bool Create(wxWindow *parent, wxWindowID id, const wxString& label, const  
wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxCP_DEFAULT_STYLE, const wxValidator& validator = wxDefaultValidator, const  
wxString& name = "collapsiblePane")
```

### Parameters

*parent*

Parent window, must not be non-NULL.

*id*

The identifier for the control.

*label*

The initial label shown in the button which allows the user to expand or collapse the pane window.

*pos*

Initial position.

*size*

Initial size.

*style*

The window style, see `wxCP_*` flags.

*validator*

Validator which can be used for additional date checks.

*name*

Control name.

### Return value

`true` if the control was successfully created or `false` if creation failed.

### **wxCollapsiblePane::IsCollapsed**

```
bool IsCollapsed() const
```

Returns `true` if the pane window is currently hidden.

### **wxCollapsiblePane::IsExpanded**

```
bool IsExpanded() const
```

Returns `true` if the pane window is currently shown.

### **wxCollapsiblePane::Collapse**

**void Collapse**(bool *collapse* = *true*)

Collapses or expands the pane window.

### **wxCollapsiblePane::Expand**

**void Expand**()

Same as *Collapse* (p. 213)(*false*).

### **wxCollapsiblePane::GetPane**

**wxWindow \* GetPane()** const

Returns a pointer to the pane window. Add controls to the returned *wxWindow* (p. 1795) to make them collapsible.

## **wxCollapsiblePaneEvent**

This event class is used for the events generated by *wxCollapsiblePane* (p. 210).

### **Derived from**

*wxCommandEvent* (p. 250)

*wxEvent* (p. 572)

*wxObject* (p. 1148)

### **Include files**

<wx/collpane.h>

### **Event handling**

To process input from a *wxCollapsiblePane*, use one of these event handler macros to direct input to member function that take a *wxCollapsiblePaneEvent* (p. 213) argument:

**EVT\_COLLAPSIBLEPANE\_CHANGED(id, func)** The user showed or hidden the collapsible pane.

### **See also**

*wxCollapsiblePane* (p. 210)

### **wxCollapsiblePaneEvent::wxCollapsiblePaneEvent**

**wxCollapsiblePaneEvent**(wxObject \* *generator*, int *id*, bool *collapsed*)

The constructor is not normally used by the user code.

### **wxCollapsiblePaneEvent::GetCollapsed**

**bool GetCollapsed() const**

Returns `true` if the pane has been collapsed.

### **wxCollapsiblePaneEvent::SetCollapsed**

**void SetCollapsed(bool collapsed)**

Sets this as a collapsed pane event (if *collapsed* is `true`) or as an expanded pane event (if *collapsed* is `false`).

## **wxColour**

A colour is an object representing a combination of Red, Green, and Blue (RGB) intensity values, and is used to determine drawing colours. See the entry for *wxColourDatabase* (p. 219) 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.

You can retrieve the current system colour settings with *wxSystemSettings* (p. 1594).

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/colour.h>

### **Predefined objects**

Objects:

#### **wxNullColour**

Pointers:

**wxBLACK**

**wxWHITE**

**wxRED**

**wxBLUE**

**wxGREEN**

**wxCYAN**

**wxLIGHT\_GREY**

**See also**

*wxColourDatabase* (p. 219), *wxPen* (p. 1175), *wxBrush* (p. 150), *wxColourDialog* (p. 221), *wxSystemSettings* (p. 1594)

## **wxColour::wxColour**

### **wxColour()**

Default constructor.

**wxColour(unsigned char *red*, unsigned char *green*, unsigned char *blue*, unsigned char *alpha*=wxALPHA\_OPAQUE)**

Constructs a colour from red, green, blue and alpha values.

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

Constructs a colour using the given string. See *Set* (p. 217) for more info.

**wxColour(const wxColour& *colour*)**

Copy constructor.

### **Parameters**

*red*

The red value.

*green*

The green value.

*blue*

The blue value.

*alpha*

The alpha value. Alpha values range from 0 (wxALPHA\_TRANSPARENT) to 255 (wxALPHA\_OPAQUE).

*colourName*

The colour name.

*colour*

The colour to copy.

### **See also**

*wxColourDatabase* (p. 219)

**wxPython note:** Constructors supported by wxPython are:

**wxColour(red=0, green=0, blue=0)**

**wxNamedColour(name)**

### **wxColour::Alpha**

**unsigned char Alpha() const**

Returns the alpha value, on platforms where alpha is not yet supported, this always returns wxALPHA\_OPAQUE.

### **wxColour::Blue**

**unsigned char Blue() const**

Returns the blue intensity.

### **wxColour::GetAsString**

**wxString GetAsString(long flags) const**

Converts this colour to a *wxString* (p. 1553) using the given *flags*.

The supported flags are **wxC2S\_NAME**, to obtain the colour name (e.g. `wxColour(255,0,0) -> "red"`), **wxC2S\_CSS\_SYNTAX**, to obtain the colour in the `"rgb(r,g,b)"` syntax (e.g. `wxColour(255,0,0) -> "rgb(255,0,0)"`), and **wxC2S\_HTML\_SYNTAX**, to obtain the colour as `"#"` followed by 6 hexadecimal digits (e.g. `wxColour(255,0,0) -> "#FF0000"`).

This function never fails and always returns a non-empty string.

This function is new since wxWidgets version 2.7.0

### **wxColour::GetPixel**

**long GetPixel() const**

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

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

### **wxColour::Green**

**unsigned char Green() const**

Returns the green intensity.



**wxColour::IsOk****bool IsOk() const**

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

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

Returns the red intensity.

**wxColour::Set****void Set(unsigned char *red*, unsigned char *green*, unsigned char *blue*, unsigned char *alpha=wxALPHA\_OPAQUE*)****void Set(unsigned long *RGB*)****bool Set(const wxString & *str*)**

Sets the RGB intensity values using the given values (first overload), extracting them from the packed long (second overload), using the given string (third overload).

When using third form, Set() accepts: colour names (those listed in *wxTheColourDatabase* (p. 219)), the CSS-like "RGB(*r*,*g*,*b*)" syntax (case insensitive) and the HTML-like syntax (i.e. "#" followed by 6 hexadecimal digits for red, green, blue components).

Returns `true` if the conversion was successful, `false` otherwise.

This function is new since wxWidgets version 2.7.0

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

Assignment operator, taking another colour object.

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

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

**See also**

*wxColourDatabase* (p. 219)

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

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

alpha values.

### **wxColour::operator !=**

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

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

## **wxColourData**

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

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/cmndata.h>

### **See also**

*wxColour* (p. 214), *wxColourDialog* (p. 221), *wxColourDialog overview* (p. 2128)

### **wxColourData::wxColourData**

**wxColourData()**

Constructor. Initializes the custom colours to `wxNullColour`, the *data colour* setting to black, and the *choose full* setting to true.

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

**~wxColourData()**

Destructor.

### **wxColourData::GetChooseFull**

**bool GetChooseFull() const**

Under Windows, determines whether the Windows colour dialog will display the full dialog with custom colour selection controls. Under PalmOS, determines whether colour dialog will display full rgb colour picker or only available palette indexer. Has no meaning under other platforms.

The default value is true.

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

Gets the current colour associated with the colour dialog.

The default colour is black.

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

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

The default custom colours are invalid colours.

**wxColourData::SetChooseFull****void SetChooseFull(const bool *flag*)**

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

The default value is true.

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

Sets the default colour for the colour dialog.

The default colour is black.

**wxColourData::SetCustomColour****void SetCustomColour(int *i*, const wxColour& *colour*)**

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

The default custom colours are invalid colours.

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

Assignment operator for the colour data.

**wxColourDatabase**

`wxWidgets` 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 *AddColour* (p. 220) and may use it to look up colours by names using *Find* (p. 221) or find the names for the standard colour using *FindName* (p. 221).

There is one predefined instance of this class called **`wxTheColourDatabase`**.

**Derived from**

None

**Include files**

<wx/gdicmn.h>

**Remarks**

The standard database contains at least the following colours:

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

**See also**

*wxColour* (p. 214)

**`wxColourDatabase::wxColourDatabase`**

**`wxColourDatabase()`**

Constructs the colour database. It will be initialized at the first use.

**`wxColourDatabase::AddColour`**

**`void AddColour(const wxString& colourName, const wxColour& colour)`**

**`void AddColour(const wxString& colourName, wxColour* colour)`**

Adds a colour to the database. If a colour with the same name already exists, it is replaced.

Please note that the overload taking a pointer is deprecated and will be removed in the next wxWidgets version, please don't use it.

### **wxColourDatabase::Find**

**wxColour Find(const wxString& colourName)**

Finds a colour given the name. Returns an invalid colour object (that is, such that its *Ok()* (p. 217) method returns *false*) if the colour wasn't found in the database.

### **wxColourDatabase::FindName**

**wxString FindName(const wxColour& colour) const**

Finds a colour name given the colour. Returns an empty string if the colour is not found in the database.

## **wxColourDialog**

This class represents the colour chooser dialog.

### **Derived from**

*wxDialog* (p. 496)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### **Include files**

<wx/colordlg.h>

### **See also**

*wxColourDialog Overview* (p. 2128),  
*wxColour* (p. 214),  
*wxColourData* (p. 218),  
*wxGetColourFromUser* (p. 1937)

### **wxColourDialog::wxColourDialog**

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

Constructor. Pass a parent window, and optionally a pointer to a block of colour data, which will be copied to the colour dialog's colour data. Custom colours from colour data object will be used in dialog's colour palette. Invalid entries in custom colours list will be ignored on some platforms (GTK) or replaced with white colour on platforms where custom colours palette has fixed size (MSW).

**See also**

*wxColourData* (p. 218)

**wxColourDialog::~wxColourDialog**

**~wxColourDialog()**

Destructor.

**wxColourDialog::Create**

**bool Create**(*wxWindow\** parent, *wxColourData\** data = *NULL*)

Same as *constructor* (p. 221).

**wxColourDialog::GetColourData**

**wxColourData& GetColourData()**

Returns the *colour data* (p. 218) associated with the colour dialog.

**wxColourDialog::ShowModal**

**int ShowModal()**

Shows the dialog, returning *wxID\_OK* if the user pressed OK, and *wxID\_CANCEL* otherwise.

**wxColourPickerCtrl**

This control allows the user to select a colour. The generic implementation is a button which brings up a *wxColourDialog* (p. 221) when clicked. Native implementation may differ but this is usually a (small) widget which give access to the colour-chooser dialog. It is only available if *wxUSE\_COLOURPICKERCTRL* is set to 1 (the default).

**Derived from**

*wxPickerBase* (p. 1183)

*wxControl* (p. 285)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

**Include files**

<wx/clrpicker.h>

**Window styles**

**wxCLRP\_DEFAULT\_STYLE**The default style: 0.

- wxCLRP\_USE\_TEXTCTRL** Creates a text control to the left of the picker button which is completely managed by the *wxColourPickerCtrl* (p. 222) and which can be used by the user to specify a colour (see *SetColour* (p. 224)). The text control is automatically synchronized with button's value. Use functions defined in *wxPickerBase* (p. 1183) to modify the text control.
- wxCLRP\_SHOW\_LABEL** Shows the colour in HTML form (AABBCC) as colour button label (instead of no label at all).

### Event handling

To process a colour picker event, use these event handler macros to direct input to member functions that take a *wxColourPickerEvent* (p. 224) argument.

- EVT\_COLOURPICKER\_CHANGED(id, func)** The user changed the colour selected in the control either using the button or using text control (see *wxCLRP\_USE\_TEXTCTRL*; note that in this case the event is fired only if the user's input is valid, i.e. recognizable).

### See also

*wxColourDialog* (p. 221),  
*wxColourPickerEvent* (p. 224)

## **wxColourPickerCtrl::wxColourPickerCtrl**

**wxColourPickerCtrl**(*wxWindow \*parent*, *wxWindowID id*, *const wxColour& colour = \*wxBLACK*, *const wxPoint& pos = wxDefaultPosition*, *const wxSize& size = wxDefaultSize*, *long style = wxCLRP\_DEFAULT\_STYLE*, *const wxValidator& validator = wxDefaultValidator*, *const wxString& name = "colourpickerctrl"*)

Initializes the object and calls *Create* (p. 223) with all the parameters.

### **wxColourPickerCtrl::Create**

**bool Create**(*wxWindow \*parent*, *wxWindowID id*, *const wxColour& colour = \*wxBLACK*, *const wxPoint& pos = wxDefaultPosition*, *const wxSize& size = wxDefaultSize*, *long style = wxCLRP\_DEFAULT\_STYLE*, *const wxValidator& validator = wxDefaultValidator*, *const wxString& name = "colourpickerctrl"*)

### Parameters

*parent*

Parent window, must not be non-NULL.

*id*

The identifier for the control.

*colour*

The initial colour shown in the control.

*pos*

Initial position.

*size*

Initial size.

*style*

The window style, see `wxCRLP_*` flags.

*validator*

Validator which can be used for additional date checks.

*name*

Control name.

### Return value

`true` if the control was successfully created or `false` if creation failed.

### **`wxColourPickerCtrl::GetColour`**

**`wxColour GetColour() const`**

Returns the currently selected colour.

### **`wxColourPickerCtrl::SetColour`**

**`void SetColour(const wxColour &col)`**

**`void SetColour(const wxString &colname)`**

Sets the currently selected colour. See `wxColour::Set` (p. 217).

## **`wxColourPickerEvent`**

This event class is used for the events generated by `wxColourPickerCtrl` (p. 222).

### Derived from

`wxCommandEvent` (p. 250)

`wxEvent` (p. 572)

`wxObject` (p. 1148)



**Include files**

<wx/clrpicker.h>

**Event handling**

To process input from a `wxColourPickerCtrl`, use one of these event handler macros to direct input to member function that take *awxColourPickerEvent* (p. 224) argument:

**EVT\_COLOURPICKER\_CHANGED(id, func)**      Generated whenever the selected colour changes.

**See also**

*wxColourPickerCtrl* (p. 222)

**wxColourPickerEvent::wxColourPickerEvent**

**wxColourPickerEvent**(wxObject \* generator, int id, const wxColour& colour)

The constructor is not normally used by the user code.

**wxColourPickerEvent::GetColour**

**wxColour GetColour()** const

Retrieve the colour the user has just selected.

**wxColourPickerEvent::SetColour**

**void SetColour**(const wxColour &pos)

Set the colour associated with the event.

**wxComboBox**

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

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

If you need a customized combobox, have a look at *wxComboCtrl* (p. 232), *wxOwnerDrawnComboBox* (p. 1152), *wxComboPopup* (p. 246) and the ready-to-use *wxBitmapComboBox* (p. 135).

**Derived from**

*wxControlWithItems* (p. 286)

*wxControl* (p. 285)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### Include files

<wx/combobox.h>

### Window styles

#### **wxCB\_SIMPLE**

Creates a combobox with a permanently displayed list. Windows only.

#### **wxCB\_DROPDOWN**

Creates a combobox with a drop-down list.

#### **wxCB\_READONLY**

Same as **wxCB\_DROPDOWN** but only the strings specified as the combobox choices can be selected, it is impossible to select (even from a program) a string which is not in the choices list.

#### **wxCB\_SORT**

Sorts the entries in the list alphabetically.

#### **wxTE\_PROCESS\_ENTER**

The control will generate the event **wxEVT\_COMMAND\_TEXT\_ENTER** (otherwise pressing Enter key is either processed internally by the control or used for navigation between dialog controls). Windows only.

See also *window styles overview* (p. 2089).

### Event handling

#### **EVT\_COMBOBOX(id, func)**

Process a **wxEVT\_COMMAND\_COMBOBOX\_SELECTED** event, when an item on the list is selected. Note that calling *GetValue* (p. 230) returns the new value of selection.

#### **EVT\_TEXT(id, func)**

Process a **wxEVT\_COMMAND\_TEXT\_UPDATED** event, when the combobox text changes.

#### **EVT\_TEXT\_ENTER(id, func)**

Process a **wxEVT\_COMMAND\_TEXT\_ENTER** event, when <RETURN> is pressed in the combobox.

### See also

*wxListBox* (p. 974), *wxTextCtrl* (p. 1633), *wxChoice* (p. 186), *wxCommandEvent* (p. 250)

### **wxComboBox::wxComboBox**

**wxComboBox()**

Default constructor.

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

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

Constructor, creating and showing a combobox.

**Parameters**

*parent*

Parent window. Must not be NULL.

*id*

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

*value*

Initial selection string. An empty string indicates no selection.

*pos*

Window position.

*size*

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

*n*

Number of strings with which to initialise the control.

*choices*

An array of strings with which to initialise the control.

*style*

Window style. See *wxComboBox* (p. 225).

*validator*

Window validator.

*name*

Window name.

### See also

`wxComboBox::Create` (p. 228), `wxValidator` (p. 1767)

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

**wxPerl note:** In `wxPerl` there is just an array reference in place of `nand choices`.

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

**`~wxComboBox()`**

Destructor, destroying the combobox.

### **wxComboBox::Create**

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

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

Creates the combobox for two-step construction. Derived classes should call or replace this function. See `wxComboBox::wxComboBox` (p. 226) for further details.

### **wxComboBox::CanCopy**

**`bool CanCopy() const`**

Returns true if the combobox is editable and there is a text selection to copy to the clipboard. Only available on Windows.

### **wxComboBox::CanCut**

**`bool CanCut() const`**

Returns true if the combobox is editable and there is a text selection to copy to the clipboard. Only available on Windows.

### **wxComboBox::CanPaste**

**`bool CanPaste() const`**

Returns true if the combobox is editable and there is text on the clipboard that can be

pasted into the text field. Only available on Windows.

### **wxComboBox::CanRedo**

**bool CanRedo() const**

Returns true if the combobox is editable and the last undo can be redone. Only available on Windows.

### **wxComboBox::CanUndo**

**bool CanUndo() const**

Returns true if the combobox is editable and the last edit can be undone. Only available on Windows.

### **wxComboBox::Copy**

**void Copy()**

Copies the selected text to the clipboard.

### **wxComboBox::Cut**

**void Cut()**

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

### **wxComboBox::GetCurrentSelection**

**int GetCurrentSelection() const**

This function does the same things as *wxChoice::GetCurrentSelection* (p. 188) and returns the item currently selected in the dropdown list if it's open or the same thing as *GetSelection* (p. 289) otherwise.

### **wxComboBox::GetInsertionPoint**

**long GetInsertionPoint() const**

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

**Note:** Under wxMSW, this function always returns 0 if the combobox doesn't have the focus.

### **wxComboBox::GetLastPosition**

**virtual wxTextPos GetLastPosition() const**

Returns the last position in the combobox text field.

**wxComboBox::GetSelection****void GetSelection(long *from*, long *to*) const**

This is the same as *wxTextCtrl::GetSelection* (p. 1643) for the text control which is part of the combobox. Notice that this is a different method from *wxControlWithItems::GetSelection* (p. 289).

Currently this method is only implemented in wxMSW and wxGTK.

**wxComboBox::GetValue****wxString GetValue() const**

Returns the current value in the combobox text field.

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

Pastes text from the clipboard to the text field.

**wxComboBox::Redo****void Redo()**

Redoes the last undo in the text field. Windows only.

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

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

**Parameters***from*

The first position.

*to*

The second position.

*text*

The text to insert.

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

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

**Parameters**

*from*

The first position.

*to*

The last position.

**wxComboBox::SetInsertionPoint**

**void SetInsertionPoint(long pos)**

Sets the insertion point in the combobox text field.

**Parameters**

*pos*

The new insertion point.

**wxComboBox::SetInsertionPointEnd**

**void SetInsertionPointEnd()**

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

**wxComboBox::SetSelection**

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

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

**Parameters**

*from*

The first position.

*to*

The second position.

**wxPython note:** This method is called `SetMark` in wxPython, `SetSelectionname` is kept for `wxControlWithItems::SetSelection` (p. 292).

**wxComboBox::SetValue**

**void SetValue(const wxString& text)**

Sets the text for the combobox text field.

**NB:** For a combobox with `wxCB_READONLY` style the string must be in the combobox choices list, otherwise the call to `SetValue()` is ignored.

### Parameters

*text*

The text to set.

### **wxComboBox::Undo**

**void Undo()**

Undoes the last edit in the text field. Windows only.

## **wxComboCtrl**

A combo control is a generic combobox that allows totally custom popup. In addition it has other customization features. For instance, position and size of the dropdown button can be changed.

### **Setting Custom Popup for wxComboCtrl**

`wxComboCtrl` needs to be told somehow which control to use and this is done by `SetPopupControl()`. However, we need something more than just a `wxControl` in this method as, for example, we need to call `SetStringValue("initial text value")` and `wxControl` doesn't have such method. So we also need *awxComboPopup* (p. 246) which is an interface which must be implemented by a control to be usable as a popup.

We couldn't derive `wxComboPopup` from `wxControl` as this would make it impossible to have a class deriving from a `wxWidgets` control and from it, so instead it is just a mix-in.

Here's a minimal sample of *wxListView* (p. 1008) popup:

```
#include <wx/combo.h>
#include <wx/listctrl.h>

class wxListViewComboPopup : public wxListView,
                             public wxComboPopup
{
public:

    // Initialize member variables
    virtual void Init()
    {
        m_value = -1;
    }

    // Create popup control
    virtual bool Create(wxWindow* parent)
    {
```



```
        return
wxListView::Create(parent,1,wxPoint(0,0),wxDefaultSize);
    }

    // Return pointer to the created control
    virtual wxWindow *GetControl() { return this; }

    // Translate string into a list selection
    virtual void SetStringValue(const wxString& s)
    {
        int n = wxListView::FindItem(-1,s);
        if ( n >= 0 && n < wxListView::GetItemCount() )
            wxListView::Select(n);
    }

    // Get list selection as a string
    virtual wxString GetStringValue() const
    {
        if ( m_value >= 0 )
            return wxListView::GetItemText(m_value);
        return wxString();
    }

    // Do mouse hot-tracking (which is typical in list popups)
    void OnMouseMove(wxMouseEvent& event)
    {
        // TODO: Move selection to cursor
    }

    // On mouse left up, set the value and close the popup
    void OnMouseClicked(wxMouseEvent& WXUNUSED(event))
    {
        m_value = wxListView::GetFirstSelected();

        // TODO: Send event as well

        Dismiss();
    }

protected:

    int          m_value; // current item index

private:
    DECLARE_EVENT_TABLE()
};

BEGIN_EVENT_TABLE(wxListViewComboPopup, wxListView)
    EVT_MOTION(wxListViewComboPopup::OnMouseMove)
    EVT_LEFT_UP(wxListViewComboPopup::OnMouseClicked)
END_EVENT_TABLE()
```

Here's how you would create and populate it in a dialog constructor:

```
    wxComboCtrl* comboCtrl = new
wxComboCtrl(this,wxID_ANY,wxEmptyString);
```

```
wxListViewComboPopup* popupCtrl = new wxListViewComboPopup();

comboCtrl->SetPopupControl(popupCtrl);

// Populate using wxListView methods
popupCtrl->InsertItem(popupCtrl->GetItemCount(), wxT("First
Item"));
popupCtrl->InsertItem(popupCtrl->GetItemCount(), wxT("Second
Item"));
popupCtrl->InsertItem(popupCtrl->GetItemCount(), wxT("Third
Item"));
```

**Derived from**

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<combo.h>

**Window styles****wxCB\_READONLY**

Text will not be editable.

**wxCB\_SORT**

Sorts the entries in the list alphabetically.

**wxTE\_PROCESS\_ENTER**

The control will generate the event `wxEVT_COMMAND_TEXT_ENTER` (otherwise pressing Enter key is either processed internally by the control or used for navigation between dialog controls). Windows only.

**wxCC\_SPECIAL\_DCLICK**

Double-clicking triggers a call to `popup's OnComboDoubleClick`. Actual behaviour is defined by a derived class. For instance, `wxOwnerDrawnComboBox` will cycle an item. This style only applies if `wxCB_READONLY` is used as well.

**wxCC\_STD\_BUTTON**

Drop button will behave more like a standard push button.

See also *window styles overview* (p. 2089).

**Event handling****EVT\_TEXT(id, func)**

Process a `wxEVT_COMMAND_TEXT_UPDATED` event, when the text changes.

**EVT\_TEXT\_ENTER(id, func)**

Process a wxEVT\_COMMAND\_TEXT\_ENTER event, when <RETURN> is pressed in the combo control.

**See also**

*wxComboBox* (p. 225), *wxChoice* (p. 186), *wxOwnerDrawnComboBox* (p. 1152), *wxComboPopup* (p. 246), *wxCommandEvent* (p. 250)

**wxComboCtrl::wxComboCtrl****wxComboCtrl()**

Default constructor.

**wxComboCtrl**(wxWindow\* *parent*, wxWindowID *id*, const wxString& *value* = "", const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size* = wxDefaultSize, long *style* = 0, const wxValidator& *validator* = wxDefaultValidator, const wxString& *name* = "comboCtrl")

Constructor, creating and showing a combo control.

**Parameters***parent*

Parent window. Must not be NULL.

*id*

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

*value*

Initial selection string. An empty string indicates no selection.

*pos*

Window position.

*size*

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

*style*

Window style. See *wxComboCtrl* (p. 232).

*validator*

Window validator.

*name*

Window name.

### See also

*wxComboCtrl::Create* (p. 236), *wxValidator* (p. 1767)

### **wxComboCtrl::~~wxComboCtrl**

**~wxComboCtrl()**

Destructor, destroying the combo control.

### **wxComboCtrl::AnimateShow**

**virtual bool AnimateShow(const wxRect& rect, int flags)**

This member function is not normally called in application code. Instead, it can be implemented in a derived class to create a custom popup animation.

### Parameters

Same as in *DoShowPopup* (p. 237).

### Return value

*true* if animation finishes before the function returns. *false* otherwise. In the latter case you need to manually call *DoShowPopup* after the animation ends.

### **wxComboCtrl::Create**

**bool Create(wxWindow\* parent, wxWindowID id, const wxString& value = "", const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0, const wxValidator& validator = wxDefaultValidator, const wxString& name = "comboCtrl")**

Creates the combo control for two-step construction. Derived classes should call or replace this function. See *wxComboCtrl::wxComboCtrl* (p. 235) for further details.

### **wxComboCtrl::Copy**

**void Copy()**

Copies the selected text to the clipboard.

### **wxComboCtrl::Cut**

**void Cut()**

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

**wxComboCtrl::DoSetPopupControl****void DoSetPopupControl(wxComboPopup\* popup)**

This member function is not normally called in application code. Instead, it can be implemented in a derived class to return default wxComboPopup, in case popup is NULL.

**Note:** If you have implemented OnButtonClick to do something else than show the popup, then DoSetPopupControl must always return NULL.

**wxComboCtrl::DoShowPopup****virtual void DoShowPopup(const wxRect& rect, int flags)**

This member function is not normally called in application code. Instead, it must be called in a derived class to make sure popup is properly shown after a popup animation has finished (but only if *AnimateShow* (p. 236) did not finish the animation within its function scope).

**Parameters***rect*

Position to show the popup window at, in screen coordinates.

*flags*

Combination of any of the following:

*wxComboCtrl::ShowAbove*

Popup is shown above the control instead of below.

*wxComboCtrl::CanDeferShow*

Showing the popup can be deferred to happen sometime after *ShowPopup* (p. 245) has finished. In this case, *AnimateShow* (p. 236) must return false.

**wxComboCtrl::EnablePopupAnimation****void EnablePopupAnimation(bool enable = true)**

Enables or disables popup animation, if any, depending on the value of the argument.

**wxComboCtrl::GetBitmapDisabled****const wxBitmap& GetBitmapDisabled() const**

Returns disabled button bitmap that has been set with *SetButtonBitmaps* (p. 242).

**Return value**

A reference to the disabled state bitmap.

**wxComboCtrl::GetBitmapHover****const wxBitmap& GetBitmapHover() const**

Returns button mouse hover bitmap that has been set with *SetButtonBitmaps* (p. 242).

**Return value**

A reference to the mouse hover state bitmap.

**wxComboCtrl::GetBitmapNormal****const wxBitmap& GetBitmapNormal() const**

Returns default button bitmap that has been set with *SetButtonBitmaps* (p. 242).

**Return value**

A reference to the normal state bitmap.

**wxComboCtrl::GetBitmapPressed****const wxBitmap& GetBitmapPressed() const**

Returns depressed button bitmap that has been set with *SetButtonBitmaps* (p. 242).

**Return value**

A reference to the depressed state bitmap.

**wxComboCtrl::GetButtonSize****wxSize GetButtonSize()**

Returns current size of the dropdown button.

**wxComboCtrl::GetCustomPaintWidth****int GetCustomPaintWidth() const**

Returns custom painted area in control.

**See also**

*wxComboCtrl::SetCustomPaintWidth* (p. 243).

**wxComboCtrl::GetFeatures****static int GetFeatures()**

Returns features supported by `wxComboCtrl`. If needed feature is missing, you need to instead use `wxGenericComboCtrl`, which however may lack native look and feel (but otherwise sports identical API).

### Return value

Value returned is a combination of following flags:

`wxComboCtrlFeatures::MovableButton` Button can be on either side of the control.

`wxComboCtrlFeatures::BitmapButton` Button may be replaced with bitmap.

`wxComboCtrlFeatures::ButtonSpacing` Button can have spacing.

`wxComboCtrlFeatures::TextIndent` `SetTextIndent` works.

`wxComboCtrlFeatures::PaintControl` Combo control itself can be custom painted.

`wxComboCtrlFeatures::PaintWritable` A variable- width area in front of writable combo control's textctrl can be custom painted.

`wxComboCtrlFeatures::Borderless` `wxNO_BORDER` window style works.

`wxComboCtrlFeatures::All` All of the above.

### `wxComboCtrl::GetInsertionPoint`

#### `long GetInsertionPoint() const`

Returns the insertion point for the combo control's text field.

**Note:** Under `wxMSW`, this function always returns 0 if the combo control doesn't have the focus.

### `wxComboCtrl::IsPopupWindowState`

#### `bool IsPopupWindowState(int state) const`

Returns `true` if the popup window is in the given state. Possible values

are: `wxComboCtrl::Hidden` Popup window is hidden.

`wxComboCtrl::Animating` Popup window is being shown, but the popup animation has not yet finished.

`wxComboCtrl::Visible` Popup window is fully visible.

### `wxComboCtrl::GetLastPosition`

#### `long GetLastPosition() const`

Returns the last position in the combo control text field.

**wxComboCtrl::GetPopupControl****wxComboPopup\* GetPopupControl()**

Returns current popup interface that has been set with SetPopupControl.

**wxComboCtrl::GetPopupWindow****wxWindow\* GetPopupWindow() const**

Returns popup window containing the popup control.

**wxComboCtrl::GetTextCtrl****wxTextCtrl\* GetTextCtrl() const**

Get the text control which is part of the combo control.

**wxComboCtrl::GetTextIndent****wxCoord GetTextIndent() const**

Returns actual indentation in pixels.

**wxComboCtrl::GetTextRect****const wxRect& GetTextRect() const**

Returns area covered by the text field (includes everything except borders and the dropdown button).

**wxComboCtrl::GetValue****wxString GetValue() const**

Returns text representation of the current value. For writable combo control it always returns the value in the text field.

**wxComboCtrl::HidePopup****void HidePopup()**

Dismisses the popup window.

**wxComboCtrl::IsPopupShown****bool IsPopupShown() const**



Returns `true` if the popup is currently shown

### **wxComboCtrl::OnButtonClick**

#### **void OnButtonClick()**

Implement in a derived class to define what happens on dropdown button click.

Default action is to show the popup.

**Note:** If you implement this to do something else than show the popup, you must then also implement *DoSetPopupControl* (p. 237) to always return `NULL`.

### **wxComboCtrl::Paste**

#### **void Paste()**

Pastes text from the clipboard to the text field.

### **wxComboCtrl::Remove**

#### **void Remove(long from, long to)**

Removes the text between the two positions in the combo control text field.

#### **Parameters**

*from*

The first position.

*to*

The last position.

### **wxComboCtrl::Replace**

#### **void Replace(long from, long to, const wxString& value)**

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

#### **Parameters**

*from*

The first position.

*to*

The second position.

*text*

The text to insert.

### **wxComboCtrl::SetButtonBitmaps**

```
void SetButtonBitmaps(const wxBitmap& bmpNormal, bool pushButtonBg = false,  
const wxBitmap& bmpPressed = wxNullBitmap, const wxBitmap& bmpHover =  
wxNullBitmap, const wxBitmap& bmpDisabled = wxNullBitmap)
```

Sets custom dropdown button graphics.

#### **Parameters**

*bmpNormal*

Default button image.

*pushButtonBg*

If `true`, blank push button background is painted below the image.

*bmpPressed*

Depressed button image.

*bmpHover*

Button image when mouse hovers above it. This should be ignored on platforms and themes that do not generally draw different kind of button on mouse hover.

*bmpDisabled*

Disabled button image.

### **wxComboCtrl::SetButtonPosition**

```
void SetButtonPosition(int width = -1, int height = -1, int side = wxRIGHT, int spacingX  
= 0)
```

Sets size and position of dropdown button.

#### **Parameters**

*width*

Button width. Value  $\leq 0$  specifies default.

*height*

Button height. Value  $\leq 0$  specifies default.

*side*

Indicates which side the button will be placed. Value can be `wxLEFT` or `wxRIGHT`.

*spacingX*

Horizontal spacing around the button. Default is 0.

### **wxComboCtrl::SetCustomPaintWidth**

**void SetCustomPaintWidth(int width)**

Set width, in pixels, of custom painted area in control without `wxCB_READONLY` style. In read-only *wxOwnerDrawnComboBox* (p. 1152), this is used to indicate area that is not covered by the focus rectangle.

### **wxComboCtrl::SetInsertionPoint**

**void SetInsertionPoint(long pos)**

Sets the insertion point in the text field.

#### **Parameters**

*pos*

The new insertion point.

### **wxComboCtrl::SetInsertionPointEnd**

**void SetInsertionPointEnd()**

Sets the insertion point at the end of the combo control text field.

### **wxComboCtrl::SetPopupAnchor**

**void SetPopupAnchor(int anchorSide)**

Set side of the control to which the popup will align itself. Valid values are `wxLEFT`, `wxRIGHT` and 0. The default value 0 means that the most appropriate side is used (which, currently, is always `wxLEFT`).

### **wxComboCtrl::SetPopupControl**

**void SetPopupControl(wxComboPopup\* popup)**

Set popup interface class derived from `wxComboPopup`. This method should be called as soon as possible after the control has been created, unless *OnButtonClick* (p. 241) has been overridden.

### **wxComboCtrl::SetPopupExtents**

**void SetPopupExtents(int extLeft, int extRight)**

Extends popup size horizontally, relative to the edges of the combo control.

**Parameters***extLeft*

How many pixel to extend beyond the left edge of the control. Default is 0.

*extRight*

How many pixel to extend beyond the right edge of the control. Default is 0.

**Remarks**

Popup minimum width may override arguments.

It is up to the popup to fully take this into account.

**wxComboCtrl::SetPopupMaxHeight****void SetPopupMaxHeight(int *height*)**

Sets preferred maximum height of the popup.

**Remarks**

Value -1 indicates the default.

Also, popup implementation may choose to ignore this.

**wxComboCtrl::SetPopupMinWidth****void SetPopupMinWidth(int *width*)**

Sets minimum width of the popup. If wider than combo control, it will extend to the left.

**Remarks**

Value -1 indicates the default.

Also, popup implementation may choose to ignore this.

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

Selects the text between the two positions, in the combo control text field.

**Parameters***from*

The first position.

*to*

The second position.

### **wxComboCtrl::SetText**

**void SetText(const wxString& value)**

Sets the text for the text field without affecting the popup. Thus, unlike *SetValue* (p. 245), it works equally well with combo control using `wxCB_READONLY` style.

### **wxComboCtrl::SetTextIndent**

**void SetTextIndent(int indent)**

This will set the space in pixels between left edge of the control and the text, regardless whether control is read-only or not. Value -1 can be given to indicate platform default.

### **wxComboCtrl::SetValue**

**void SetValue(const wxString& value)**

Sets the text for the combo control text field.

**NB:** For a combo control with `wxCB_READONLY` style the string must be accepted by the popup (for instance, exist in the dropdown list), otherwise the call to `SetValue()` is ignored

### **wxComboCtrl::SetValueWithEvent**

**void SetValueWithEvent(const wxString& value, bool withEvent = true)**

Same as `SetValue`, but also sends `wxCommandEvent` of type `wxEVT_COMMAND_TEXT_UPDATED` if `withEvent` is `true`.

### **wxComboCtrl::ShowPopup**

**void ShowPopup()**

Show the popup.

### **wxComboCtrl::Undo**

**void Undo()**

Undoes the last edit in the text field. Windows only.

### **wxComboCtrl::UseAltPopupWindow**

**void UseAltPopupWindow(bool enable = true)**

Enable or disable usage of an alternative popup window, which guarantees ability to focus the popup control, and allows common native controls to function normally. This

alternative popup window is usually a `wxDialog`, and as such, when it is shown, its parent top-level window will appear as if the focus has been lost from it.

## **wxComboPopup**

In order to use a custom popup with `wxComboCtrl` (p. 232), an interface class must be derived from `wxComboPopup`. For more information how to use it, see *Setting Custom Popup for wxComboCtrl* (p. 232).

### **Include files**

`<combo.h>`

### **See also**

`wxComboCtrl` (p. 232)

## **wxComboPopup::wxComboPopup**

### **wxComboPopup()**

Default constructor. It is recommended that internal variables are prepared in *Init* (p. 247) instead (because `m_combo` (p. 246) is not valid in constructor).

## **wxComboPopup::m\_combo**

### **wxComboCtrl m\_combo**

Parent `wxComboCtrl` (p. 232). This is parameter has been prepared before *Init* (p. 247) is called.

## **wxComboPopup::Create**

### **bool Create(wxWindow\* parent)**

The derived class must implement this to create the popup control.

### **Return value**

`true` if the call succeeded, `false` otherwise.

## **wxComboPopup::Dismiss**

### **void Dismiss()**

Utility function that hides the popup.

## **wxComboPopup::GetAdjustedSize**

**wxSize GetAdjustedSize(int minWidth, int prefHeight, int maxHeight)**

The derived class may implement this to return adjusted size for the popup control, according to the variables given.

**Parameters**

*minWidth*

Preferred minimum width.

*prefHeight*

Preferred height. May be -1 to indicate no preference.

*maxWidth*

Max height for window, as limited by screen size.

**Remarks**

Called each time popup is about to be shown.

**wxComboPopup::GetControl****wxWindow\* GetControl()**

The derived class must implement this to return pointer to the associated control created in *Create* (p. 246).

**wxComboPopup::GetStringValue****wxString GetStringValue() const**

The derived class must implement this to return string representation of the value.

**wxComboPopup::Init****void Init()**

The derived class must implement this to initialize its internal variables. This method is called immediately after construction finishes. *m\_combo* (p. 246) member variable has been initialized before the call.

**wxComboPopup::IsCreated****bool IsCreated() const**

Utility method that returns `true` if *Create* has been called.

Useful in conjunction with *LazyCreate* (p. 248).

**wxComboPopup::LazyCreate****bool LazyCreate()**

The derived class may implement this to return `true` if it wants to delay call to *Create* (p. 246) until the popup is shown for the first time. It is more efficient, but on the other hand it is often more convenient to have the control created immediately.

**Remarks**

Base implementation returns `false`.

**wxComboPopup::OnComboDoubleClick****void OnComboDoubleClick()**

The derived class may implement this to do something when the parent *wxComboCtrl* (p. 232) gets double-clicked.

**wxComboPopup::OnComboKeyEvent****void OnComboKeyEvent(wxKeyEvent& event)**

The derived class may implement this to receive key events from the parent *wxComboCtrl* (p. 232).

Events not handled should be skipped, as usual.

**wxComboPopup::OnDismiss****void OnDismiss()**

The derived class may implement this to do special processing when popup is hidden.

**wxComboPopup::OnPopup****void OnPopup()**

The derived class may implement this to do special processing when popup is shown.

**wxComboPopup::PaintComboControl****void PaintComboControl(wxDC& dc, const wxRect& rect)**

The derived class may implement this to paint the parent *wxComboCtrl* (p. 232).

Default implementation draws value as string.

**wxComboPopup::SetStringValue****void SetStringValue(const wxString& value)**



The derived class must implement this to receive string value changes from *wxComboCtrl* (p. 232).

## **wxCommand**

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

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/cmdproc.h>

### **See also**

*Overview* (p. 2135)

## **wxCommand::wxCommand**

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

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

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

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

## **wxCommand::~~wxCommand**

**~wxCommand**()

Destructor.

## **wxCommand::CanUndo**

**bool CanUndo**()

Returns true if the command can be undone, false otherwise.

## **wxCommand::Do**

**bool Do()**

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

**wxCommand::GetName****wxString GetName()**

Returns the command name.

**wxCommand::Undo****bool Undo()**

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

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

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

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

**wxCommandEvent**

This event class contains information about command events, which originate from a variety of simple controls. More complex controls, such as *wxTreeCtrl* (p. 1728), have separate command event classes.

**Derived from**

*wxEvent* (p. 572)

**Include files**

<wx/event.h>

**Event table macros**

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

<b>EVT_COMMAND(id, event, func)</b>	Process a command, supplying the window identifier, command event identifier, and member function.
<b>EVT_COMMAND_RANGE(id1, id2, event, func)</b>	Process a command for a range of window identifiers, supplying the minimum and maximum window identifiers, command event identifier, and member function.
<b>EVT_BUTTON(id, func)</b>	Process a <code>wxEVT_COMMAND_BUTTON_CLICKED</code> command, which is generated by a <code>wxButton</code> control.
<b>EVT_CHECKBOX(id, func)</b>	Process a <code>wxEVT_COMMAND_CHECKBOX_CLICKED</code> command, which is generated by a <code>wxCheckBox</code> control.
<b>EVT_CHOICE(id, func)</b>	Process a <code>wxEVT_COMMAND_CHOICE_SELECTED</code> command, which is generated by a <code>wxChoice</code> control.
<b>EVT_COMBOBOX(id, func)</b>	Process a <code>wxEVT_COMMAND_COMBOBOX_SELECTED</code> command, which is generated by a <code>wxComboBox</code> control.
<b>EVT_LISTBOX(id, func)</b>	Process a <code>wxEVT_COMMAND_LISTBOX_SELECTED</code> command, which is generated by a <code>wxListBox</code> control.
<b>EVT_LISTBOX_DCLICK(id, func)</b>	Process a <code>wxEVT_COMMAND_LISTBOX_DOUBLECLICKED</code> command, which is generated by a <code>wxListBox</code> control.
<b>EVT_MENU(id, func)</b>	Process a <code>wxEVT_COMMAND_MENU_SELECTED</code> command, which is generated by a menu item.
<b>EVT_MENU_RANGE(id1, id2, func)</b>	Process a <code>wxEVT_COMMAND_MENU_RANGE</code> command, which is generated by a range of menu items.
<b>EVT_CONTEXT_MENU(func)</b>	Process the event generated when the user has requested a popup menu to appear by pressing

	a special keyboard key (under Windows) or by right clicking the mouse.
<b>EVT_RADIOBOX(id, func)</b>	Process a <code>wxEVT_COMMAND_RADIOBOX_SELECTED</code> command, which is generated by a <code>wxRadioBox</code> control.
<b>EVT_RADIOBUTTON(id, func)</b>	Process a <code>wxEVT_COMMAND_RADIOBUTTON_SELECTED</code> command, which is generated by a <code>wxRadioButton</code> control.
<b>EVT_SCROLLBAR(id, func)</b>	Process a <code>wxEVT_COMMAND_SCROLLBAR_UPDATED</code> command, which is generated by a <code>wxScrollBar</code> control. This is provided for compatibility only; more specific scrollbar event macros should be used instead (see <i>wxScrollEvent</i> (p. 1423)).
<b>EVT_SLIDER(id, func)</b>	Process a <code>wxEVT_COMMAND_SLIDER_UPDATED</code> command, which is generated by a <code>wxSlider</code> control.
<b>EVT_TEXT(id, func)</b>	Process a <code>wxEVT_COMMAND_TEXT_UPDATED</code> command, which is generated by a <code>wxTextCtrl</code> control.
<b>EVT_TEXT_ENTER(id, func)</b>	Process a <code>wxEVT_COMMAND_TEXT_ENTER</code> command, which is generated by a <code>wxTextCtrl</code> control. Note that you must use <code>wxTE_PROCESS_ENTER</code> flag when creating the control if you want it to generate such events.
<b>EVT_TEXT_MAXLEN(id, func)</b>	Process a <code>wxEVT_COMMAND_TEXT_MAXLEN</code> command, which is generated by a <code>wxTextCtrl</code> control when the user tries to enter more characters into it than the limit previously set with <i>SetMaxLength</i> (p. 1649).
<b>EVT_TOGGLEBUTTON(id, func)</b>	Process a <code>wxEVT_COMMAND_TOGGLEBUTTON_CLICKED</code> event.
<b>EVT_TOOL(id, func)</b>	Process a <code>wxEVT_COMMAND_TOOL_CLICKED</code> event (a synonym for <code>wxEVT_COMMAND_MENU_SELECTED</code> ). Pass the id of the tool.

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

### **wxCommandEvent::wxCommandEvent**

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

Constructor.

**wxCommandEvent::Checked****bool Checked() const**

Deprecated, use *IsChecked* (p. 255) instead.

**wxCommandEvent::GetClientData****void\* GetClientData()**

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

**wxCommandEvent::GetClientObject****wxClientData \* GetClientObject()**

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

**wxCommandEvent::GetExtraLong****long GetExtraLong()**

Returns extra information dependant on the event objects type. 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::GetInt****int GetInt()**

Returns the 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::GetSelection****int GetSelection()**

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

**wxCommandEvent::GetString****wxString GetString()**

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

**wxCommandEvent::IsChecked****bool IsChecked() const**

This method can be used with checkbox and menu events: for the checkboxes, the method returns `true` for a selection event and `false` for a deselection one. For the menu events, this method indicates if the menu item just has become checked or unchecked (and thus only makes sense for checkable menu items).

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

For a listbox or similar event, returns `true` if it is a selection, `false` if it is a deselection.

**wxCommandEvent::SetClientData****void SetClientData(void\* *clientData*)**

Sets the client data for this event.

**wxCommandEvent::SetClientObject****void SetClientObject(wxClientData\* *clientObject*)**

Sets the client object for this event. The client object is *not* owned by the event object and the event object will not delete the client object in its destructor. The client object must be owned and deleted by another object (e.g. a control) that has longer life time than the event object.

**wxCommandEvent::SetExtraLong****void SetExtraLong(long *extraLong*)**

Sets the `m_extraLong` member.

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

Sets the `m_commandInt` member.

**wxCommandEvent::SetString****void SetString(const wxString& *string*)**

Sets the `m_commandString` member.

**wxCommandProcessor**

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

**Derived from**

`wxObject` (p. 1148)

**Include files**

`<wx/cmdproc.h>`

**See also**

*wxCommandProcessor overview* (p. 2136), *wxCommand* (p. 249)

**wxCommandProcessor::wxCommandProcessor**

`wxCommandProcessor(int maxCommands = -1)`

Constructor.

*maxCommands* may be set to a positive integer to limit the number of commands stored to it, otherwise (and by default) the list of commands can grow arbitrarily.

**wxCommandProcessor::~~wxCommandProcessor**

`~wxCommandProcessor()`

Destructor.

**wxCommandProcessor::CanUndo**

`virtual bool CanUndo()`

Returns true if the currently-active command can be undone, false otherwise.

**wxCommandProcessor::ClearCommands**

`virtual void ClearCommands()`

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

**wxCommandProcessor::Redo**

`virtual bool Redo()`

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



**wxCommandProcessor::GetCommands****wxList& GetCommands() const**

Returns the list of commands.

**wxCommandProcessor::GetMaxCommands****int GetMaxCommands() const**

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

**wxCommandProcessor::GetEditMenu****wxMenu\* GetEditMenu() const**

Returns the edit menu associated with the command processor.

**wxCommandProcessor::GetRedoAccelerator****const wxString& GetRedoAccelerator() const**

Returns the string that will be appended to the Redo menu item.

**wxCommandProcessor::GetRedoMenuLabel****wxString GetRedoMenuLabel() const**

Returns the string that will be shown for the redo menu item.

**wxCommandProcessor::GetUndoAccelerator****const wxString& GetUndoAccelerator() const**

Returns the string that will be appended to the Undo menu item.

**wxCommandProcessor::GetUndoMenuLabel****wxString GetUndoMenuLabel() const**

Returns the string that will be shown for the undo menu item.

**wxCommandProcessor::Initialize****virtual void Initialize()**

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

**wxCommandProcessor::IsDirty**

**virtual bool IsDirty()**

Returns a boolean value that indicates if changes have been made since the last save operation. This only works if *wxCommandProcessor::MarkAsSaved* (p. 258) is called whenever the project is saved.

**wxCommandProcessor::MarkAsSaved****virtual void MarkAsSaved()**

You must call this method whenever the project is saved if you plan to use *wxCommandProcessor::IsDirty* (p. 257).

**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::SetMenuStrings****void SetMenuStrings()**

Sets the menu labels according to the currently set menu and the current command state.

**wxCommandProcessor::SetRedoAccelerator****void SetRedoAccelerator(const wxString&accel)**

Sets the string that will be appended to the Redo menu item.

**wxCommandProcessor::SetUndoAccelerator****void SetUndoAccelerator(const wxString&accel)**

Sets the string that will be appended to the Undo menu item.

**wxCommandProcessor::Submit****virtual bool Submit(wxCommand \*command, bool storeIt = true)**

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

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

## **wxCommandProcessor::Undo**

### **virtual bool Undo()**

Undoes the command just executed.

## **wxCondition**

wxCondition variables correspond to pthread conditions or to Win32 event objects. They may be used in a multithreaded application to wait until the given condition becomes true which happens when the condition becomes signaled.

For example, if a worker thread is doing some long task and another thread has to wait until it is finished, the latter thread will wait on the condition object and the worker thread will signal it on exit (this example is not perfect because in this particular case it would be much better to just *Wait()* (p. 1678) for the worker thread, but if there are several worker threads it already makes much more sense).

Note that a call to *Signal()* (p. 261) may happen before the other thread calls *Wait()* (p. 262) and, just as with the pthread conditions, the signal is then lost and so if you want to be sure that you don't miss it you must keep the mutex associated with the condition initially locked and lock it again before calling *Signal()* (p. 261). Of course, this means that this call is going to block until *Wait()* (p. 262) is called by another thread.

### **Example**

This example shows how a main thread may launch a worker thread which starts running and then waits until the main thread signals it to continue:

```
class MySignallingThread : public wxThread
{
public:
    MySignallingThread(wxMutex *mutex, wxCondition *condition)
    {
        m_mutex = mutex;
        m_condition = condition;

        Create();
    }

    virtual ExitCode Entry()
    {
        ... do our job ...

        // tell the other(s) thread(s) that we're about to terminate:
we must    // lock the mutex first or we might signal the condition before
the        // waiting threads start waiting on it!
            wxMutexLocker lock(*m_mutex);
            m_condition->Broadcast(); // same as Signal() here -- one
waiter only

        return 0;
    }
};
```

```
    }

private:
    wxCondition *m_condition;
    wxMutex *m_mutex;
};

int main()
{
    wxMutex mutex;
    wxCondition condition(mutex);

    // the mutex should be initially locked
    mutex.Lock();

    // create and run the thread but notice that it won't be able to
    // exit (and signal its exit) before we unlock the mutex below
    MySignallingThread *thread = new MySignallingThread(&mutex,
&condition);

    thread->Run();

    // wait for the thread termination: Wait() atomically unlocks the
mutex
    // which allows the thread to continue and starts waiting
    condition.Wait();

    // now we can exit
    return 0;
}
```

Of course, here it would be much better to simply use a joinable thread and call *wxThread::Wait* (p. 1678) on it, but this example does illustrate the importance of properly locking the mutex when using *wxCondition*.

### Constants

The following return codes are returned by *wxCondition* member functions:

```
enum wxCondError
{
    wxCOND_NO_ERROR = 0,        // successful completion
    wxCOND_INVALID,            // object hasn't been initialized
successfully
    wxCOND_TIMEOUT,            // WaitTimeout() has timed out
    wxCOND_MISC_ERROR          // some other error
};
```

### Derived from

None.

### Include files

<wx/thread.h>

**See also**

*wxThread* (p. 1670), *wxMutex* (p. 1131)

**wxCondition::wxCondition**

**wxCondition(wxMutex& mutex)**

Default and only constructor. The *mutex* must be locked by the caller before calling *Wait* (p. 262) function.

Use *IsOk* (p. 261) to check if the object was successfully initialized.

**wxCondition::~~wxCondition**

**~wxCondition()**

Destroys the *wxCondition* object. The destructor is not virtual so this class should not be used polymorphically.

**wxCondition::Broadcast**

**void Broadcast()**

Broadcasts to all waiting threads, waking all of them up. Note that this method may be called whether the mutex associated with this condition is locked or not.

**See also**

*wxCondition::Signal* (p. 261)

**wxCondition::IsOk**

**bool IsOk() const**

Returns *true* if the object had been initialized successfully, *false* if an error occurred.

**wxCondition::Signal**

**void Signal()**

Signals the object waking up at most one thread. If several threads are waiting on the same condition, the exact thread which is woken up is undefined. If no threads are waiting, the signal is lost and the condition would have to be signalled again to wake up any thread which may start waiting on it later.

Note that this method may be called whether the mutex associated with this condition is locked or not.

**See also**

*wxCondition::Broadcast* (p. 261)

## **wxCondition::Wait**

### **wxCondError Wait()**

Waits until the condition is signalled.

This method atomically releases the lock on the mutex associated with this condition (this is why it must be locked prior to calling *Wait*) and puts the thread to sleep until *Signal* (p. 261) or *Broadcast* (p. 261) is called. It then locks the mutex again and returns.

Note that even if *Signal* (p. 261) had been called before *Wait* without waking up any thread, the thread would still wait for another one and so it is important to ensure that the condition will be signalled after *Wait* or the thread may sleep forever.

### **Return value**

Returns `wxCOND_NO_ERROR` on success, another value if an error occurred.

### **See also**

*WaitTimeout* (p. 262)

## **wxCondition::WaitTimeout**

### **wxCondError WaitTimeout(unsigned long milliseconds)**

Waits until the condition is signalled or the timeout has elapsed.

This method is identical to *Wait* (p. 262) except that it returns, with the return code of `wxCOND_TIMEOUT` as soon as the given timeout expires.

### **Parameters**

*milliseconds*

Timeout in milliseconds

### **Return value**

Returns `wxCOND_NO_ERROR` if the condition was signalled, `wxCOND_TIMEOUT` if the timeout elapsed before this happened or another error code from `wxCondError` enum.

## **wxConfigBase**

`wxConfigBase` class defines the basic interface of all config classes. It can not be used by itself (it is an abstract base class) and you will always use one of its derivations: *wxFileConfig* (p. 598), *wxRegConfig* or any other.

However, usually you don't even need to know the precise nature of the class you're working with but you would just use the `wxConfigBase` methods. This allows you to write

the same code regardless of whether you're working with the registry under Win32 or text-based config files under Unix (or even Windows 3.1 .INI files if you're really unlucky). To make writing the portable code even easier, wxWidgets provides a typedef `wxConfig` which is mapped onto the native `wxConfigBase` implementation on the given platform: i.e. `wxRegConfig` under Win32 and `wxFileConfig` otherwise.

See *config overview* (p. 2074) for the descriptions of all features of this class.

It is highly recommended to use static functions `Get()` and/or `Set()`, so please have a *look at them*. (p. 264)

### Derived from

No base class

### Include files

<wx/config.h> (to let wxWidgets choose a wxConfig class for your platform)

<wx/confbase.h> (base config class)

<wx/fileconf.h> (wxFileConfig class)

<wx/msw/regconf.h> (wxRegConfig class)

### Example

Here is how you would typically use this class:

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

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

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

// the changes will be written back automatically
delete config;
```

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

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

## Static functions

These functions deal with the "default" config object. Although its usage is not at all mandatory it may be convenient to use a global config object instead of creating and deleting the local config objects each time you need one (especially because creating a `wxFileConfig` object might be a time consuming operation). In this case, you may create this global config object in the very start of the program and `Set()` it as the default. Then, from anywhere in your program, you may access it using the `Get()` function. This global `wxConfig` object will be deleted by `wxWidgets` automatically if it exists. Note that this implies that if you do delete this object yourself (usually in `wxApp::OnExit` (p. 51)) you must use `Set(NULL)` to prevent `wxWidgets` from deleting it the second time.

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

**Note:** You should use either `Set()` or `Get()` because `wxWidgets` library itself would take advantage of it and could save various information in it. For example `wxFontMapper` (p. 674) or Unix version of `wxFileDialog` (p. 600) have the ability to use `wxConfig` class.

`Set` (p. 275)

`Get` (p. 271)

`Create` (p. 270)

`DontCreateOnDemand` (p. 270)

## Constructor and destructor

`wxConfigBase` (p. 268)

`~wxConfigBase` (p. 270)

## Path management

As explained in *config overview* (p. 2074), the config classes support a file system-like hierarchy of keys (files) and groups (directories). As in the file system case, to specify a key in the config class you must use a path to it. Config classes also support the notion of the current group, which makes it possible to use the relative paths. To clarify all this, here is an example (it is only for the sake of demonstration, it doesn't do anything sensible!):



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

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

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

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

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

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

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

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

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

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

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

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

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

    foo(config);

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

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

*SetPath* (p. 275)

*GetPath* (p. 273)

## Enumeration

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

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

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

```
wxConfigBase *config = ...;
wxArrayString aNames;

// enumeration variables
wxString str;
long dummy;

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

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

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

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

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

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

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

*GetFirstGroup* (p. 271)  
*GetNextGroup* (p. 272)  
*GetFirstEntry* (p. 272)  
*GetNextEntry* (p. 272)  
*GetNumberOfEntries* (p. 272)  
*GetNumberOfGroups* (p. 272)

### Tests of existence

*HasGroup* (p. 273)  
*HasEntry* (p. 273)

*Exists* (p. 271)

*GetEntryType* (p. 271)

### Miscellaneous functions

*GetAppName* (p. 271)

*GetVendorName* (p. 273)

*SetUmask* (p. 598)

### Key access

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

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

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

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

*Read* (p. 273)

*Write* (p. 276)

*Flush* (p. 271)

### Rename entries/groups

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

*RenameEntry* (p. 275)

*RenameGroup* (p. 275)

### Delete entries/groups

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

*DeleteEntry* (p. 270)

*DeleteGroup* (p. 270)

*DeleteAll* (p. 270)

## Options

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

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

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

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

Although this feature is very useful, it may be annoying if you read a value which contains '\$' or '%' symbols (% is used for environment variables expansion under Windows) which are not used for environment variable expansion. In this situation you may call `SetExpandEnvVars(false)` just before reading this value and `SetExpandEnvVars(true)` just after. Another solution would be to prefix the offending symbols with a backslash.

The following functions control this option:

*IsExpandingEnvVars* (p. 273)

*SetExpandEnvVars* (p. 275)

*SetRecordDefaults* (p. 276)

*IsRecordingDefaults* (p. 273)

## `wxConfigBase::wxConfigBase`

**`wxConfigBase(const wxString& appName = wxEmptyString, const wxString& vendorName = wxEmptyString, const wxString& localFilename = wxEmptyString, const wxString& globalFilename = wxEmptyString, long style = 0, wxMBConv& conv = wxConvUTF8)`**

This is the default and only constructor of the `wxConfigBase` class, and derived classes.

### Parameters

*appName*

The application name. If this is empty, the class will normally use `wxApp::GetAppName` (p. 47) to set it. The application name is used in the registry key on Windows, and can be used to deduce the local filename parameter if that is missing.

*vendorName*

The vendor name. If this is empty, it is assumed that no vendor name is wanted, if this is optional for the current config class. The vendor name is appended to the application name for `wxRegConfig`.

#### *localFilename*

Some config classes require a local filename. If this is not present, but required, the application name will be used instead.

#### *globalFilename*

Some config classes require a global filename. If this is not present, but required, the application name will be used instead.

#### *style*

Can be one of `wxCONFIG_USE_LOCAL_FILE` and `wxCONFIG_USE_GLOBAL_FILE`. The style interpretation depends on the config class and is ignored by some implementations. For `wxFileConfig`, these styles determine whether a local or global config file is created or used: if `wxCONFIG_USE_GLOBAL_FILE` is used, then settings are read from the global config file and if `wxCONFIG_USE_LOCAL_FILE` is used, settings are read from and written to local config file (if they are both set, global file is read first, then local file, overwriting global settings). If the flag is present but the parameter is empty, the parameter will be set to a default. If the parameter is present but the style flag not, the relevant flag will be added to the style. For `wxRegConfig`, this GLOBAL flag refers to HKLM key while LOCAL one is for the usual HKCU one.

For `wxFileConfig` you can also add `wxCONFIG_USE_RELATIVE_PATH` by logically or'ing it to either of the `_FILE` options to tell `wxFileConfig` to use relative instead of absolute paths.

On non-VMS Unix systems, the default local configuration file is `~/ .appname`. However, this path may be also used as user data directory (see `wxStandardPaths::GetUserDataDir` (p. 1526)) if the application has several data files. In this case `wxCONFIG_USE_SUBDIR` flag, which changes the default local configuration file to `~/ .appname/appnameshould` be used. Notice that this flag is ignored on non-Unix system, including VMS, or if a non-default *localFilename* is provided. This function is new since `wxWidgets` version 2.8.2

For `wxFileConfig`, you can also add `wxCONFIG_USE_NO_ESCAPE_CHARACTERS` which will turn off character escaping for the values of entries stored in the config file: for example a `foo` key with some backslash characters will be stored as `foo=C:\mydir` instead of the usual storage of `foo=C:\\mydir`.

The `wxCONFIG_USE_NO_ESCAPE_CHARACTERS` style can be helpful if your config file must be read or written to by a non-`wxWidgets` program (which might not understand the escape characters). Note, however, that if `wxCONFIG_USE_NO_ESCAPE_CHARACTERS` style is used, it is now your application's responsibility to ensure that there is no newline or other illegal characters in a value, before writing that value to the file.

*conv*

This parameter is only used by `wxFileConfig` when compiled in Unicode mode. It specifies the encoding in which the configuration file is written.

### Remarks

By default, environment variable expansion is on and recording defaults is off.

### **wxConfigBase::~~wxConfigBase**

**~wxConfigBase()**

Empty but ensures that dtor of all derived classes is virtual.

### **wxConfigBase::Create**

**static wxConfigBase \* Create()**

Create a new config object: this function will create the "best" implementation of `wxConfig` available for the current platform, see comments near the definition of `wxCONFIG_WIN32_NATIVE` for details. It returns the created object and also sets it as the current one.

### **wxConfigBase::DontCreateOnDemand**

**void DontCreateOnDemand()**

Calling this function will prevent `Get()` from automatically creating a new config object if the current one is `NULL`. It might be useful to call it near the program end to prevent "accidental" creation of a new config object.

### **wxConfigBase::DeleteAll**

**bool DeleteAll()**

Delete the whole underlying object (disk file, registry key, ...). Primarily for use by uninstallation routine.

### **wxConfigBase::DeleteEntry**

**bool DeleteEntry(const wxString& key, bool bDeleteGroupIfEmpty = true)**

Deletes the specified entry and the group it belongs to if it was the last key in it and the second parameter is true.

### **wxConfigBase::DeleteGroup**

**bool DeleteGroup(const wxString& key)**

Delete the group (with all subgroups). If the current path is under the group being deleted

it is changed to its deepest still existing component. E.g. if the current path is /A/B/C/D and the group C is deleted the path becomes /A/B.

### **wxConfigBase::Exists**

**bool Exists(wxString& strName) const**

returns `true` if either a group or an entry with a given name exists

### **wxConfigBase::Flush**

**bool Flush(bool bCurrentOnly = false)**

permanently writes all changes (otherwise, they're only written from object's destructor)

### **wxConfigBase::Get**

**static wxConfigBase \* Get(bool CreateOnDemand = true)**

Get the current config object. If there is no current object and *CreateOnDemand* is true, creates one (using *Create*) unless *DontCreateOnDemand* was called previously.

### **wxConfigBase::GetAppName**

**wxString GetAppName() const**

Returns the application name.

### **wxConfigBase::GetEntryType**

**enum wxConfigBase::EntryType GetEntryType(const wxString& name) const**

Returns the type of the given entry or *Unknown* if the entry doesn't exist. This function should be used to decide which version of *Read()* should be used because some of *wxConfig* implementations will complain about type mismatch otherwise: e.g., an attempt to read a string value from an integer key with *wxRegConfig* will fail.

The result is an element of enum *EntryType*:

```
enum EntryType
{
    Type_Unknown,
    Type_String,
    Type_Boolean,
    Type_Integer,
    Type_Float
};
```

### **wxConfigBase::GetFirstGroup**

**bool GetFirstGroup(wxString& str, long& index) const**

Gets the first group.

**wxPython note:** The wxPython version of this method returns a 3-tuple consisting of the continue flag, the value string, and the index for the next call.

**wxPerl note:** In wxPerl this method takes no arguments and returns a 3-element list ( `continue`, `str`, `index` ).

### **wxConfigBase::GetFirstEntry**

**bool GetFirstEntry(wxString& str, long& index) const**

Gets the first entry.

**wxPython note:** The wxPython version of this method returns a 3-tuple consisting of the continue flag, the value string, and the index for the next call.

**wxPerl note:** In wxPerl this method takes no arguments and returns a 3-element list ( `continue`, `str`, `index` ).

### **wxConfigBase::GetNextGroup**

**bool GetNextGroup(wxString& str, long& index) const**

Gets the next group.

**wxPython note:** The wxPython version of this method returns a 3-tuple consisting of the continue flag, the value string, and the index for the next call.

**wxPerl note:** In wxPerl this method only takes the `index` parameter and returns a 3-element list ( `continue`, `str`, `index` ).

### **wxConfigBase::GetNextEntry**

**bool GetNextEntry(wxString& str, long& index) const**

Gets the next entry.

**wxPython note:** The wxPython version of this method returns a 3-tuple consisting of the continue flag, the value string, and the index for the next call.

**wxPerl note:** In wxPerl this method only takes the `index` parameter and returns a 3-element list ( `continue`, `str`, `index` ).

### **wxConfigBase::GetNumberOfEntries**

**uint GetNumberOfEntries(bool bRecursive = false) const**

### **wxConfigBase::GetNumberOfGroups**

**uint GetNumberOfGroups(bool bRecursive = false) const**



Get number of entries/subgroups in the current group, with or without its subgroups.

**wxConfigBase::GetPath****const wxString& GetPath() const**

Retrieve the current path (always as absolute path).

**wxConfigBase::GetVendorName****wxString GetVendorName() const**

Returns the vendor name.

**wxConfigBase::HasEntry****bool HasEntry(wxString& strName) const**

returns `true` if the entry by this name exists

**wxConfigBase::HasGroup****bool HasGroup(const wxString& strName) const**

returns `true` if the group by this name exists

**wxConfigBase::IsExpandingEnvVars****bool IsExpandingEnvVars() const**

Returns `true` if we are expanding environment variables in key values.

**wxConfigBase::IsRecordingDefaults****bool IsRecordingDefaults() const**

Returns `true` if we are writing defaults back to the config file.

**wxConfigBase::Read****bool Read(const wxString& key, wxString\* str) const**

Read a string from the key, returning `true` if the value was read. If the key was not found, `str` is not changed.

**bool Read(const wxString& key, wxString\* str, const wxString& defaultVal) const**

Read a string from the key. The default value is returned if the key was not found.

Returns `true` if value was really read, `false` if the default was used.

**wxString Read(const wxString& key, const wxString& defaultVal) const**

Another version of *Read()*, returning the string value directly.

**bool Read(const wxString& key, long\* l) const**

Reads a long value, returning `true` if the value was found. If the value was not found, *l* is not changed.

**bool Read(const wxString& key, long\* l, long defaultVal) const**

Reads a long value, returning `true` if the value was found. If the value was not found, *defaultVal* is used instead.

**long Read(const wxString& key, long defaultVal) const**

Reads a long value from the key and returns it. *defaultVal* is returned if the key is not found.

NB: writing

```
conf->Read("key", 0);
```

won't work because the call is ambiguous: compiler can not choose between two *Read* functions. Instead, write:

```
conf->Read("key", 0l);
```

**bool Read(const wxString& key, double\* d) const**

Reads a double value, returning `true` if the value was found. If the value was not found, *d* is not changed.

**bool Read(const wxString& key, double\* d, double defaultVal) const**

Reads a double value, returning `true` if the value was found. If the value was not found, *defaultVal* is used instead.

**bool Read(const wxString& key, bool\* b) const**

Reads a bool value, returning `true` if the value was found. If the value was not found, *b* is not changed.

**bool Read(const wxString& key, bool\* d, bool defaultVal) const**

Reads a bool value, returning `true` if the value was found. If the value was not found, *defaultVal* is used instead.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

<b>Read(key, default="")</b>	Returns a string.
<b>ReadInt(key, default=0)</b>	Returns an int.
<b>ReadFloat(key, default=0.0)</b>	Returns a floating point number.

**wxPerl note:** In place of a single overloaded method, wxPerl uses:

<b>Read(key, default="")</b>	Returns a string
<b>ReadInt(key, default=0)</b>	Returns an integer
<b>ReadFloat(key, default=0.0)</b>	Returns a floating point number
<b>ReadBool(key, default=0)</b>	Returns a boolean

### **wxConfigBase::RenameEntry**

**bool RenameEntry(const wxString& oldName, const wxString& newName)**

Renames an entry in the current group. The entries names (both the old and the new one) shouldn't contain backslashes, i.e. only simple names and not arbitrary paths are accepted by this function.

Returns `false` if *oldName* doesn't exist or if *newName* already exists.

### **wxConfigBase::RenameGroup**

**bool RenameGroup(const wxString& oldName, const wxString& newName)**

Renames a subgroup of the current group. The subgroup names (both the old and the new one) shouldn't contain backslashes, i.e. only simple names and not arbitrary paths are accepted by this function.

Returns `false` if *oldName* doesn't exist or if *newName* already exists.

### **wxConfigBase::Set**

**static wxConfigBase \* Set(wxConfigBase \*pConfig)**

Sets the config object as the current one, returns the pointer to the previous current object (both the parameter and returned value may be NULL)

### **wxConfigBase::SetExpandEnvVars**

**void SetExpandEnvVars (bool bDolt = true)**

Determine whether we wish to expand environment variables in key values.

### **wxConfigBase::SetPath**

**void SetPath(const wxString& strPath)**

Set current path: if the first character is '/', it is the absolute path, otherwise it is a relative path. '.' is supported. If strPath doesn't exist it is created.

**wxConfigBase::SetRecordDefaults**

**void SetRecordDefaults(bool bDoIt = true)**

Sets whether defaults are recorded to the config file whenever an attempt to read the value which is not present in it is done.

If on (default is off) all default values for the settings used by the program are written back to the config file. This allows the user to see what config options may be changed and is probably useful only for wxFileConfig.

**wxConfigBase::Write**

**bool Write(const wxString& key, const wxString& value)**

**bool Write(const wxString& key, long value)**

**bool Write(const wxString& key, double value)**

**bool Write(const wxString& key, bool value)**

These functions write the specified value to the config file and return `true` on success.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

<b>Write(key, value)</b>	Writes a string.
<b>WriteInt(key, value)</b>	Writes an int.
<b>WriteFloat(key, value)</b>	Writes a floating point number.

**wxPerl note:** In place of a single overloaded method, wxPerl uses:

<b>Write(key, value)</b>	Writes a string
<b>WriteInt(key, value)</b>	Writes an integer
<b>WriteFloat(key, value)</b>	Writes a floating point number
<b>WriteBool(key, value)</b>	Writes a boolean

## wxConnection

A wxConnection object represents the connection between a client and a server. It is

created by making a connection using a *wxClient* (p. 191) object, or by the acceptance of a connection by a *wxServer* (p. 1431) object. The bulk of a DDE-like (Dynamic Data Exchange) conversation is controlled by calling members in a **wxConnection** object or by overriding its members. The actual DDE-based implementation using *wxDDEConnection* is available on Windows only, but a platform-independent, socket-based version of this API is available using *wxTCPConnection*, which has the same API.

An application should normally derive a new connection class from *wxConnection*, in order to override the communication event handlers to do something interesting.

### Derived from

*wxConnectionBase*  
*wxObject* (p. 1148)

### Include files

<wx/ipc.h>

### Types

*wxIPCFormat* is defined as follows:

```
enum wxIPCFormat
{
    wxIPC_INVALID =          0,
    wxIPC_TEXT =             1, /* CF_TEXT */
    wxIPC_BITMAP =           2, /* CF_BITMAP */
    wxIPC_METAFILE =         3, /* CF_METAFILEPICT */
    wxIPC_SYLK =              4,
    wxIPC_DIF =               5,
    wxIPC_TIFF =              6,
    wxIPC_OEMTEXT =           7, /* CF_OEMTEXT */
    wxIPC_DIB =               8, /* CF_DIB */
    wxIPC_PALETTE =           9,
    wxIPC_PENDATA =          10,
    wxIPC_RIFF =              11,
    wxIPC_WAVE =              12,
    wxIPC_UNICODETEXT =       13,
    wxIPC_ENHMETAFILE =       14,
    wxIPC_FILENAME =          15, /* CF_HDROP */
    wxIPC_LOCALE =            16,
    wxIPC_PRIVATE =           20
};
```

### See also

*wxClient* (p. 191), *wxServer* (p. 1431), *Interprocess communications overview* (p. 2175)

## wxConnection::wxConnection

**wxConnection()**

**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` (p. 1431) and/or `wxClient::OnMakeConnection` (p. 192) members should be replaced by functions which construct the new connection object.

If the arguments of the `wxConnection` constructor are void then the `wxConnection` object manages its own connection buffer, allocating memory as needed. A programmer-supplied buffer cannot be increased if necessary, and the program will assert if it is not large enough. The programmer-supplied buffer is included mainly for backwards compatibility.

**wxConnection::Advise**

**bool Advise(const wxString& item, char\* data, int size = -1, wxIPCFormat format = wxCF\_TEXT)**

Called by the server application to advise the client of a change in the data associated with the given item. Causes the client connection's `wxConnection::OnAdvise` (p. 278) member to be called. Returns true if successful.

**wxConnection::Execute**

**bool Execute(char\* data, int size = -1, wxIPCFormat format = wxCF\_TEXT)**

Called by the client application to execute a command on the server. Can also be used to transfer arbitrary data to the server (similar to `wxConnection::Poke` (p. 280) in that respect). Causes the server connection's `wxConnection::OnExecute` (p. 279) member to be called. Returns true if successful.

**wxConnection::Disconnect**

**bool Disconnect()**

Called by the client or server application to disconnect from the other program; it causes the `wxConnection::OnDisconnect` (p. 279) message to be sent to the corresponding connection object in the other program. Returns true if successful or already disconnected. The application that calls **Disconnect** must explicitly delete its side of the connection.

**wxConnection::OnAdvise**

**virtual bool OnAdvise(const wxString& topic, const wxString& item, char\* data, int size, wxIPCFormat format)**

Message sent to the client application when the server notifies it of a change in the data associated with the given item, using `Advise` (p. 278).

**wxConnection::OnDisconnect****virtual bool OnDisconnect()**

Message sent to the client or server application when the other application notifies it to end the connection. The default behaviour is to delete the connection object and return true, so applications should generally override **OnDisconnect** (finally calling the inherited method as well) so that they know the connection object is no longer available.

**wxConnection::OnExecute****virtual bool OnExecute(const wxString& topic, char\* data, int size, wxIPCFormat format)**

Message sent to the server application when the client notifies it to execute the given data, using *Execute* (p. 278). Note that there is no item associated with this message.

**wxConnection::OnPoke****virtual bool OnPoke(const wxString& topic, const wxString& item, char\* data, int size, wxIPCFormat format)**

Message sent to the server application when the client notifies it to accept the given data.

**wxConnection::OnRequest****virtual char\* OnRequest(const wxString& topic, const wxString& item, int \*size, wxIPCFormat format)**

Message sent to the server application when the client calls *wxConnection::Request* (p. 280). The server's *OnRequest* (p. 279) method should respond by returning a character string, or NULL to indicate no data, and setting \*size. The character string must of course persist after the call returns.

**wxConnection::OnStartAdvise****virtual bool OnStartAdvise(const wxString& topic, const wxString& item)**

Message sent to the server application by the client, when the client wishes to start an 'advise loop' for the given topic and item. The server can refuse to participate by returning false.

**wxConnection::OnStopAdvise****virtual bool OnStopAdvise(const wxString& topic, const wxString& item)**

Message sent to the server application by the client, when the client wishes to stop an 'advise loop' for the given topic and item. The server can refuse to stop the advise loop by returning false, although this doesn't have much meaning in practice.

**wxConnection::Poke**

**bool Poke(const wxString& item, char\* data, int size = -1, wxIPCFormat format = wxCF\_TEXT)**

Called by the client application to poke data into the server. Can be used to transfer arbitrary data to the server. Causes the server connection's *wxConnection::OnPoke* (p. 279) member to be called. If size is -1 the size is computed from the string length of data.

Returns true if successful.

**wxConnection::Request**

**char\* Request(const wxString& item, int \*size, wxIPCFormat format = wxIPC\_TEXT)**

Called by the client application to request data from the server. Causes the server connection's *wxConnection::OnRequest* (p. 279) member to be called. Size may be NULL or a pointer to a variable to receive the size of the requested item.

Returns a character string (actually a pointer to the connection's buffer) if successful, NULL otherwise. This buffer does not need to be deleted.

**wxConnection::StartAdvise**

**bool StartAdvise(const wxString& item)**

Called by the client application to ask if an advise loop can be started with the server. Causes the server connection's *wxConnection::OnStartAdvise* (p. 279) member to be called. Returns true if the server okays it, false otherwise.

**wxConnection::StopAdvise**

**bool StopAdvise(const wxString& item)**

Called by the client application to ask if an advise loop can be stopped. Causes the server connection's *wxConnection::OnStopAdvise* (p. 279) member to be called. Returns true if the server okays it, false otherwise.

**wxChildFocusEvent**

A child focus event is sent to a (parent-)window when one of its child windows gains focus, so that the window could restore the focus back to its corresponding child if it loses it now and regains later.

Notice that child window is the direct child of the window receiving event. Use *FindFocus* (p. 1806) to retrieve the window which is actually getting focus.

**Derived from**

*wxCommandEvent* (p. 250)

*wxEvent* (p. 572)



*wxObject* (p. 1148)

### Include files

<wx/event.h>

### Event table macros

To process a child focus event, use this event handler macro to direct input to a member function that takes a `wxChildFocusEvent` argument.

**EVT\_CHILD\_FOCUS(func)**                      Process a `wxEVT_CHILD_FOCUS` event.

### See also

*Event handling overview* (p. 2077)

## **wxChildFocusEvent::wxChildFocusEvent**

**wxChildFocusEvent(wxWindow \*win = NULL)**

Constructor.

### Parameters

*win*

The direct child which is (or which contains the window which is) receiving the focus.

## **wxChildFocusEvent::GetWindow**

Returns the direct child which receives the focus, or a (grand-)parent of the control receiving the focus.

To get the actually focused control use *wxWindow::FindFocus* (p. 1806).

## **wxContextMenuEvent**

This class is used for context menu events, sent to give the application a chance to show a context (popup) menu.

Note that if *GetPosition* (p. 282) returns `wxDefaultPosition`, this means that the event originated from a keyboard context button event, and you should compute a suitable position yourself, for example by calling *wxGetMousePosition* (p. 1957).

When a keyboard context menu button is pressed on Windows, a right-click event with default position is sent first, and if this event is not processed, the context menu event is sent. So if you process mouse events and you find your context menu event handler is not being called, you could call `wxEvt::Skip` for mouse right-down events.

### Derived from

*wxCommandEvent* (p. 250)

*wxEvent* (p. 572)

*wxObject* (p. 1148)

### Include files

<wx/event.h>

### Event table macros

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

**EVT\_CONTEXT\_MENU(func)**                      A right click (or other context menu command depending on platform) has been detected.

### See also

*Command events* (p. 250),

*Event handling overview* (p. 2077)

## **wxContextMenuEvent::wxContextMenuEvent**

**wxContextMenuEvent(WXTYPE id = 0, int id = 0, const wxPoint& pos=wxDefaultPosition)**

Constructor.

## **wxContextMenuEvent::GetPosition**

**wxPoint GetPosition() const**

Returns the position in screen coordinates at which the menu should be shown. Use *wxWindow::ScreenToClient* (p. 1832) to convert to client coordinates. You can also omit a position from *wxWindow::PopupMenu* (p. 1828) in order to use the current mouse pointer position.

If the event originated from a keyboard event, the value returned from this function will be *wxDefaultPosition*.

## **wxContextMenuEvent::SetPosition**

**void SetPosition(const wxPoint& point)**

Sets the position at which the menu should be shown.

## **wxContextHelp**

This class changes the cursor to a query and puts the application into a 'context-sensitive

help mode'. When the user left-clicks on a window within the specified window, a `wxEVT_HELP` event is sent to that control, and the application may respond to it by popping up some help.

For example:

```
wxContextHelp contextHelp(myWindow);
```

There are a couple of ways to invoke this behaviour implicitly:

- Use the `wxDIALOG_EX_CONTEXTHELP` style for a dialog (Windows only). This will put a question mark in the titlebar, and Windows will put the application into context-sensitive help mode automatically, with further programming.
- Create a `wxContextHelpButton` (p. 284), whose predefined behaviour is to create a context help object. Normally you will write your application so that this button is only added to a dialog for non-Windows platforms (use `wxDIALOG_EX_CONTEXTHELP` on Windows).

Note that on Mac OS X, the cursor does not change when in context-sensitive help mode.

#### **Derived from**

`wxObject` (p. 1148)

#### **Include files**

`<wx/cshelp.h>`

#### **See also**

`wxHelpEvent` (p. 815), `wxHelpController` (p. 809), `wxContextHelpButton` (p. 284)

### **wxContextHelp::wxContextHelp**

**wxContextHelp**(wxWindow\* window = NULL, bool doNow = true)

Constructs a context help object, calling `BeginContextHelp` (p. 283) if `doNow` is true (the default).

If `window` is NULL, the top window is used.

### **wxContextHelp::~~wxContextHelp**

**~wxContextHelp**()

Destroys the context help object.

### **wxContextHelp::BeginContextHelp**

**bool BeginContextHelp**(wxWindow\* window = NULL)

Puts the application into context-sensitive help mode. *window* is the window which will be used to catch events; if NULL, the top window will be used.

Returns true if the application was successfully put into context-sensitive help mode. This function only returns when the event loop has finished.

### **wxContextHelp::EndContextHelp**

#### **bool EndContextHelp()**

Ends context-sensitive help mode. Not normally called by the application.

## **wxContextHelpButton**

Instances of this class may be used to add a question mark button that when pressed, puts the application into context-help mode. It does this by creating a *wxContextHelp* (p. 282) object which itself generates a wxEVT\_HELP event when the user clicks on a window.

On Windows, you may add a question-mark icon to a dialog by use of the wxDIALOG\_EX\_CONTEXTHELP extra style, but on other platforms you will have to add a button explicitly, usually next to OK, Cancel or similar buttons.

#### **Derived from**

*wxBitmapButton* (p. 139)

*wxButton* (p. 164)

*wxControl* (p. 285)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

#### **Include files**

<wx/cshelp.h>

#### **See also**

*wxBitmapButton* (p. 139), *wxContextHelp* (p. 282)

### **wxContextHelpButton::wxContextHelpButton**

#### **wxContextHelpButton()**

Default constructor.

**wxContextHelpButton(wxWindow\* parent, wxWindowID id = wxID\_CONTEXT\_HELP, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxBU\_AUTODRAW)**

Constructor, creating and showing a context help button.

**Parameters***parent*

Parent window. Must not be NULL.

*id*

Button identifier. Defaults to `wxID_CONTEXT_HELP`.

*pos*

Button position.

*size*

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

*style*

Window style.

**Remarks**

Normally you need pass only the parent window to the constructor, and use the defaults for the remaining parameters.

**wxControl**

This is the base class for a control or "widget".

A control is generally a small window which processes user input and/or displays one or more item of data.

**Derived from**

*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/control.h>

**See also**

*wxValidator* (p. 1767)

**wxControl::Command**

**void Command(wxCommandEvent& event)**

Simulates the effect of the user issuing a command to the item. See *wxCommandEvent* (p. 250).

### **wxControl::GetLabel**

**const wxString& GetLabel() const**

Returns the control's text.

Note that the returned string contains the mnemonics (& characters) if any, use *wxControl::GetLabelText* (p. 286) if they are undesired.

### **wxControl::GetLabelText**

**const wxString& GetLabelText() const**

**static const wxString& GetLabelText(const wxString& label)**

Returns the control's label or the given *label* string for the static version without the mnemonics characters.

### **wxControl::SetLabel**

**void SetLabel(const wxString& label)**

Sets the item's text.

The & characters in the *label* are special and indicate that the following character is a mnemonic for this control and can be used to activate it from the keyboard (typically by using *Alt* key in combination with it). To insert a literal ampersand character, you need to double it, i.e. use "&&".

## **wxControlWithItems**

This class is an abstract base class for some wxWidgets controls which contain several items, such as *wxListBox* (p. 974) and *wxCheckListBox* (p. 183) derived from it, *wxChoice* (p. 186) and *wxComboBox* (p. 225).

It defines the methods for accessing the controls items and although each of the derived classes implements them differently, they still all conform to the same interface.

The items in a *wxControlWithItems* have (non-empty) string labels and, optionally, client data associated with them. Client data may be of two different kinds: either simple untyped (*void \**) pointers which are simply stored by the control but not used in any way by it, or typed pointers (*wxClientData \**) which are owned by the control meaning that the typed client data (and only it) will be deleted when an item is *deleted* (p. 288) or the entire control is *cleared* (p. 287) (which also happens when it is destroyed). Finally note that in the same control all items must have client data of the same type (typed or untyped), if any. This type is determined by the first call to *Append* (p. 287) (the version with client data pointer) or *SetClientData* (p. 291).

**Derived from**

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/ctrlsub.h> but usually never included directly

**wxControlWithItems::Append**

**int Append(const wxString& item)**

Adds the item to the end of the list box.

**int Append(const wxString& item, void \*clientData)**

**int Append(const wxString& item, wxClientData \*clientData)**

Adds the item to the end of the list box, associating the given, typed or untyped, client data pointer with the item.

**void Append(const wxArrayString& strings)**

Appends several items at once to the control. Notice that calling this method may be much faster than appending the items one by one if you need to add a lot of items.

**Parameters**

*item*

String to add.

*clientData*

Client data to associate with the item.

**Return value**

When appending a single item, the return value is the index of the newly added item which may be different from the last one if the control is sorted (e.g. has `wxLB_SORT` or `wxCB_SORT` style).

**wxControlWithItems::Clear**

**void Clear()**

Removes all items from the control.

*Clear()* also deletes the client data of the existing items if it is owned by the control.

**wxControlWithItems::Delete****void Delete(unsigned int *n*)**

Deletes an item from the control. The client data associated with the item will be also deleted if it is owned by the control.

Note that it is an error (signalled by an assert failure in debug builds) to remove an item with the index negative or greater or equal than the number of items in the control.

**Parameters***n*

The zero-based item index.

**See also**

*Clear* (p. 287)

**wxControlWithItems::FindString****int FindString(const wxString& *string*, bool *caseSensitive* = false)**

Finds an item whose label matches the given string.

**Parameters***string*

String to find.

*caseSensitive*

Whether search is case sensitive (default is not).

**Return value**

The zero-based position of the item, or `wxNOT_FOUND` if the string was not found.

**wxControlWithItems::GetClientData****void \* GetClientData(unsigned int *n*) const**

Returns a pointer to the client data associated with the given item (if any). It is an error to call this function for a control which doesn't have untyped client data at all although it is ok to call it even if the given item doesn't have any client data associated with it (but other items do).

**Parameters***n*

The zero-based position of the item.



**Return value**

A pointer to the client data, or `NULL` if not present.

**wxControlWithItems::GetClientObject**

**wxClientData \* GetClientObject(unsigned int *n*) const**

Returns a pointer to the client data associated with the given item (if any). It is an error to call this function for a control which doesn't have typed client data at all although it is ok to call it even if the given item doesn't have any client data associated with it (but other items do).

**Parameters**

*n*

The zero-based position of the item.

**Return value**

A pointer to the client data, or `NULL` if not present.

**wxControlWithItems::GetCount**

**unsigned int GetCount() const**

Returns the number of items in the control.

**See also**

*IsEmpty* (p. 291)

**wxControlWithItems::GetSelection**

**int GetSelection() const**

Returns the index of the selected item or `wxNOT_FOUND` if no item is selected.

**Return value**

The position of the current selection.

**Remarks**

This method can be used with single selection list boxes only, you should use *wxListBox::GetSelections* (p. 977) for the list boxes with `wxLB_MULTIPLE` style.

**See also**

*SetSelection* (p. 292), *GetStringSelection* (p. 290)

**wxControlWithItems::GetString**

**wxString GetString(unsigned int *n*) const**

Returns the label of the item with the given index.

**Parameters**

*n*

The zero-based index.

**Return value**

The label of the item or an empty string if the position was invalid.

**wxControlWithItems::GetStrings****wxArrayString GetStrings() const**

Returns the array of the labels of all items in the control.

**wxControlWithItems::GetStringSelection****wxString GetStringSelection() const**

Returns the label of the selected item or an empty string if no item is selected.

**See also**

*GetSelection* (p. 289)

**wxControlWithItems::Insert****int Insert(const wxString& *item*, unsigned int *pos*)**

Inserts the item into the list before *pos*. Not valid for `wxLB_SORT` or `wxCB_SORT` styles, use `Append` instead.

**int Insert(const wxString& *item*, unsigned int *pos*, void \**clientData*)****int Insert(const wxString& *item*, unsigned int *pos*, wxClientData \**clientData*)**

Inserts the item into the list before *pos*, associating the given, typed or untyped, client data pointer with the item. Not valid for `wxLB_SORT` or `wxCB_SORT` styles, use `Append` instead.

**Parameters**

*item*

String to add.

*pos*

Position to insert item before, zero based.

*clientData*

Client data to associate with the item.

### **Return value**

The return value is the index of the newly inserted item. If the insertion failed for some reason, -1 is returned.

### **wxControlWithItems::IsEmpty**

**bool IsEmpty() const**

Returns `true` if the control is empty or `false` if it has some items.

### **See also**

*GetCount* (p. 289)

### **wxControlWithItems::Number**

**int Number() const**

**Obsolescence note:** This method is obsolete and was replaced with *GetCount* (p. 289), please use the new method in the new code. This method is only available if `wxWidgets` was compiled with `WXWIN_COMPATIBILITY_2_2` defined and will disappear completely in future versions.

### **wxControlWithItems::Select**

**void Select(int *n*)**

This is the same as *SetSelection* (p. 292) and exists only because it is slightly more natural for controls which support multiple selection.

### **wxControlWithItems::SetClientData**

**void SetClientData(unsigned int *n*, void \**data*)**

Associates the given untyped client data pointer with the given item. Note that it is an error to call this function if any typed client data pointers had been associated with the control items before.

### **Parameters**

*n*

The zero-based item index.

*data*

The client data to associate with the item.

**wxControlWithItems::SetClientObject****void SetClientObject(unsigned int *n*, wxClientData \**data*)**

Associates the given typed client data pointer with the given item: the *data* object will be deleted when the item is deleted (either explicitly by using *Deletes* (p. 288) or implicitly when the control itself is destroyed).

Note that it is an error to call this function if any untyped client data pointers had been associated with the control items before.

**Parameters***n*

The zero-based item index.

*data*

The client data to associate with the item.

**wxControlWithItems::SetSelection****void SetSelection(int *n*)**

Sets the selection to the given item *n* or removes the selection entirely if *n* == wxNOT\_FOUND.

Note that this does not cause any command events to be emitted nor does it deselect any other items in the controls which support multiple selections.

**Parameters***n*

The string position to select, starting from zero.

**See also**

*SetString* (p. 292), *SetStringSelection* (p. 293)

**wxControlWithItems::SetString****void SetString(unsigned int *n*, const wxString& *string*)**

Sets the label for the given item.

**Parameters***n*

The zero-based item index.

*string*

The label to set.

### **wxControlWithItems::SetStringSelection**

**bool SetStringSelection(const wxString& *string*)**

Selects the item with the specified string in the control. This doesn't cause any command events being emitted.

#### **Parameters**

*string*

The string to select.

#### **Return value**

`true` if the specified string has been selected, `false` if it wasn't found in the control.

#### **See also**

*SetSelection* (p. 292)

## **wxCountingOutputStream**

`wxCountingOutputStream` is a specialized output stream which does not write any data anywhere, instead it counts how many bytes would get written if this were a normal stream. This can sometimes be useful or required if some data gets serialized to a stream or compressed by using stream compression and thus the final size of the stream cannot be known other than pretending to write the stream. One case where the resulting size would have to be known is if the data has to be written to a piece of memory and the memory has to be allocated before writing to it (which is probably always the case when writing to a memory stream).

#### **Derived from**

*wxOutputStream* (p. 1156) *wxStreamBase* (p. 1544)

#### **Include files**

<wx/stream.h>

### **wxCountingOutputStream::wxCountingOutputStream**

**wxCountingOutputStream()**

Creates a `wxCountingOutputStream` object.

### **wxCountingOutputStream::~~wxCountingOutputStream**

**~wxCountingOutputStream()**

Destructor.

**wxCountingOutputStream::GetSize****size\_t GetSize() const**

Returns the current size of the stream.

**wxCriticalSection**

A critical section object is used for exactly the same purpose as *mutexes* (p. 1131). The only difference is that under Windows platform critical sections are only visible inside one process, while mutexes may be shared between processes, so using critical sections is slightly more efficient. The terminology is also slightly different: mutex may be locked (or acquired) and unlocked (or released) while critical section is entered and left by the program.

Finally, you should try to use *wxCriticalSectionLocker* (p. 295) class whenever possible instead of directly using *wxCriticalSection* for the same reasons *wxMutexLocker* (p. 1133) is preferable to *wxMutex* (p. 1131) - please see *wxMutex* for an example.

**Derived from**

None.

**Include files**

<wx/thread.h>

**See also**

*wxThread* (p. 1670), *wxCondition* (p. 259), *wxCriticalSectionLocker* (p. 295)

**wxCriticalSection::wxCriticalSection****wxCriticalSection()**

Default constructor initializes critical section object.

**wxCriticalSection::~~wxCriticalSection****~wxCriticalSection()**

Destructor frees the resources.

**wxCriticalSection::Enter**

**void Enter()**

Enter the critical section (same as locking a mutex). There is no error return for this function. After entering the critical section protecting some global data the thread running in critical section may safely use/modify it.

**wxCriticalSection::Leave****void Leave()**

Leave the critical section allowing other threads use the global data protected by it. There is no error return for this function.

**wxCriticalSectionLocker**

This is a small helper class to be used with *wxCriticalSection* (p. 294) objects. A *wxCriticalSectionLocker* enters the critical section in the constructor and leaves it in the destructor making it much more difficult to forget to leave a critical section (which, in general, will lead to serious and difficult to debug problems).

Example of using it:

```
void Set Foo()
{
    // gs_critSect is some (global) critical section guarding access
    to the
    // object "foo"
    wxCriticalSectionLocker locker(gs_critSect);

    if ( ... )
    {
        // do something
        ...

        return;
    }

    // do something else
    ...

    return;
}
```

Without *wxCriticalSectionLocker*, you would need to remember to manually leave the critical section before each `return`.

**Derived from**

None.

**Include files**

<wx/thread.h>

**See also**

*wxCriticalSection* (p. 294), *wxMutexLocker* (p. 1133)

**wxCriticalSectionLocker::wxCriticalSectionLocker**

**wxCriticalSectionLocker**(**wxCriticalSection&** *criticalsection*)

Constructs a *wxCriticalSectionLocker* object associated with given *criticalsection* and enters it.

**wxCriticalSectionLocker::~~wxCriticalSectionLocker**

**~wxCriticalSectionLocker**()

Destructor leaves the critical section.

**wxCSCnv**

This class converts between any character sets and Unicode. It has one predefined instance, **wxCnvLocal**, for the default user character set.

**Derived from**

*wxMBConv* (p. 1038)

**Include files**

<wx/strconv.h>

**See also**

*wxMBConv* (p. 1038), *wxEncodingConverter* (p. 568), *wxMBConv classes overview* (p. 2059)

**wxCSCnv::wxCSCnv**

**wxCSCnv**(**const wxChar\*** *charset*)

**wxCSCnv**(**wxFontEncoding** *encoding*)

Constructor. You may specify either the name of the character set you want to convert from/to or an encoding constant. If the character set name (or the encoding) is not recognized, ISO 8859-1 is used as fall back.

**wxCSCnv::~~wxCSCnv**



**~wxCSConv()**

Destructor frees any resources needed to perform the conversion.

**wxCSConv::IsOk****bool IsOk() const**

Returns `true` if the charset (or the encoding) given at constructor is really available to use. Returns `false` if ISO 8859-1 will be used instead.

Note this does *not* mean that a given string will be correctly converted. A malformed string may still make conversion functions return `wxCONV_FAILED`.

This function is new since wxWidgets version 2.8.2

**wxCSConv::MB2WC****size\_t MB2WC(wchar\_t\* buf, const char\* psz, size\_t n) const**

Converts from the selected character set to Unicode. Returns length of string written to destination buffer.

**wxCSConv::WC2MB****size\_t WC2MB(char\* buf, const wchar\_t\* psz, size\_t n) const**

Converts from Unicode to the selected character set. Returns length of string written to destination buffer.

**wxCursor**

A cursor is a small bitmap usually used for denoting where the mouse pointer is, with a picture that might indicate the interpretation of a mouse click. As with icons, cursors in X and MS Windows are created in a different manner. Therefore, separate cursors will be created for the different environments. Platform-specific methods for creating a **wxCursor** object are catered for, and this is an occasion where conditional compilation will probably be required (see *wxIcon* (p. 894) for an example).

A single cursor object may be used in many windows (any subwindow type). The wxWidgets convention is to set the cursor for a window, as in X, rather than to set it globally as in MS Windows, although a global `::wxSetCursor` (p. 1946) is also available for MS Windows use.

**Derived from**

*wxBitmap* (p. 123)

*wxGDIObject* (p. 709)

*wxObject* (p. 1148)

**Include files**

<wx/cursor.h>

## Predefined objects

Objects:

**wxNullCursor**

Pointers:

**wxSTANDARD\_CURSOR**

**wxHOURLASS\_CURSOR**

**wxCROSS\_CURSOR**

## See also

*wxBitmap* (p. 123), *wxIcon* (p. 894), *wxWindow::SetCursor* (p. 1837), *::wxSetCursor* (p. 1946)

## wxCursor::wxCursor

**wxCursor()**

Default constructor.

**wxCursor(const char bits[], int width, int height, int hotSpotX=-1, int hotSpotY=-1, const char maskBits[]=NULL, wxColour\* fg=NULL, wxColour\* bg=NULL)**

Constructs a cursor by passing an array of bits (Motif and GTK+ only). *maskBits* is used only under Motif and GTK+. The parameters *fg* and *bg* are only present on GTK+, and force the cursor to use particular background and foreground colours.

If either *hotSpotX* or *hotSpotY* is -1, the hotspot will be the centre of the cursor image (Motif only).

**wxCursor(const wxString& cursorName, long type, int hotSpotX=0, int hotSpotY=0)**

Constructs a cursor by passing a string resource name or filename.

On MacOS when specifying a string resource name, first the color cursors 'crsr' and then the black/white cursors 'CURS' in the resource chain are scanned through.

*hotSpotX* and *hotSpotY* are currently only used under Windows when loading from an icon file, to specify the cursor hotspot relative to the top left of the image.

**wxCursor(int cursorId)**

Constructs a cursor using a cursor identifier.

**wxCursor(const wxImage& image)**

Constructs a cursor from a *wxImage*. The cursor is monochrome, colors with the RGB elements all greater than 127 will be foreground, colors less than this background. The

mask (if any) will be used as transparent.

In MSW the foreground will be white and the background black. If the cursor is larger than 32x32 it is resized. In GTK, the two most frequent colors will be used for foreground and background. The cursor will be displayed at the size of the image. On MacOS if the cursor is larger than 16x16 it is resized and currently only shown as black/white (mask respected).

**wxCursor(const wxCursor& cursor)**

Copy constructor, uses *reference counting* (p. 2046).

### Parameters

*bits*

An array of bits.

*maskBits*

Bits for a mask bitmap.

*width*

Cursor width.

*height*

Cursor height.

*hotSpotX*

Hotspot x coordinate.

*hotSpotY*

Hotspot y coordinate.

*type*

Icon type to load. Under Motif, *type* defaults to **wxBITMAP\_TYPE\_XBM**. Under Windows, it defaults to **wxBITMAP\_TYPE\_CUR\_RESOURCE**. Under MacOS, it defaults to **wxBITMAP\_TYPE\_MACCURSOR\_RESOURCE**.

Under X, the permitted cursor types are:

**wxBITMAP\_TYPE\_XBM**            Load an X bitmap file.

Under Windows, the permitted types are:

**wxBITMAP\_TYPE\_CUR**            Load a cursor from a .cur cursor file (only if  
USE\_RESOURCE\_LOADING\_IN\_MSW is  
enabled in setup.h).

**wxBITMAP\_TYPE\_CUR\_RESOURCE**    Load a Windows resource (as  
specified in the .rc file).

<b>wxBITMAP_TYPE_ICO</b>	Load a cursor from a .ico icon file (only if <code>USE_RESOURCE_LOADING_IN_MSW</code> is enabled in <code>setup.h</code> ). Specify <i>hotSpotX</i> and <i>hotSpotY</i> .
--------------------------	---

#### *cursorId*

A stock cursor identifier. May be one of:

<b>wxCURSOR_ARROW</b>	A standard arrow cursor.
<b>wxCURSOR_RIGHT_ARROW</b>	A standard arrow cursor pointing to the right.
<b>wxCURSOR_BLANK</b>	Transparent cursor.
<b>wxCURSOR_BULLSEYE</b>	Bullseye cursor.
<b>wxCURSOR_CHAR</b>	Rectangular character cursor.
<b>wxCURSOR_CROSS</b>	A cross cursor.
<b>wxCURSOR_HAND</b>	A hand cursor.
<b>wxCURSOR_IBEAM</b>	An I-beam cursor (vertical line).
<b>wxCURSOR_LEFT_BUTTON</b>	Represents a mouse with the left button depressed.
<b>wxCURSOR_MAGNIFIER</b>	A magnifier icon.
<b>wxCURSOR_MIDDLE_BUTTON</b>	Represents a mouse with the middle button depressed.
<b>wxCURSOR_NO_ENTRY</b>	A no-entry sign cursor.
<b>wxCURSOR_PAINT_BRUSH</b>	A paintbrush cursor.
<b>wxCURSOR_PENCIL</b>	A pencil cursor.
<b>wxCURSOR_POINT_LEFT</b>	A cursor that points left.
<b>wxCURSOR_POINT_RIGHT</b>	A cursor that points right.
<b>wxCURSOR_QUESTION_ARROW</b>	An arrow and question mark.
<b>wxCURSOR_RIGHT_BUTTON</b>	Represents a mouse with the right button depressed.
<b>wxCURSOR_SIZENESW</b>	A sizing cursor pointing NE-SW.
<b>wxCURSOR_SIZENS</b>	A sizing cursor pointing N-S.
<b>wxCURSOR_SIZENWSE</b>	A sizing cursor pointing NW-SE.
<b>wxCURSOR_SIZEWE</b>	A sizing cursor pointing W-E.

<b>wxCURSOR_SIZING</b>	A general sizing cursor.
<b>wxCURSOR_SPRAYCAN</b>	A spraycan cursor.
<b>wxCURSOR_WAIT</b>	A wait cursor.
<b>wxCURSOR_WATCH</b>	A watch cursor.
<b>wxCURSOR_ARROWWAIT</b>	A cursor with both an arrow and an hourglass, (windows.)

Note that not all cursors are available on all platforms.

*cursor*

Pointer or reference to a cursor to copy.

**wxPython note:** Constructors supported by wxPython are:

<b>wxCursor(name, flags, hotSpotX=0, hotSpotY=0)</b>	Constructs a cursor from a filename
<b>wxStockCursor(id)</b>	Constructs a stock cursor

**wxPerl note:** Constructors supported by wxPerl are:

- ::Cursor->new( name, type, hotSpotX = 0, hotSpotY = 0 )
- ::Cursor->new( id )
- ::Cursor->new( image )
- ::Cursor->newData( bits, width, height, hotSpotX = -1, hotSpotY = -1, maskBits = 0 )

### Example

The following is an example of creating a cursor from 32x32 bitmap data (`down_bits`) and a mask (`down_mask`) where 1 is black and 0 is white for the bits, and 1 is opaque and 0 is transparent for the mask. It works on Windows and GTK+.

```
static char down_bits[] = { 255, 255, 255, 255, 31,
    255, 255, 255, 31, 255, 255, 255, 31, 255, 255, 255,
    31, 255, 255, 255, 31, 255, 255, 255, 31, 255, 255,
    255, 31, 255, 255, 255, 31, 255, 255, 255, 25, 243,
    255, 255, 19, 249, 255, 255, 7, 252, 255, 255, 15, 254,
    255, 255, 31, 255, 255, 255, 191, 255, 255, 255, 255,
    255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
    255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
    255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
    255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
    255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
    255 };
```

```
static char down_mask[] = { 240, 1, 0, 0, 240, 1,
    0, 0, 240, 1, 0, 0, 240, 1, 0, 0, 240, 1, 0, 0, 240, 1,
    0, 0, 240, 1, 0, 0, 240, 1, 0, 0, 255, 31, 0, 0, 255,
    31, 0, 0, 254, 15, 0, 0, 252, 7, 0, 0, 248, 3, 0, 0,
    240, 1, 0, 0, 224, 0, 0, 0, 64, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0 };

#ifdef __WXMSW__
wxBitmap down_bitmap(down_bits, 32, 32);
wxBitmap down_mask_bitmap(down_mask, 32, 32);

down_bitmap.SetMask(new wxMask(down_mask_bitmap));
wxImage down_image = down_bitmap.ConvertToImage();
down_image.SetOption(wxIMAGE_OPTION_CUR_HOTSPOT_X, 6);
down_image.SetOption(wxIMAGE_OPTION_CUR_HOTSPOT_Y, 14);
wxCursor down_cursor = wxCursor(down_image);
#else
wxCursor down_cursor = wxCursor(down_bits, 32, 32,
    6, 14, down_mask, wxWHITE, wxBLACK);
#endif
```

## **wxCursor::~wxCursor**

### **~wxCursor()**

Destroys the cursor. See *reference-counted object destruction* (p. 2046) for more info.

A cursor can be reused for more than one window, and does not get destroyed when the window is destroyed. wxWidgets destroys all cursors on application exit, although it is best to clean them up explicitly.

## **wxCursor::IsOk**

### **bool IsOk() const**

Returns true if cursor data is present.

## **wxCursor::operator =**

### **wxCursor& operator =(const wxCursor& cursor)**

Assignment operator, using *reference counting* (p. 2046).

## **wxCustomDataObject**

wxCustomDataObject is a specialization of *wxDataObjectSimple* (p. 335) for some application-specific data in arbitrary (either custom or one of the standard ones). The only restriction is that it is supposed that this data can be copied bitwise (i.e. with `memcpy()`), so it would be a bad idea to make it contain a C++ object (though C struct is fine).

By default, `wxCustomDataObject` stores the data inside in a buffer. To put the data into the buffer you may use either *SetData* (p. 304) or *TakeData* (p. 304) depending on whether you want the object to make a copy of data or not.

If you already store the data in another place, it may be more convenient and efficient to provide the data on-demand which is possible too if you override the virtual functions mentioned below.

### Virtual functions to override

This class may be used as is, but if you don't want store the data inside the object but provide it on demand instead, you should override *GetSize* (p. 304), *GetData* (p. 304) and *SetData* (p. 304) (or may be only the first two or only the last one if you only allow reading/writing the data)

### Derived from

*wxDataObjectSimple* (p. 335)

*wxDataObject* (p. 311)

### Include files

<wx/dataobj.h>

### See also

*wxDataObject* (p. 311)

## **wxCustomDataObject::wxCustomDataObject**

**wxCustomDataObject(const wxDataFormat& *format* = wxFormatInvalid)**

The constructor accepts a *format* argument which specifies the (single) format supported by this object. If it isn't set here, *SetFormat* (p. 336) should be used.

## **wxCustomDataObject::~~wxCustomDataObject**

**~wxCustomDataObject()**

The destructor will free the data hold by the object. Notice that although it calls a virtual *Free()* (p. 304) function, the base class version will always be called (C++ doesn't allow calling virtual functions from constructors or destructors), so if you override *Free()*, you should override the destructor in your class as well (which would probably just call the derived class' version of *Free()*).

## **wxCustomDataObject::Alloc**

**virtual void \* Alloc(size\_t *size*)**

This function is called to allocate *size* bytes of memory from *SetData()*. The default version

just uses the operator `new`.

### **wxCustomDataObject::Free**

**virtual void Free()**

This function is called when the data is freed, you may override it to anything you want (or may be nothing at all). The default version calls `operator delete[]` on the data.

### **wxCustomDataObject::GetSize**

**virtual size\_t GetSize() const**

Returns the data size in bytes.

### **wxCustomDataObject::GetData**

**virtual void \* GetData() const**

Returns a pointer to the data.

### **wxCustomDataObject::SetData**

**virtual void SetData( size\_t size, const void \*data)**

Set the data. The data object will make an internal copy.

**wxPython note:** This method expects a string in wxPython. You can pass nearly any object by pickling it first.

### **wxCustomDataObject::TakeData**

**virtual void TakeData( size\_t size, const void \*data)**

Like *SetData* (p. 304), but doesn't copy the data - instead the object takes ownership of the pointer.

**wxPython note:** This method expects a string in wxPython. You can pass nearly any object by pickling it first.

## **wxDataFormat**

A `wxDataFormat` is an encapsulation of a platform-specific format handle which is used by the system for the clipboard and drag and drop operations. The applications are usually only interested in, for example, pasting data from the clipboard only if the data is in a format the program understands and a data format is something which uniquely identifies this format.

On the system level, a data format is usually just a number (`CLIPFORMAT` under Windows or `Atom` under X11, for example) and the standard formats are, indeed, just numbers



which can be implicitly converted to `wxDataFormat`. The standard formats are:

<code>wxDF_INVALID</code>	An invalid format - used as default argument for functions taking a <code>wxDataFormat</code> argument sometimes
<code>wxDF_TEXT</code>	Text format ( <code>wxString</code> )
<code>wxDF_BITMAP</code>	A bitmap ( <code>wxBitmap</code> )
<code>wxDF_METAFILE</code>	A metafile ( <code>wxMetafile</code> , Windows only)
<code>wxDF_FILENAME</code>	A list of filenames
<code>wxDF_HTML</code>	An HTML string. This is only valid when passed to <code>wxSetClipboardData</code> when compiled with Visual C++ in non-Unicode mode

As mentioned above, these standard formats may be passed to any function taking `wxDataFormat` argument because `wxDataFormat` has an implicit conversion from them (or, to be precise from the type `wxDataFormat::NativeFormat` which is the type used by the underlying platform for data formats).

Aside the standard formats, the application may also use custom formats which are identified by their names (strings) and not numeric identifiers. Although internally custom format must be created (or *registered*) first, you shouldn't care about it because it is done automatically the first time the `wxDataFormat` object corresponding to a given format name is created. The only implication of this is that you should avoid having global `wxDataFormat` objects with non-default constructor because their constructors are executed before the program has time to perform all necessary initialisations and so an attempt to do clipboard format registration at this time will usually lead to a crash!

#### Virtual functions to override

None

#### Derived from

None

#### See also

*Clipboard and drag and drop overview* (p. 2150), *DnD sample* (p. 2033), *wxDataObject* (p. 311)

#### Include files

<wx/dataobj.h>

#### `wxDataFormat::wxDataFormat`

`wxDataFormat(NativeFormat format = wxDF_INVALID)`

Constructs a data format object for one of the standard data formats or an empty data object (use *SetType* (p. 306) or *SetId* (p. 306) later in this case)

**wxPerl note:** In wxPerl this function is named `newNative`.

### **wxDataFormat::wxDataFormat**

**wxDataFormat(const wxChar \*format)**

Constructs a data format object for a custom format identified by its name *format*.

**wxPerl note:** In wxPerl this function is named `newUser`.

### **wxDataFormat::operator ==**

**bool operator ==(const wxDataFormat& format) const**

Returns true if the formats are equal.

### **wxDataFormat::operator !=**

**bool operator !=(const wxDataFormat& format) const**

Returns true if the formats are different.

### **wxDataFormat::GetId**

**wxString GetId() const**

Returns the name of a custom format (this function will fail for a standard format).

### **wxDataFormat::GetType**

**NativeFormat GetType() const**

Returns the platform-specific number identifying the format.

### **wxDataFormat::SetId**

**void SetId(const wxChar \*format)**

Sets the format to be the custom format identified by the given name.

### **wxDataFormat::SetType**

**void SetType(NativeFormat format)**

Sets the format to the given value, which should be one of `wxDF_XXX` constants.

## **wxDatagramSocket**

### **Derived from**

*wxSocketBase* (p. 1472)

### **Include files**

<wx/socket.h>

### **wxDatagramSocket::wxDatagramSocket**

**wxDatagramSocket(wxSocketFlags flags = wxSOCKET\_NONE)**

Constructor.

### **Parameters**

*flags*

Socket flags (See *wxSocketBase::SetFlags* (p. 1480))

### **wxDatagramSocket::~~wxDatagramSocket**

**~wxDatagramSocket()**

Destructor. Please see *wxSocketBase::Destroy* (p. 1476).

### **wxDatagramSocket::ReceiveFrom**

**wxDatagramSocket& ReceiveFrom(wxSockAddress& address, void \* buffer, wxUint32 nbytes)**

This function reads a buffer of *nbytes* bytes from the socket.

Use *LastCount* (p. 1478) to verify the number of bytes actually read.

Use *Error* (p. 1476) to determine if the operation succeeded.

### **Parameters**

*address*

Any address - will be overwritten with the address of the peer that sent that data.

*buffer*

Buffer where to put read data.

*nbytes*

Number of bytes.

### Return value

Returns a reference to the current object, and the address of the peer that sent the data on address param.

### See also

*wxSocketBase::Error* (p. 1476), *wxSocketBase::LastError* (p. 1478),  
*wxSocketBase::LastCount* (p. 1478), *wxSocketBase::SetFlags* (p. 1480),

## wxDatagramSocket::SendTo

**wxDatagramSocket& SendTo(const wxSockAddress& address, const void \* buffer, wxUint32 nbytes)**

This function writes a buffer of *nbytes* bytes to the socket.

Use *LastCount* (p. 1478) to verify the number of bytes actually wrote.

Use *Error* (p. 1476) to determine if the operation succeeded.

### Parameters

*address*

The address of the destination peer for this data.

*buffer*

Buffer where read data is.

*nbytes*

Number of bytes.

### Return value

Returns a reference to the current object.

### See also

*wxSocketBase::Error* (p. 1476), *wxSocketBase::LastError* (p. 1478),  
*wxSocketBase::LastCount* (p. 1478), *wxSocketBase::SetFlags* (p. 1480)

## wxDataInputStream

This class provides functions that read binary data types in a portable way. Data can be read in either big-endian or little-endian format, little-endian being the default on all architectures.

If you want to read data from text files (or streams) use *wxTextInputStream* (p. 1663)

instead.

The >> operator is overloaded and you can use this class like a standard C++ iostream. Note, however, that the arguments are the fixed size types wxUInt32, wxInt32 etc and on a typical 32-bit computer, none of these match to the "long" type (wxInt32 is defined as signed int on 32-bit architectures) so that you cannot use long. To avoid problems (here and elsewhere), make use of the wxInt32, wxUInt32, etc types.

For example:

```
wxFileInputStream input( "mytext.dat" );
wxDataInputStream store( input );
wxUInt8 i1;
float f2;
wxString line;

store >> i1;           // read a 8 bit integer.
store >> i1 >> f2;     // read a 8 bit integer followed by float.
store >> line;         // read a text line
```

See also *wxDataOutputStream* (p. 337).

### Derived from

None

### Include files

<wx/datstrm.h>

## wxDataInputStream::wxDataInputStream

**wxDataInputStream(wxInputStream& stream)**

**wxDataInputStream(wxInputStream& stream, wxMBConv& conv = wxMBConvUTF8)**

Constructs a datastream object from an input stream. Only read methods will be available. The second form is only available in Unicode build of wxWidgets.

### Parameters

*stream*

The input stream.

*conv*

Charset conversion object object used to decode strings in Unicode mode (see *wxDataInputStream::ReadString* (p. 311) documentation for detailed description). Note that you must not destroy *conv* before you destroy this *wxDataInputStream* instance!

**wxDatInputStream::~~wxDatInputStream****~wxDatInputStream()**

Destroys the wxDatInputStream object.

**wxDatInputStream::BigEndianOrdered****void BigEndianOrdered(bool be\_order)**

If *be\_order* is true, all data will be read in big-endian order, such as written by programs on a big endian architecture (e.g. Sparc) or written by Java-Streams (which always use big-endian order).

**wxDatInputStream::Read8****wxUInt8 Read8()**

Reads a single byte from the stream.

**void Read8(wxUInt8 \*buffer, size\_t size)**

Reads bytes from the stream in a specified buffer. The amount of bytes to read is specified by the *size* variable.

**wxDatInputStream::Read16****wxUInt16 Read16()**

Reads a 16 bit unsigned integer from the stream.

**void Read16(wxUInt16 \*buffer, size\_t size)**

Reads 16 bit unsigned integers from the stream in a specified buffer. the amount of 16 bit unsigned integer to read is specified by the *size* variable.

**wxDatInputStream::Read32****wxUInt32 Read32()**

Reads a 32 bit unsigned integer from the stream.

**void Read32(wxUInt32 \*buffer, size\_t size)**

Reads 32 bit unsigned integers from the stream in a specified buffer. the amount of 32 bit unsigned integer to read is specified by the *size* variable.

**wxDatInputStream::Read64****wxUInt64 Read64()**

Reads a 64 bit unsigned integer from the stream.

**void Read64(wxUint64 \*buffer, size\_t size)**

Reads 64 bit unsigned integers from the stream in a specified buffer. the amount of 64 bit unsigned integer to read is specified by the *size* variable.

### **wxDataInputStream::ReadDouble**

**double ReadDouble()**

Reads a double (IEEE encoded) from the stream.

**void ReadDouble(double \*buffer, size\_t size)**

Reads double data (IEEE encoded) from the stream in a specified buffer. the amount of double to read is specified by the *size* variable.

### **wxDataInputStream::ReadString**

**wxString ReadString()**

Reads a string from a stream. Actually, this function first reads a long integer specifying the length of the string (without the last null character) and then reads the string.

In Unicode build of wxWidgets, the function first reads multibyte (char\*) string from the stream and then converts it to Unicode using the *convobject* passed to constructor and returns the result as wxString. You are responsible for using the same convertor as when writing the stream.

See also *wxDataOutputStream::WriteString* (p. 339).

## **wxDataObject**

A wxDataObject represents data that can be copied to or from the clipboard, or dragged and dropped. The important thing about wxDataObject is that this is a 'smart' piece of data unlike 'dumb' data containers such as memory buffers or files. Being 'smart' here means that the data object itself should know what data formats it supports and how to render itself in each of its supported formats.

A supported format, incidentally, is exactly the format in which the data can be requested from a data object or from which the data object may be set. In the general case, an object may support different formats on 'input' and 'output', i.e. it may be able to render itself in a given format but not be created from data on this format or vice versa. wxDataObject defines an enumeration type

```
enum Direction
{
    Get    = 0x01,    // format is supported by GetDataHere()
    Set    = 0x02     // format is supported by SetData()
};
```

which distinguishes between them. See *wxDataFormat* (p. 304) documentation for more about formats.

Not surprisingly, being 'smart' comes at a price of added complexity. This is reasonable for the situations when you really need to support multiple formats, but may be annoying if you only want to do something simple like cut and paste text.

To provide a solution for both cases, `wxWidgets` has two predefined classes which derive from `wxDataObject`: `wxDataObjectSimple` (p. 335) and `wxDataObjectComposite` (p. 334). `wxDataObjectSimple` (p. 335) is the simplest `wxDataObject` possible and only holds data in a single format (such as HTML or text) and `wxDataObjectComposite` (p. 334) is the simplest way to implement a `wxDataObject` that does support multiple formats because it achieves this by simply holding several `wxDataObjectSimple` objects.

So, you have several solutions when you need a `wxDataObject` class (and you need one as soon as you want to transfer data via the clipboard or drag and drop):

- 1. Use one of the built-in classes** You may use `wxTextDataObject`, `wxBitmapDataObject` or `wxFileDataObject` in the simplest cases when you only need to support one format and your data is either text, bitmap or list of files.
- 2. Use `wxDataObjectSimple`** Deriving from `wxDataObjectSimple` is the simplest solution for custom data - you will only support one format and so probably won't be able to communicate with other programs, but data transfer will work in your program (or between different copies of it).
- 3. Use `wxDataObjectComposite`** This is a simple but powerful solution which allows you to support any number of formats (either standard or custom if you combine it with the previous solution).
- 4. Use `wxDataObject` directly** This is the solution for maximal flexibility and efficiency, but it is also the most difficult to implement.

Please note that the easiest way to use drag and drop and the clipboard with multiple formats is by using `wxDataObjectComposite`, but it is not the most efficient one as each `wxDataObjectSimple` would contain the whole data in its respective formats. Now imagine that you want to paste 200 pages of text in your proprietary format, as well as Word, RTF, HTML, Unicode and plain text to the clipboard and even today's computers are in trouble. For this case, you will have to derive from `wxDataObject` directly and make it enumerate its formats and provide the data in the requested format on demand.

Note that neither the GTK+ data transfer mechanisms for clipboard and drag and drop, nor OLE data transfer, copy any data until another application actually requests the data. This is in contrast to the 'feel' offered to the user of a program who would normally think that the data resides in the clipboard after having pressed 'Copy' - in reality it is only declared to be available.

There are several predefined data object classes derived from `wxDataObjectSimple`: `wxFileDataObject` (p. 599), `wxTextDataObject` (p. 1653) and `wxBitmapDataObject` (p. 145) which can be used without change.

You may also derive your own data object classes from `wxCustomDataObject` (p. 302) for user-defined types. The format of user-defined data is given as a mime-type string literal, such as "application/word" or "image/png". These strings are used as they are under Unix



(so far only GTK+) to identify a format and are translated into their Windows equivalent under Win32 (using the OLE IDataObject for data exchange to and from the clipboard and for drag and drop). Note that the format string translation under Windows is not yet finished.

**wxPython note:** At this time this class is not directly usable from wxPython. Derive a class from *wxPyDataObjectSimple* (p. 335) instead.

**wxPerl note:** This class is not currently usable from wxPerl; you may use *Wx::PIDataObjectSimple* (p. 335) instead.

### Virtual functions to override

Each class derived directly from *wxDataObject* must override and implement all of its functions which are pure virtual in the base class.

The data objects which only render their data or only set it (i.e. work in only one direction), should return 0 from *GetFormatCount* (p. 314).

### Derived from

None

### Include files

<wx/dataobj.h>

### See also

*Clipboard and drag and drop overview* (p. 2150), *DnD sample* (p. 2033), *wxFileDataObject* (p. 599), *wxTextDataObject* (p. 1653), *wxBitmapDataObject* (p. 145), *wxCustomDataObject* (p. 302), *wxDropTarget* (p. 561), *wxDropSource* (p. 558), *wxTextDropTarget* (p. 1654), *wxFileDropTarget* (p. 604)

## **wxDataObject::wxDataObject**

**wxDataObject()**

Constructor.

## **wxDataObject::~~wxDataObject**

**~wxDataObject()**

Destructor.

## **wxDataObject::GetAllFormats**

**virtual void GetAllFormats( wxDataFormat \*formats, Direction dir = Get) const**

Copy all supported formats in the given direction to the array pointed to by *formats*. There

is enough space for `GetFormatCount(dir)` formats in it.

**wxPerl note:** In wxPerl this method only takes the `dir` parameter. In scalar context it returns the first format, in list context it returns a list containing all the supported formats.

### **wxDataObject::GetDataHere**

**virtual bool GetDataHere(const wxDataFormat& *format*, void \**buf*) const**

The method will write the data of the format *format* in the buffer *buf* and return true on success, false on failure.

### **wxDataObject::GetDataSize**

**virtual size\_t GetDataSize(const wxDataFormat& *format*) const**

Returns the data size of the given format *format*.

### **wxDataObject::GetFormatCount**

**virtual size\_t GetFormatCount(Direction *dir* = Get) const**

Returns the number of available formats for rendering or setting the data.

### **wxDataObject::GetPreferredFormat**

**virtual wxDataFormat GetPreferredFormat(Direction *dir* = Get) const**

Returns the preferred format for either rendering the data (if *dir* is `Get`, its default value) or for setting it. Usually this will be the native format of the `wxDataObject`.

### **wxDataObject::SetData**

**virtual bool SetData( const wxDataFormat& *format*, size\_t *len*, const void \**buf*)**

Set the data in the format *format* of the length *len* provided in the buffer *buf*.

Returns true on success, false on failure.

## **wxDataViewColumn**

This class represents a column in a `wxDataViewCtrl` (p. 317). One `wxDataViewColumn` is bound to one column in the data model, to which the `wxDataViewCtrl` has been associated.

An instance of `wxDataViewRenderer` (p. 329) is used by this class to render its data.

### **Constants**

These flags define behavi

```
enum wxDataViewColumnFlags
{
    wxDATAVIEW_COL_RESIZABLE = 1,    // the user can resize the column
    wxDATAVIEW_COL_SORTABLE  = 2,    // same as SetSortable()
    wxDATAVIEW_COL_HIDDEN    = 4     // column is hidden
};
```

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/dataview.h>

**wxDataViewColumn::wxDataViewColumn**

**wxDataViewColumn(const wxString& title, wxDataViewRenderer\* renderer,  
unsigned int model\_column, int width = 80, int flags =  
wxDATAVIEW\_COL\_RESIZABLE)**

**wxDataViewColumn(const wxBitmap& bitmap, wxDataViewRenderer\* renderer,  
unsigned int model\_column, int width = 80, int flags =  
wxDATAVIEW\_COL\_RESIZABLE)**

Constructors.

**wxDataViewColumn::~wxDataViewColumn**

**~wxDataViewColumn()**

Destructor.

**wxDataViewColumn::GetBitmap**

**const wxBitmap& GetBitmap()**

Returns the bitmap in the header of the column, if any.

**wxDataViewColumn::GetModelColumn**

**unsigned int GetModelColumn()**

Returns the index of the column of the model, which this wxDataViewColumn is displaying.

**wxDataViewColumn::GetOwner**

**wxDataViewCtrl\* GetOwner()**

Returns the owning *wxDataViewCtrl* (p. 317).

### **wxDataViewColumn::GetRenderer**

**wxDataViewRenderer\* GetRenderer()**

Returns the renderer of this *wxDataViewColumn*.

See also *wxDataViewRenderer* (p. 329).

### **wxDataViewColumn::GetSortable**

**bool GetSortable()**

Returns true if the column is sortable.

See *SetSortable* (p. 317)

### **wxDataViewColumn::GetWidth**

**int GetWidth()**

Returns the width of the column.

### **wxDataViewColumn::IsSortOrderAscending**

**bool IsSortOrderAscending()**

Returns true, if the sort order is ascending.

See also *SetSortOrder* (p. 316)

### **wxDataViewColumn::SetAlignment**

**void SetAlignment(wxAlignment align)**

Set the alignment of the column header.

### **wxDataViewColumn::SetBitmap**

**void SetBitmap(const wxBitmap& bitmap)**

Set the bitmap of the column header.

### **wxDataViewColumn::SetSortOrder**

**void SetSortOrder(bool ascending)**

Indicate the sort order if the implementation of the *wxDataViewCtrl* supports it, most commonly by showing a little arrow. Use this in conjunction with *wxDataViewSortedListModel::SetAscending* (p. 329) to sort the actual data.

**wxDataViewColumn::SetSortable****void SetSortable**(bool *sortable*)

Indicate that the column is sortable. This is only to provide a visual hint in the column (such as a sort order indicator). It will not actually sort the data. Use a *wxDataViewSortedListModel* (p. 328) to do the sorting.

**wxDataViewColumn::SetTitle****void SetTitle**(const wxString& *title*)

Set the title of the column header to *title*.

**wxDataViewCtrl**

This class and its documentation are work in progress and certainly subject to change.

*wxDataViewCtrl* is planned to be a control to display data either in a tree like fashion or in a tabular form or both. Currently, only the tabular form is implemented. *wxDataViewCtrl* doesn't get its data from the user through virtual functions or events, instead you need to write your own *wxDataViewListModel* (p. 325) and associate it with this control. Then you need to add a number of *wxDataViewColumn* (p. 314) to this control to define what each column shall display. Each *wxDataViewColumn* in turn owns 1 instance of a *wxDataViewRenderer* (p. 329) to render its cells. A number of standard renderers for rendering text, dates, images, toggle, a progress bar etc. are provided. Additionally, the user can write custom renderers deriving from *wxDataViewCustomRenderer* (p. 333) for displaying anything.

All data transfer from the control to the model and the user code is done through *wxVariant* (p. 1769) which can be extended to support more data formats as necessary. Accordingly, all type information uses the strings returned from *wxVariant::GetType* (p. 1774).

So far, this control has only been implemented for GTK+ and there are only barely working stubs for a generic implementation. It is planned to implement the control natively under OS X and use generic code under Windows (and elsewhere).

**Window styles****wxDV\_SINGLE**                      Single selection mode. This is the default.**wxDV\_MULTIPLE**                  Multiple selection mode.**Event handling**

To process input from a dataview control, use the following event handler macros to direct input to member functions that take a *wxDataViewEvent* (p. 321) argument.

**EVT\_DATAVIEW\_ROW\_SELECTED(id, func)**      Processes a  
wx.EVT\_COMMAND\_DATAVIEW\_ROW\_SELECTED event.

**EVT\_DATAVIEW\_ROW\_ACTIVATED(id, func)** Processes a `wxEVT_COMMAND_DATAVIEW_ROW_ACTIVATED` event.

**EVT\_DATAVIEW\_COLUMN\_HEADER\_CLICK(id, func)** Processes a `wxEVT_COMMAND_DATAVIEW_COLUMN_HEADER_CLICKED` event.

**EVT\_DATAVIEW\_COLUMN\_HEADER\_RIGHT\_CLICK(id, func)** Processes a `wxEVT_COMMAND_DATAVIEW_COLUMN_HEADER_RIGHT_CLICKED` event.

#### Derived from

`wxControl` (p. 285)

#### Include files

`<wx/dataview.h>`

#### **`wxDataViewCtrl::wxDataViewCtrl`**

**`wxDataViewCtrl()`**

**`wxDataViewCtrl(wxWindow* parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0, const wxValidator& validator = wxDefaultValidator)`**

Constructor. Calls `Create` (p. 319).

#### **`wxDataViewCtrl::~~wxDataViewCtrl`**

**`~wxDataViewCtrl()`**

Destructor.

#### **`wxDataViewCtrl::AppendColumn`**

**`virtual bool AppendColumn(wxDataViewColumn* col)`**

Add a `wxDataViewColumn` (p. 314) to the control. Note that there is a number of short cut methods which implicitly create a `wxDataViewColumn` (p. 314) and a `wxDataViewRenderer` (p. 329) for it (see below).

#### **`wxDataViewCtrl::AppendBitmapColumn`**

**`bool AppendBitmapColumn(const wxString& label, unsigned int model_column, wxDataViewCellMode mode = wxDATAVIEW_CELL_INERT, int width = -1)`**

**`bool AppendBitmapColumn(const wxBitmap& label, unsigned int model_column, wxDataViewCellMode mode = wxDATAVIEW_CELL_INERT, int width = -1)`**

Appends a column for rendering a bitmap.

#### **wxDataViewCtrl::AppendDateColumn**

```
bool AppendDateColumn(const wxString& label, unsigned int model_column,  
wxDataViewCellMode mode = wxDATAVIEW_CELL_ACTIVATABLE, int width = -1)
```

```
bool AppendDateColumn(const wxBitmap& label, unsigned int model_column,  
wxDataViewCellMode mode = wxDATAVIEW_CELL_ACTIVATABLE, int width = -1)
```

Appends a column for rendering a date.

#### **wxDataViewCtrl::AppendProgressColumn**

```
bool AppendProgressColumn(const wxString& label, unsigned int model_column,  
wxDataViewCellMode mode = wxDATAVIEW_CELL_INERT, int width = 80)
```

```
bool AppendProgressColumn(const wxBitmap& label, unsigned int model_column,  
wxDataViewCellMode mode = wxDATAVIEW_CELL_INERT, int width = 80)
```

Appends a column for rendering a progress indicator.

#### **wxDataViewCtrl::AppendTextColumn**

```
bool AppendTextColumn(const wxString& label, unsigned int model_column,  
wxDataViewCellMode mode = wxDATAVIEW_CELL_INERT, int width = -1)
```

```
bool AppendTextColumn(const wxBitmap& label, unsigned int model_column,  
wxDataViewCellMode mode = wxDATAVIEW_CELL_INERT, int width = -1)
```

Appends a column for rendering text.

#### **wxDataViewCtrl::AppendToggleColumn**

```
bool AppendToggleColumn(const wxString& label, unsigned int model_column,  
wxDataViewCellMode mode = wxDATAVIEW_CELL_INERT, int width = 30)
```

```
bool AppendToggleColumn(const wxBitmap& label, unsigned int model_column,  
wxDataViewCellMode mode = wxDATAVIEW_CELL_INERT, int width = 30)
```

Appends a column for rendering a toggle.

#### **wxDataViewCtrl::AssociateModel**

```
virtual bool AssociateModel(wxDataViewListModel* model)
```

Associates a *wxDataViewListModel* (p. 325) with the control. In the future this should be changed to supporting any data model including a to-be-written *wxDataViewTreeModel*.

#### **wxDataViewCtrl::Create**

**bool Create(wxWindow\* parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0, const wxValidator& validator = wxDefaultValidator)**

Create the control. Useful for two step creation.

### **wxDataViewCtrl::ClearColumns**

**virtual bool ClearColumns()**

Removes all columns.

### **wxDataViewCtrl::ClearSelection**

**void ClearSelection()**

Unselects all rows.

### **wxDataViewCtrl::DeleteColumn**

**virtual bool DeleteColumn(unsigned int pos)**

Deletes given column.

### **wxDataViewCtrl::GetColumn**

**virtual wxDataViewColumn\* GetColumn(unsigned int pos)**

Returns pointer to the column.

### **wxDataViewCtrl::GetModel**

**virtual wxDataViewListModel\* GetModel()**

Returns pointer to the data model associated with the control (if any).

### **wxDataViewCtrl::GetNumberOfColumns**

**virtual unsigned int GetNumberOfColumns()**

Returns the number of columns.

### **wxDataViewCtrl::GetSelection**

**virtual int GetSelection() const**

Returns the index of the currently selected row. It will return -1 if no row is selected and the index of the first selected row if more than one row are selected.

### **wxDataViewCtrl::GetSelections**



**virtual int GetSelections(wxArrayInt& aSelections) const**

Returns the number of selected rows and fills an array of int with the indices of the selected rows.

**wxDataViewCtrl::IsSelected**

**virtual bool IsSelected(unsigned int row) const**

Returns *true* if the row is selected, *false* otherwise.

**wxDataViewCtrl::SetSelection**

**virtual void SetSelection(int row)**

Sets the selection. Use -1 to unselect all rows.

**wxDataViewCtrl::SetSelectionRange**

**virtual void SetSelectionRange(unsigned int from, unsigned int to)**

Set a range of selection.

**wxDataViewCtrl::SetSelections**

**virtual void SetSelections(const wxArrayInt& aSelections)**

Set the selection to the array of int.

**wxDataViewCtrl::Unselect**

**virtual void Unselect(unsigned int row)**

Unselect a particular row.

## **wxDataViewEvent**

wxDataViewEvent - the event class for the wxDataViewCtrl notifications

**Derived from**

*wxNotifyEvent* (p. 1146)

**Include files**

<wx/dataview.h>

**wxDataViewEvent::wxDataViewEvent**

**wxDataViewEvent(wxEventType commandType = wxEVT\_NULL, int winid = 0)**

**wxDataViewEvent(const wxDataViewEvent& event)**

**wxDataViewEvent::Clone**

**wxEvent\* Clone() const**

**wxDataViewEvent::GetColumn**

**int GetColumn() const**

**wxDataViewEvent::GetDataViewColumn**

**wxDataViewColumn\* GetDataViewColumn()**

**wxDataViewEvent::GetModel**

**wxDataViewModel\* GetModel() const**

**wxDataViewEvent::GetRow**

**int GetRow() const**

**wxDataViewEvent::GetValue**

**const wxVariant& GetValue() const**

**wxDataViewEvent::IsEditCancelled**

**bool IsEditCancelled() const**

Was cell editing canceled? For wxEVT\_COMMAND\_DATAVIEW\_END\_CELL\_EDIT only.

**wxDataViewEvent::SetColumn**

**void SetColumn(int col)**

**wxDataViewEvent::SetDataViewColumn**

**void SetDataViewColumn(wxDataViewColumn\* col)**

For wxEVT\_DATAVIEW\_COLUMN\_HEADER\_CLICKED only.

**wxDataViewEvent::SetEditCanceled**

**void SetEditCanceled(bool editCancelled)**

**wxDataViewEvent::SetModel**

**void SetModel(wxDataViewModel\* model)**

**wxDataViewEvent::SetRow**

**void SetRow(int row)**

**wxDataViewEvent::SetValue**

**void SetValue(const wxVariant& value)**

## **wxDataViewListModelNotifier**

A *wxDataViewListModelNotifier* instance is owned by a *wxDataViewListModel* (p. 325) and mostly mirrors its interface. See the documentation of that class for further information.

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/dataview.h>

**wxDataViewListModelNotifier::wxDataViewListModelNotifier**

**wxDataViewListModelNotifier()**

Constructor.

**wxDataViewListModelNotifier::~~wxDataViewListModelNotifier**

**~wxDataViewListModelNotifier()**

Destructor.

**wxDataViewListModelNotifier::Cleared**

**bool Cleared()**

Called by owning list model.

**wxDataViewListModelNotifier::GetOwner**

**wxDataViewListModel\* GetOwner()**

Returns the owning *wxDataViewListModel* (p. 325).

**wxDataViewListModelNotifier::RowAppended**

**bool RowAppended()**

Called by owning list model.

**wxDataViewListModelNotifier::RowChanged****bool RowChanged(unsigned int *row*)**

Called by owning list model.

**wxDataViewListModelNotifier::RowDeleted****bool RowDeleted(unsigned int *row*)**

Called by owning list model.

**wxDataViewListModelNotifier::RowInserted****bool RowInserted(unsigned int *before*)**

Called by owning list model.

**wxDataViewListModelNotifier::RowPrepended****bool RowPrepended()**

Called by owning list model.

**wxDataViewListModelNotifier::RowsReordered****bool RowsReordered(unsigned int\* *new\_order*)**

Called by owning list model.

**wxDataViewListModelNotifier::SetOwner****void SetOwner(wxDataViewListModel\* *owner*)**

Set the owning *wxDataViewListModel* (p. 325). This method is called by the *wxDataViewListModel::AddNotifier* (p. 326).

**wxDataViewListModelNotifier::ValueChanged****bool ValueChanged(unsigned int *col*, unsigned int *row*)**

Called by owning list model.

**wxDataViewModel**

`wxDataViewModel` is the base class for all data model to be displayed by a `wxDataViewCtrl` (p. 317). Currently, this class has no functionality at all and the only existing implementation of it is the `wxDataViewListModel` (p. 325). The plan is to move all functionality of `wxDataViewListModel` and a to-be-written `wxDataViewTreeModel` into `wxDataViewModel` and make `wxDataViewListModel` and `wxDataViewTreeModel` subsets of the abstract `wxDataViewModel`.

#### Derived from

`wxObject` (p. 1148)

#### Include files

<wx/dataview.h>

### **wxDataViewModel::wxDataViewModel**

**wxDataViewModel()**

Constructor.

### **wxDataViewModel::~~wxDataViewModel**

**~wxDataViewModel()**

Destructor.

## **wxDataViewListModel**

`wxDataViewListModel` is currently the only existing variant of a `wxDataViewModel` (p. 324). It allows to define a table like data model to be displayed by a `wxDataViewCtrl` (p. 317). You need to derive from this class to define your own data model.

You need to override `GetNumberOfRows` (p. 327), `GetNumberOfCols` (p. 327), `GetColType` (p. 327) and `GetValue` (p. 327) in order to define the data model (which acts as an interface between your actual data and the `wxDataViewCtrl`). Since you will usually also allow the `wxDataViewCtrl` to change your data through its graphical interface, you will also have to override `SetValue` (p. 328) which the `wxDataViewCtrl` will call when a change to some data has been committed.

The data that is presented through this data model is expected to change at run-time. You need to inform the data model that a change happens. Depending on what happened you need to call one of the following methods: `ValueChanged` (p. 328), `RowChanged` (p. 327), `RowAppended` (p. 327), `RowPrepended` (p. 328), `RowInserted` (p. 328), `RowDeleted` (p. 328), `RowsReordered` (p. 328) or `RowsCleared` (p. 326).

`wxDataViewModel` and this class (as indeed the entire `wxDataViewCtrl` code) is using `wxVariant` (p. 1769) to store data and its type in a generic way. `wxVariant` can be extended to contain almost any data without changes to the original class.

This class maintains a list of `wxDataListViewListModelNotifier` (p. 323) which link this class

to the specific implementations on the supported platforms so that e.g. calling *ValueChanged* (p. 328) on this model will just call *wxDataListViewListModelNotifier::ValueChanged* (p. 324) for each notifier that has been added. This is used both for informing the native controls to redraw themselves and for informing e.g. the *wxDataViewSortedListModel* (p. 328) to resort itself. You can also add your own notifier in order to get informed about any changes to the data in the list model.

Additionally, this class maintains a list of all *wxDataViewColumns* (p. 314) which display a certain column of this list model. This is mostly used internally.

### Derived from

*wxDataViewModel* (p. 324)  
*wxObject* (p. 1148)

### Include files

<wx/dataview.h>

### **wxDataViewListModel::wxDataViewListModel**

**wxDataViewListModel()**

Constructor.

### **wxDataViewListModel::~~wxDataViewListModel**

**~wxDataViewListModel()**

Destructor.

### **wxDataViewListModel::AddNotifier**

**void AddNotifier(wxDataViewListModelNotifier\* notifier)**

Adds *notifier* to the internal list of notifiers.

See also *RemoveNotifier* (p. 327).

### **wxDataViewListModel::AddViewingColumn**

**void AddViewingColumn(wxDataViewColumn\* view\_column, unsigned int model\_column)**

Used internally. Used for maintaining a list of *wxDataViewColumn* (p. 314) that display a certain column of this model.

### **wxDataViewListModel::Cleared**

**bool virtual Cleared()**

Call this if all data in your model has been cleared.

**wxDataViewListModel::GetColType****virtual wxString GetColType(unsigned int col)**

Override this to indicate what type of data is stored in the column specified by *col*. This should return a string indicating the type of data as reported by *wxVariant* (p. 1769).

**wxDataViewListModel::GetNumberOfCols****virtual unsigned int GetNumberOfCols()**

Override this to indicate, how many columns the list model has.

**wxDataViewListModel::GetNumberOfRows****virtual unsigned int GetNumberOfRows()**

Override this to indicate, how many rows the list model has.

**wxDataViewListModel::GetValue****virtual void GetValue(wxVariant& variant, unsigned int col, unsigned int row)**

Override this to indicate the value of a given value in the list model. A *wxVariant* (p. 1769) is used to store the data.

**wxDataViewListModel::RemoveNotifier****void RemoveNotifier(wxDataViewListModelNotifier\* notifier)**

Removes the notifier from the list of notifiers.

See also *AddNotifier* (p. 326).

**wxDataViewListModel::RemoveViewingColumn****void RemoveViewingColumn(wxDataViewColumn\* column)**

Used internally. Used for maintaining a list of *wxDataViewColumn* (p. 314) that display a certain column of this model.

**wxDataViewListModel::RowAppended****virtual bool RowAppended()**

Call this if a row has been appended to the list model.

**wxDataViewListModel::RowChanged****virtual bool RowChanged(unsigned int row)**

Call this if the values of this row have been changed.

#### **wxDataViewListModel::RowDeleted**

**virtual bool RowDeleted(unsigned int *row*)**

Call this if this row has been deleted.

#### **wxDataViewListModel::RowInserted**

**virtual bool RowInserted(unsigned int *before*)**

Call this if a row has been inserted.

#### **wxDataViewListModel::RowPrepended**

**virtual bool RowPrepended()**

Call this if a row has been prepended.

#### **wxDataViewListModel::RowsReordered**

**virtual bool RowsReordered(unsigned int\* *new\_order*)**

Call this if the rows have been reordered.

#### **wxDataViewListModel::SetValue**

**virtual bool SetValue(wxVariant& *variant*, unsigned int *col*, unsigned int *row*)**

This method gets called by e.g. the `wxDataViewCtrl` class if a value has been changed through its graphical interface. You need to override this method in order to update the data in the underlying data structure. Afterwards, *ValueChanged* (p. 328) is called.

#### **wxDataViewListModel::ValueChanged**

**virtual bool ValueChanged(unsigned int *col*, unsigned int *row*)**

Call this if a value in the model has been changed.

### **wxDataViewSortedListModel**

This class is used for sorting data. It does not contain any data itself. Rather, it provides a sorted interface for another list model.

Currently, the sorting algorithm isn't thread safe. This needs to be fixed.

#### **Derived from**

*wxDataViewListModel* (p. 325)



*wxDataViewModel* (p. 324)

*wxObject* (p. 1148)

### Include files

<wx/dataview.h>

### **wxDataViewSortedListModel::wxDataViewSortedListModel**

**wxDataViewSortedListModel**(*wxDataViewListModel\* child*)

Constructor. *child* is the child data model the data of which this model is supposed to present in a sorted way.

### **wxDataViewSortedListModel::~~wxDataViewSortedListModel**

**~wxDataViewSortedListModel**()

Destructor.

### **wxDataViewSortedListModel::GetAscending**

**bool** **GetAscending**()

Returns true if the data is sorted in ascending order.

### **wxDataViewSortedListModel::Resort**

**void** **Resort**()

Tell the model to resort its data.

### **wxDataViewSortedListModel::SetAscending**

**void** **SetAscending**(**bool** *ascending*)

Set the sort order of the data.

## **wxDataViewRenderer**

This class is used by *wxDataViewCtrl* (p. 317) to render the individual cells. One instance of a renderer class is owned by *wxDataViewColumn* (p. 314). There is a number of ready-to-use renderers provided: *wxDataViewTextRenderer* (p. 331), *wxDataViewToggleRenderer* (p. 332), *wxDataViewProgressRenderer* (p. 331), *wxDataViewBitmapRenderer* (p. 332), *wxDataViewDateRenderer* (p. 332).

Additionally, the user can write own renderers by deriving from *wxDataViewCustomRenderer* (p. 333).

These flags control the behaviour of the renderer and they are used for controlling in what

mode the renderer shall render its contents:

```
enum wxDataViewCellMode
{
    wxDATAVIEW_CELL_INERT,
    wxDATAVIEW_CELL_ACTIVATABLE,
    wxDATAVIEW_CELL_EDITABLE
};

enum wxDataViewCellRenderState
{
    wxDATAVIEW_CELL_SELECTED      = 1,
    wxDATAVIEW_CELL_PRELIT        = 2,
    wxDATAVIEW_CELL_INSENSITIVE   = 4,
    wxDATAVIEW_CELL_FOCUSED       = 8
};
```

### Derived from

*wxObject* (p. 1148)

### Include files

<wx/dataview.h>

### **wxDataViewRenderer::wxDataViewRenderer**

**wxDataViewRenderer(const wxString& varianttype, wxDataViewCellMode mode = wxDATAVIEW\_CELL\_INERT)**

Constructor.

### **wxDataViewRenderer::GetMode**

**virtual wxDataViewCellMode GetMode()**

Returns the cell mode.

### **wxDataViewRenderer::GetOwner**

**virtual wxDataViewColumn\* GetOwner()**

Returns pointer to the owning *wxDataViewColumn* (p. 314).

### **wxDataViewRenderer::GetValue**

**virtual bool GetValue(wxVariant& value)**

This methods retrieves the value from the renderer in order to transfer the value back to the data model. Returns *false* on failure.

**wxDataViewRenderer::GetVariantType****virtual wxString GetVariantType()**

Returns a string with the type of the *wxVariant* (p. 1769) supported by this renderer.

**wxDataViewRenderer::SetOwner****virtual void SetOwner(wxDataViewColumn\* owner)**

Sets the owning *wxDataViewColumn* (p. 314). This is usually called from within *wxDataViewColumn*.

**wxDataViewRenderer::SetValue****virtual bool SetValue(const wxVariant& value)**

Set the value of the renderer (and thus its cell) to *value*. The internal code will then render this cell with this data.

**wxDataViewRenderer::Validate****virtual bool Validate(wxVariant& value)**

To be implemented.

**wxDataViewTextRenderer***wxDataViewTextRenderer***Derived from***wxDataViewRenderer* (p. 329)**Include files**`<wx/dataview.h>`**wxDataViewTextRenderer::wxDataViewTextRenderer****wxDataViewTextRenderer**(const wxString& *varianttype* = wxT("string"),  
**wxDataViewCellMode** *mode* = wxDATAVIEW\_CELL\_INERT)**wxDataViewProgressRenderer***wxDataViewProgressRenderer***Derived from***wxDataViewRenderer* (p. 329)

**Include files**

<wx/dataview.h>

**wxDataViewProgressRenderer::wxDataViewProgressRenderer**

**wxDataViewProgressRenderer**(const wxString& *label* = wxEmptyString, const wxString& *varianttype* = wxT("long"), wxDataViewCellMode *mode* = wxDATAVIEW\_CELL\_INERT)

**wxDataViewToggleRenderer**

wxDataViewToggleRenderer

**Derived from**

*wxDataViewRenderer* (p. 329)

**Include files**

<wx/dataview.h>

**wxDataViewToggleRenderer::wxDataViewToggleRenderer**

**wxDataViewToggleRenderer**(const wxString& *varianttype* = wxT("bool"), wxDataViewCellMode *mode* = wxDATAVIEW\_CELL\_INERT)

**wxDataViewBitmapRenderer**

wxDataViewBitmapRenderer

**Derived from**

*wxDataViewRenderer* (p. 329)

**Include files**

<wx/dataview.h>

**wxDataViewBitmapRenderer::wxDataViewBitmapRenderer**

**wxDataViewBitmapRenderer**(const wxString& *varianttype* = wxT("wxBitmap"), wxDataViewCellMode *mode* = wxDATAVIEW\_CELL\_INERT)

**wxDataViewDateRenderer**

wxDataViewDateRenderer

**Derived from**

*wxDataViewRenderer* (p. 329)

#### **Include files**

<wx/dataview.h>

#### **wxDataViewDateRenderer::wxDataViewDateRenderer**

**wxDataViewDateRenderer**(const wxString& *varianttype* = wxT("datetime"),  
**wxDataViewCellMode** *mode* = wxDATAVIEW\_CELL\_ACTIVATABLE)

### **wxDataViewCustomRenderer**

wxDataViewCustomRenderer

#### **Derived from**

*wxDataViewRenderer* (p. 329)

#### **Include files**

<wx/dataview.h>

#### **wxDataViewCustomRenderer::wxDataViewCustomRenderer**

**wxDataViewCustomRenderer**(const wxString& *varianttype* = wxT("string"),  
**wxDataViewCellMode** *mode* = wxDATAVIEW\_CELL\_INERT, bool *no\_init* = false)

Constructor.

#### **wxDataViewCustomRenderer::~~wxDataViewCustomRenderer**

**~wxDataViewCustomRenderer**()

Destructor.

#### **wxDataViewCustomRenderer::Activate**

**virtual bool Activate**(wxRect *cell*, wxDataViewListModel\* *model*, unsigned int *col*,  
unsigned int *row*)

Override this to react to double clicks or <ENTER>.

#### **wxDataViewCustomRenderer::GetDC**

**virtual wxDC\* GetDC**()

Create DC on request. Internal.

#### **wxDataViewCustomRenderer::GetSize**

**virtual wxSize GetSize()**

Return size required to show content.

**wxDataViewCustomRenderer::LeftClick**

**virtual bool LeftClick**(wxPoint *cursor*, wxRect *cell*, wxDataViewListModel\* *model*, unsigned int *col*, unsigned int *row*)

Override this to react to a left click.

**wxDataViewCustomRenderer::Render**

**virtual bool Render**(wxRect *cell*, wxDC\* *dc*, int *state*)

Override this to render the cell. Before this is called, *SetValue* (p. 331) was called so that this instance knows what to render.

**wxDataViewCustomRenderer::RightClick**

**virtual bool RightClick**(wxPoint *cursor*, wxRect *cell*, wxDataViewListModel\* *model*, unsigned int *col*, unsigned int *row*)

Override this to react to a right click.

**wxDataViewCustomRenderer::StartDrag**

**virtual bool StartDrag**(wxPoint *cursor*, wxRect *cell*, wxDataViewListModel\* *model*, unsigned int *col*, unsigned int *row*)

Override this to start a drag operation.

**wxDataObjectComposite**

*wxDataObjectComposite* is the simplest *wxDataObject* (p. 311) derivation which may be used to support multiple formats. It contains several *wxDataObjectSimple* (p. 335) objects and supports any format supported by at least one of them. Only one of these data objects is *preferred* (the first one if not explicitly changed by using the second parameter of *Add* (p. 335)) and its format determines the preferred format of the composite data object as well.

See *wxDataObject* (p. 311) documentation for the reasons why you might prefer to use *wxDataObject* directly instead of *wxDataObjectComposite* for efficiency reasons.

**Virtual functions to override**

None, this class should be used directly.

**Derived from**

*wxDataObject* (p. 311)

**Include files**

<wx/dataobj.h>

**See also**

*Clipboard and drag and drop overview* (p. 2150), *wxDataObject* (p. 311), *wxDataObjectSimple* (p. 335), *wxFileDataObject* (p. 599), *wxTextDataObject* (p. 1653), *wxBitmapDataObject* (p. 145)

**wxDataObjectComposite::wxDataObjectComposite**

**wxDataObjectComposite()**

The default constructor.

**wxDataObjectComposite::Add**

**void Add( wxDataObjectSimple \*dataObject, bool preferred = false)**

Adds the *dataObject* to the list of supported objects and it becomes the preferred object if *preferred* is true.

**wxDataObjectComposite::GetReceivedFormat**

**wxDataFormat GetReceivedFormat() const**

Report the format passed to the *SetData* method. This should be the format of the data object within the composite that received data from the clipboard or the DnD operation. You can use this method to find out what kind of data object was received.

**wxDataObjectSimple**

This is the simplest possible implementation of the *wxDataObject* (p. 311) class. The data object of (a class derived from) this class only supports one format, so the number of virtual functions to be implemented is reduced.

Notice that this is still an abstract base class and cannot be used but should be derived from.

**wxPython note:** If you wish to create a derived *wxDataObjectSimple* class in wxPython you should derive the class from *wxPyDataObjectSimple* in order to get Python-aware capabilities for the various virtual methods.

**wxPerl note:** In wxPerl, you need to derive your data object class from *Wx::PIDataObjectSimple*.

**Virtual functions to override**

The objects supporting rendering the data must override *GetDataSize* (p. 336) and *GetDataHere* (p. 336) while the objects which may be set must override *SetData* (p. 337). Of course, the objects supporting both operations must override all three methods.

**Derived from**

*wxDataObject* (p. 311)

**Include files**

<wx/dataobj.h>

**See also**

*Clipboard and drag and drop overview* (p. 2150), *DnD sample* (p. 2033), *wxFileDataObject* (p. 599), *wxTextDataObject* (p. 1653), *wxBitmapDataObject* (p. 145)

**wxDataObjectSimple::wxDataObjectSimple**

**wxDataObjectSimple(const wxDataFormat& format = wxFormatInvalid)**

Constructor accepts the supported format (none by default) which may also be set later with *SetFormat* (p. 336).

**wxDataObjectSimple::GetFormat**

**const wxDataFormat& GetFormat() const**

Returns the (one and only one) format supported by this object. It is supposed that the format is supported in both directions.

**wxDataObjectSimple::SetFormat**

**void SetFormat(const wxDataFormat& format)**

Sets the supported format.

**wxDataObjectSimple::GetDataSize**

**virtual size\_t GetDataSize() const**

Gets the size of our data. Must be implemented in the derived class if the object supports rendering its data.

**wxDataObjectSimple::GetDataHere**

**virtual bool GetDataHere(void \*buf) const**

Copy the data to the buffer, return true on success. Must be implemented in the derived class if the object supports rendering its data.



**wxPython note:** When implementing this method in wxPython, no additional parameters are required and the data should be returned from the method as a string.

### **wxDataObjectSimple::SetData**

**virtual bool SetData**(size\_t *len*, const void \**buf*)

Copy the data from the buffer, return true on success. Must be implemented in the derived class if the object supports setting its data.

**wxPython note:** When implementing this method in wxPython, the data comes as a single string parameter rather than the two shown here.

## **wxDataOutputStream**

This class provides functions that write binary data types in a portable way. Data can be written in either big-endian or little-endian format, little-endian being the default on all architectures.

If you want to write data to text files (or streams) use *wxTextOutputStream* (p. 1666) instead.

The << operator is overloaded and you can use this class like a standard C++ ostream. See *wxDataInputStream* (p. 308) for its usage and caveats.

See also *wxDataInputStream* (p. 308).

### **Derived from**

None

### **Include files**

<wx/datstrm.h>

### **wxDataOutputStream::wxDataOutputStream**

**wxDataOutputStream**(wxOutputStream& *stream*)

**wxDataOutputStream**(wxOutputStream& *stream*, wxMBConv& *conv* = wxMBConvUTF8)

Constructs a datastream object from an output stream. Only write methods will be available. The second form is only available in Unicode build of wxWidgets.

### **Parameters**

*stream*

The output stream.

*conv*

Charset conversion object object used to encoding Unicode strings before writing them to the stream in Unicode mode (see *wxDataOutputStream::WriteString* (p. 339) documentation for detailed description). Note that you must not destroy *conv* before you destroy this *wxDataOutputStream* instance! It is recommended to use default value (UTF-8).

### **wxDataOutputStream::~~wxDataOutputStream**

**~wxDataOutputStream()**

Destroys the *wxDataOutputStream* object.

### **wxDataOutputStream::BigEndianOrdered**

**void BigEndianOrdered(bool *be\_order*)**

If *be\_order* is true, all data will be written in big-endian order, e.g. for reading on a Sparc or from Java-Streams (which always use big-endian order), otherwise data will be written in little-endian order.

### **wxDataOutputStream::Write8**

**void Write8(wxUInt8 *i8*)**

Writes the single byte *i8* to the stream.

**void Write8(const wxUInt8 \**buffer*, size\_t *size*)**

Writes an array of bytes to the stream. The amount of bytes to write is specified with the *size* variable.

### **wxDataOutputStream::Write16**

**void Write16(wxUInt16 *i16*)**

Writes the 16 bit unsigned integer *i16* to the stream.

**void Write16(const wxUInt16 \**buffer*, size\_t *size*)**

Writes an array of 16 bit unsigned integer to the stream. The amount of 16 bit unsigned integer to write is specified with the *size* variable.

### **wxDataOutputStream::Write32**

**void Write32(wxUInt32 *i32*)**

Writes the 32 bit unsigned integer *i32* to the stream.

**void Write32(const wxUInt32 \**buffer*, size\_t *size*)**

Writes an array of 32 bit unsigned integer to the stream. The amount of 32 bit unsigned integer to write is specified with the *size* variable.

### **wxDataOutputStream::Write64**

**void Write64(wxUint64 i64)**

Writes the 64 bit unsigned integer *i64* to the stream.

**void Write64(const wxUint64 \*buffer, size\_t size)**

Writes an array of 64 bit unsigned integer to the stream. The amount of 64 bit unsigned integer to write is specified with the *size* variable.

### **wxDataOutputStream::WriteDouble**

**void WriteDouble(double f)**

Writes the double *f* to the stream using the IEEE format.

**void WriteDouble(const double \*buffer, size\_t size)**

Writes an array of double to the stream. The amount of double to write is specified with the *size* variable.

### **wxDataOutputStream::WriteString**

**void WriteString(const wxString&string)**

Writes *string* to the stream. Actually, this method writes the size of the string before writing *string* itself.

In ANSI build of wxWidgets, the string is written to the stream in exactly same way it is represented in memory. In Unicode build, however, the string is first converted to multibyte representation with *conv* object passed to stream's constructor (consequently, ANSI application can read data written by Unicode application, as long as they agree on encoding) and this representation is written to the stream. UTF-8 is used by default.

## **wxDateEvent**

This event class holds information about a date change and is used together with *wxDatePickerCtrl* (p. 340). It also serves as a base class for *wxCalendarEvent* (p. 176).

### **Derived from**

*wxCommandEvent* (p. 250)

*wxEvent* (p. 572)

*wxObject* (p. 1148)

### **Include files**

<wx/dateevt.h>

### **wxDateEvent::GetDate**

**const wxDateTime& GetDate() const**

Returns the date.

### **wxDateEvent::SetDate**

**void SetDate(const wxDateTime& date)**

Sets the date carried by the event, normally only used by the library internally.

## **wxDatePickerCtrl**

This control allows the user to select a date. Unlike *wxCalendarCtrl* (p. 168), which is a relatively big control, *wxDatePickerCtrl* is implemented as a small window showing the currently selected date. The control can be edited using the keyboard, and can also display a popup window for more user-friendly date selection, depending on the styles used and the platform, except PalmOS where date is selected using native dialog.

It is only available if `wxUSE_DATEPICKCTRL` is set to 1.

### **Derived from**

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### **Include files**

<wx/datectrl.h>

(only available if `wxUSE_DATEPICKCTRL` is set to 1)

### **Window styles**

<b>wxDP_SPIN</b>	Creates a control without a month calendar drop down but with spin-control-like arrows to change individual date components. This style is not supported by the generic version.
<b>wxDP_DROPDOWN</b>	Creates a control with a month calendar drop-down part from which the user can select a date.
<b>wxDP_DEFAULT</b>	Creates a control with the style that is best supported for the current platform (currently <code>wxDP_SPIN</code> under Windows and <code>wxDP_DROPDOWN</code> elsewhere).

<b>wxDP_ALLOWNONE</b>	With this style, the control allows the user to not enter any valid date at all. Without it - the default - the control always has some valid date.
<b>wxDP_SHOWCENTURY</b>	Forces display of the century in the default date format. Without this style the century could be displayed, or not, depending on the default date representation in the system.

### Event handling

<b>EVT_DATE_CHANGED(id, func)</b>	This event fires when the user changes the current selection in the control.
-----------------------------------	--

### See also

*wxCalendarCtrl* (p. 168),  
*wxDateEvent* (p. 339)

## **wxDatePickerCtrl::wxDatePickerCtrl**

**wxDatePickerCtrl**(*wxWindow* \*parent, *wxWindowID* id, **const wxDateTime&** dt = *wxDefaultDateTime*, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = *wxDP\_DEFAULT* | *wxDP\_SHOWCENTURY*, **const wxValidator&** validator = *wxDefaultValidator*, **const wxString&** name = "datectrl")

Initializes the object and calls *Create* (p. 341) with all the parameters.

## **wxDatePickerCtrl::Create**

**bool** Create(*wxWindow* \*parent, *wxWindowID* id, **const wxDateTime&** dt = *wxDefaultDateTime*, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = *wxDP\_DEFAULT* | *wxDP\_SHOWCENTURY*, **const wxValidator&** validator = *wxDefaultValidator*, **const wxString&** name = "datectrl")

### Parameters

*parent*

Parent window, must not be non-NULL.

*id*

The identifier for the control.

*dt*

The initial value of the control, if an invalid date (such as the default value) is used, the control is set to today.

*pos*

Initial position.

*size*

Initial size. If left at default value, the control chooses its own best size by using the height approximately equal to a text control and width large enough to show the date string fully.

*style*

The window style, should be left at 0 as there are no special styles for this control in this version.

*validator*

Validator which can be used for additional date checks.

*name*

Control name.

### **Return value**

`true` if the control was successfully created or `false` if creation failed.

### **wxDatePickerCtrl::GetRange**

**bool GetRange(wxDateTime \*dt1, wxDateTime \*dt2) const**

If the control had been previously limited to a range of dates using *SetRange()* (p. 343), returns the lower and upper bounds of this range. If no range is set (or only one of the bounds is set), *dt1* and/or *dt2* are set to be invalid.

### **Parameters**

*dt1*

Pointer to the object which receives the lower range limit or becomes invalid if it is not set. May be `NULL` if the caller is not interested in lower limit

*dt2*

Same as above but for the upper limit

### **Return value**

`false` if no range limits are currently set, `true` if at least one bound is set.

### **wxDatePickerCtrl::GetValue**

**wxDateTime GetValue() const**

Returns the currently selected. If there is no selection or the selection is outside of the current range, an invalid object is returned.

**wxDatePickerCtrl::SetFormat****void SetFormat(const wxChar\* format)**

Please note that this function is only available in the generic version of this control. The native version always uses the current system locale.

Sets the display format for the date in the control. See *wxDateTime* for the meaning of format strings.

**Remarks**

If the format parameter is invalid, the behaviour is undefined.

**wxDatePickerCtrl::SetRange****void SetRange(const wxDateTime& dt1, const wxDateTime& dt2)**

Sets the valid range for the date selection. If *dt1* is valid, it becomes the earliest date (inclusive) accepted by the control. If *dt2* is valid, it becomes the latest possible date.

**Remarks**

If the current value of the control is outside of the newly set range bounds, the behaviour is undefined.

**wxDatePickerCtrl::SetValue****void SetValue(const wxDateTime& dt)**

Changes the current value of the control. The date should be valid and included in the currently selected range, if any.

Calling this method does not result in a date change event.

**wxDateSpan**

This class is a "logical time span" and is useful for implementing program logic for such things as "add one month to the date" which, in general, doesn't mean to add 60\*60\*24\*31 seconds to it, but to take the same date the next month (to understand that this is indeed different consider adding one month to Feb, 15 -- we want to get Mar, 15, of course).

When adding a month to the date, all lesser components (days, hours, ...) won't be changed unless the resulting date would be invalid: for example, Jan 31 + 1 month will be Feb 28, not (non-existing) Feb 31.

Because of this feature, adding and subtracting back again the same *wxDateSpan* will **not**, in general give back the original date: Feb 28 - 1 month will be Jan 28, not Jan 31!

*wxDateSpan* objects can be either positive or negative. They may be multiplied by scalars which multiply all deltas by the scalar: i.e. 2\*(1 month and 1 day) is 2 months and 2 days. They can be added together and with *wxDateTime* (p. 348) or *wxTimeSpan* (p. 1683), but

the type of result is different for each case.

Beware about weeks: if you specify both weeks and days, the total number of days added will be  $7 * \text{weeks} + \text{days}$ ! See also `GetTotalDays()` function.

Equality operators are defined for `wxDateSpans`. Two date spans are equal if and only if they both give the same target date when added to **every** source date. Thus `wxDateSpan::Months(1)` is not equal to `wxDateSpan::Days(30)`, because they don't give the same date when added to 1 Feb. But `wxDateSpan::Days(14)` is equal to `wxDateSpan::Weeks(2)`

Finally, notice that for adding hours, minutes and so on you don't need this class at all: `wxTimeSpan` (p. 1683) will do the job because there are no subtleties associated with those (we don't support leap seconds).

### Derived from

No base class

### Include files

<wx/datetime.h>

### See also

*Date classes overview* (p. 2052), *wxDateTime* (p. 348)

## **wxDateSpan::wxDateSpan**

**wxDateSpan**(int *years* = 0, int *months* = 0, int *weeks* = 0, int *days* = 0)

Constructs the date span object for the given number of years, months, weeks and days. Note that the weeks and days add together if both are given.

## **wxDateSpan::Add**

**wxDateSpan** Add(const **wxDateSpan**& *other*) const

**wxDateSpan**& Add(const **wxDateSpan**& *other*)

**wxDateSpan**& operator+=(const **wxDateSpan**& *other*)

Returns the sum of two date spans. The first version returns a new object, the second and third ones modify this object in place.

## **wxDateSpan::Day**

static **wxDateSpan** Day()

Returns a date span object corresponding to one day.



**See also**

*Days* (p. 345)

**wxDateSpan::Days**

**static wxDateSpan Days(int days)**

Returns a date span object corresponding to the given number of days.

**See also**

*Day* (p. 344)

**wxDateSpan::GetDays**

**int GetDays() const**

Returns the number of days (only, that it not counting the weeks component!) in this date span.

**See also**

*GetTotalDays* (p. 345)

**wxDateSpan::GetMonths**

**int GetMonths() const**

Returns the number of the months (not counting the years) in this date span.

**wxDateSpan::GetTotalDays**

**int GetTotalDays() const**

Returns the combined number of days in this date span, counting both weeks and days. It still doesn't take neither months nor years into the account.

**See also**

*GetWeeks* (p. 345), *GetDays* (p. 345)

**wxDateSpan::GetWeeks**

**int GetWeeks() const**

Returns the number of weeks in this date span.

**See also**

*GetTotalDays* (p. 345)

**wxDateSpan::GetYears****int GetYears() const**

Returns the number of years in this date span.

**wxDateSpan::Month****static wxDateSpan Month()**

Returns a date span object corresponding to one month.

**See also**

*Months* (p. 346)

**wxDateSpan::Months****static wxDateSpan Months(int mon)**

Returns a date span object corresponding to the given number of months.

**See also**

*Month* (p. 346)

**wxDateSpan::Multiply****wxDateSpan Multiply(int factor) const****wxDateSpan& Multiply(int factor)****wxDateSpan& operator\*=(int factor)**

Returns the product of the date span by the specified *factor*. The product is computed by multiplying each of the components by the factor.

The first version returns a new object, the second and third ones modify this object in place.

**wxDateSpan::Negate****wxDateSpan Negate() const**

Returns the date span with the opposite sign.

**See also**

*Neg* (p. 346)

**wxDateSpan::Neg**

**wxDateSpan& Neg()**

**wxDateSpan& operator-()**

Changes the sign of this date span.

**See also**

*Negate* (p. 346)

**wxDateSpan::SetDays**

**wxDateSpan& SetDays(int *n*)**

Sets the number of days (without modifying any other components) in this date span.

**wxDateSpan::SetYears**

**wxDateSpan& SetYears(int *n*)**

Sets the number of years (without modifying any other components) in this date span.

**wxDateSpan::SetMonths**

**wxDateSpan& SetMonths(int *n*)**

Sets the number of months (without modifying any other components) in this date span.

**wxDateSpan::SetWeeks**

**wxDateSpan& SetWeeks(int *n*)**

Sets the number of weeks (without modifying any other components) in this date span.

**wxDateSpan::Subtract**

**wxDateSpan Subtract(const wxDateSpan& *other*) const**

**wxDateSpan& Subtract(const wxDateSpan& *other*)**

**wxDateSpan& operator+=(const wxDateSpan& *other*)**

Returns the difference of two date spans. The first version returns a new object, the second and third ones modify this object in place.

**wxDateSpan::Week**

**static wxDateSpan Week()**

Returns a date span object corresponding to one week.

**See also**

*Weeks* (p. 348)

### **wxDateSpan::Weeks**

**static wxDateSpan Weeks(int weeks)**

Returns a date span object corresponding to the given number of weeks.

#### **See also**

*Week* (p. 347)

### **wxDateSpan::Year**

**static wxDateSpan Year()**

Returns a date span object corresponding to one year.

#### **See also**

*Years* (p. 348)

### **wxDateSpan::Years**

**static wxDateSpan Years(int years)**

Returns a date span object corresponding to the given number of years.

#### **See also**

*Year* (p. 348)

### **wxDateSpan::operator==**

**bool operator==(wxDateSpan& other) const**

Returns `true` if this date span is equal to the other one. Two date spans are considered equal if and only if they have the same number of years and months and the same total number of days (counting both days and weeks).

### **wxDateSpan::operator!=**

**bool operator!=(wxDateSpan& other) const**

Returns `true` if this date span is different from the other one.

#### **See also**

*operator==* (p. 348)

## **wxDateTime**

`wxDateTime` class represents an absolute moment in the time.

## Types

The type `wxDateTime_t` is typedefed as `unsigned short` and is used to contain the number of years, hours, minutes, seconds and milliseconds.

## Constants

Global constant `wxDefaultDateTime` and synonym for it `wxInvalidDateTime` are defined. This constant will be different from any valid `wxDateTime` object.

All the following constants are defined inside `wxDateTime` class (i.e., to refer to them you should prepend their names with `wxDateTime::`).

Time zone symbolic names:

```
enum TZ
{
    // the time in the current time zone
    Local,

    // zones from GMT (= Greenwich Mean Time): they're guaranteed
    // consequent numbers, so writing something like `GMT0 +
    // offset' is
    // safe if abs(offset) <= 12

    // underscore stands for minus
    GMT_12, GMT_11, GMT_10, GMT_9, GMT_8, GMT_7,
    GMT_6, GMT_5, GMT_4, GMT_3, GMT_2, GMT_1,
    GMT0,
    GMT1, GMT2, GMT3, GMT4, GMT5, GMT6,
    GMT7, GMT8, GMT9, GMT10, GMT11, GMT12, GMT13,
    // Note that GMT12 and GMT_12 are not the same: there is a
    // difference
    // of exactly one day between them

    // some symbolic names for TZ

    // Europe
    WET = GMT0, // Western Europe Time
    WEST = GMT1, // Western Europe Summer

    CET = GMT1, // Central Europe Time
    CEST = GMT2, // Central Europe Summer

    EET = GMT2, // Eastern Europe Time
    EEST = GMT3, // Eastern Europe Summer

    MSK = GMT3, // Moscow Time
    MSD = GMT4, // Moscow Summer Time

    // US and Canada
    AST = GMT_4, // Atlantic Standard Time
    ADT = GMT_3, // Atlantic Daylight Time
```

---

```

        EST = GMT_5,                // Eastern Standard Time
        EDT = GMT_4,                // Eastern Daylight Saving
Time
        CST = GMT_6,                // Central Standard Time
        CDT = GMT_5,                // Central Daylight Saving
Time
        MST = GMT_7,                // Mountain Standard Time
        MDT = GMT_6,                // Mountain Daylight Saving
Time
        PST = GMT_8,                // Pacific Standard Time
        PDT = GMT_7,                // Pacific Daylight Saving
Time
        HST = GMT_10,               // Hawaiian Standard Time
        AKST = GMT_9,               // Alaska Standard Time
        AKDT = GMT_8,               // Alaska Daylight Saving
Time

        // Australia

        A_WST = GMT8,                // Western Standard Time
        A_CST = GMT13 + 1,           // Central Standard Time
(+9.5)
        A_EST = GMT10,               // Eastern Standard Time
        A_ESST = GMT11,              // Eastern Summer Time

        // New Zealand
        NZST = GMT12,                // Standard Time
        NZDT = GMT13,                // Daylight Saving Time

        // Universal Coordinated Time = the new and politically
correct name
        // for GMT
        UTC = GMT0
    };

```

Month names: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec and Inv\_Month for an invalid.month value are the values of wxDateTime::Monthenum.

Likewise, Sun, Mon, Tue, Wed, Thu, Fri, Sat, and Inv\_WeekDay are the values in wxDateTime::WeekDay enum.

Finally, Inv\_Year is defined to be an invalid value for year parameter.

*GetMonthName()* (p. 357) and *GetWeekDayName* (p. 358) functions use the following flags:

```

enum NameFlags
{
    Name_Full = 0x01,                // return full name
    Name_Abbr = 0x02                 // return abbreviated name
};

```

Several functions accept an extra parameter specifying the calendar to use (although most of them only support now the Gregorian calendar). This parameters is one of the following values:

```
enum Calendar
{
    Gregorian, // calendar currently in use in Western countries
    Julian      // calendar in use since -45 until the 1582 (or
later)
};
```

Date calculations often depend on the country and `wxDatetime` allows to set the country whose conventions should be used using `SetCountry` (p. 359). It takes one of the following values as parameter:

```
enum Country
{
    Country_Unknown, // no special information for this country
    Country_Default, // set the default country with SetCountry()
method
    // or use the default country with any other

    Country_WesternEurope_Start,
    Country_EEC = Country_WesternEurope_Start,
    France,
    Germany,
    UK,
    Country_WesternEurope_End = UK,

    Russia,

    USA
};
```

Different parts of the world use different conventions for the week start. In some countries, the week starts on Sunday, while in others -- on Monday. The ISO standard doesn't address this issue, so we support both conventions in the functions whose result depends on it (`GetWeekOfYear` (p. 365) and `GetWeekOfMonth` (p. 365)).

The desired behaviour may be specified by giving one of the following constants as argument to these functions:

```
enum WeekFlags
{
    Default_First, // Sunday_First for US, Monday_First for the
rest
    Monday_First, // week starts with a Monday
    Sunday_First  // week starts with a Sunday
};
```

### **Derived from**

No base class

### **Include files**

<wx/datetime.h>

### **See also**

*Date classes overview* (p. 2052), *wxTimeSpan* (p. 1683), *wxDateSpan* (p. 343), *wxCalendarCtrl* (p. 168)

## Static functions

For convenience, all static functions are collected here. These functions either set or return the static variables of *wxDateSpan* (the country), return the current moment, year, month or number of days in it, or do some general calendar-related actions.

Please note that although several function accept an extra *Calendar* parameter, it is currently ignored as only the Gregorian calendar is supported. Future versions will support other calendars.

**wxPython note:** These methods are standalone functions named `wxDateTime_<StaticMethodName>` in wxPython.

*SetCountry* (p. 359)  
*GetCountry* (p. 356)  
*IsWestEuropeanCountry* (p. 359)  
*GetCurrentYear* (p. 357)  
*ConvertYearToBC* (p. 356)  
*GetCurrentMonth* (p. 357)  
*IsLeapYear* (p. 358)  
*GetCentury* (p. 357)  
*GetNumberOfDays* (p. 357)  
*GetNumberOfDays* (p. 357)  
*GetMonthName* (p. 357)  
*GetWeekDayName* (p. 358)  
*GetAmPmStrings* (p. 356)  
*IsDSTApplicable* (p. 359)  
*GetBeginDST* (p. 356)  
*GetEndDST* (p. 357)  
*Now* (p. 359)  
*UNow* (p. 360)  
*Today* (p. 359)

## Constructors, assignment operators and setters

Constructors and various `Set ( )` methods are collected here. If you construct a date object from separate values for day, month and year, you should use *IsValid* (p. 363) method to check that the values were correct as constructors can not return an error code.

*wxDateTime()* (p. 360)  
*wxDateTime(time\_t)* (p. 360)  
*wxDateTime(struct tm)* (p. 360)  
*wxDateTime(double jdn)* (p. 360)  
*wxDateTime(h, m, s, ms)* (p. 361)  
*wxDateTime(day, mon, year, h, m, s, ms)* (p. 361)  
*SetToCurrent* (p. 361)



*Set(time\_t)* (p. 361)  
*Set(struct tm)* (p. 361)  
*Set(double jdn)* (p. 361)  
*Set(h, m, s, ms)* (p. 362)  
*Set(day, mon, year, h, m, s, ms)* (p. 362)  
*SetFromDOS(unsigned long ddt)* (p. 366)  
*ResetTime* (p. 362)  
*SetYear* (p. 362)  
*SetMonth* (p. 362)  
*SetDay* (p. 362)  
*SetHour* (p. 362)  
*SetMinute* (p. 362)  
*SetSecond* (p. 363)  
*SetMillisecond* (p. 363)  
*operator=(time\_t)* (p. 363)  
*operator=(struct tm)* (p. 363)

### Accessors

Here are the trivial accessors. Other functions, which might have to perform some more complicated calculations to find the answer are under the *Calendar calculations* (p. 355) section.

*IsValid* (p. 363)  
*GetTicks* (p. 364)  
*GetCentury* (p. 364)  
*GetYear* (p. 364)  
*GetMonth* (p. 364)  
*GetDay* (p. 364)  
*GetWeekDay* (p. 364)  
*GetHour* (p. 364)  
*GetMinute* (p. 364)  
*GetSecond* (p. 364)  
*GetMillisecond* (p. 365)  
*GetDayOfYear* (p. 365)  
*GetWeekOfYear* (p. 365)  
*GetWeekOfMonth* (p. 365)  
*GetYearDay* (p. 373)  
*IsWorkDay* (p. 365)  
*IsGregorianDate* (p. 365)  
*GetAsDOS* (p. 366)

### Date comparison

There are several function to allow date comparison. To supplement them, a few global operators *>*, *<* etc taking *wxDateTime* are defined.

*IsEqualTo* (p. 366)  
*IsEarlierThan* (p. 366)  
*IsLaterThan* (p. 366)  
*IsStrictlyBetween* (p. 366)

*IsBetween* (p. 366)  
*IsSameDate* (p. 367)  
*IsSameTime* (p. 367)  
*IsEqualUpTo* (p. 367)

## Date arithmetics

These functions carry out *arithmetics* (p. 2054) on the *wxDateTime* objects. As explained in the overview, either *wxTimeSpan* or *wxDateSpan* may be added to *wxDateTime*, hence all functions are overloaded to accept both arguments.

Also, both *Add()* and *Subtract()* have both *const* and *non-const* version. The first one returns a new object which represents the sum/difference of the original one with the argument while the second form modifies the object to which it is applied. The operators *-=* and *+=* are defined to be equivalent to the second forms of these functions.

*Add(wxTimeSpan)* (p. 367)  
*Add(wxDateSpan)* (p. 367)  
*Subtract(wxTimeSpan)* (p. 367)  
*Subtract(wxDateSpan)* (p. 368)  
*Subtract(wxDateTime)* (p. 368)  
*operator+=(wxTimeSpan)* (p. 367)  
*operator+=(wxDateSpan)* (p. 367)  
*operator-=(wxTimeSpan)* (p. 367)  
*operator-=(wxDateSpan)* (p. 368)

## Parsing and formatting dates

These functions convert *wxDateTime* objects to and from text. The conversions to text are mostly trivial: you can either do it using the default date and time representations for the current locale (*FormatDate* (p. 370) and *FormatTime* (p. 370)), using the international standard representation defined by ISO 8601 (*FormatISODate* (p. 370) and *FormatISOTime* (p. 370)) or by specifying any format at all and using *Format* (p. 370) directly.

The conversions from text are more interesting, as there are much more possibilities to care about. The simplest cases can be taken care of with *ParseFormat* (p. 368) which can parse any date in the given (rigid) format. *ParseRfc822Date* (p. 368) is another function for parsing dates in predefined format -- the one of RFC 822 which (still...) defines the format of email messages on the Internet. This format can not be described with *strptime(3)*-like format strings used by *Format* (p. 370), hence the need for a separate function.

But the most interesting functions are *ParseTime* (p. 369), *ParseDate* (p. 369) and *ParseDateTime* (p. 369). They try to parse the date and time (or only one of them) in 'free' format, i.e. allow them to be specified in any of possible ways. These functions will usually be used to parse the (interactive) user input which is not bound to be in any predefined format. As an example, *ParseDateTime* (p. 369) can parse the strings such as "tomorrow", "March first" and even "next Sunday".

*ParseRfc822Date* (p. 368)

*ParseFormat* (p. 368)  
*ParseDateTime* (p. 369)  
*ParseDate* (p. 369)  
*ParseTime* (p. 369)  
*Format* (p. 370)  
*FormatDate* (p. 370)  
*FormatTime* (p. 370)  
*FormatISODate* (p. 370)  
*FormatISOTime* (p. 370)

### **Calendar calculations**

The functions in this section perform the basic calendar calculations, mostly related to the week days. They allow to find the given week day in the week with given number (either in the month or in the year) and so on.

All (non-const) functions in this section don't modify the time part of the *wxDateTime* -- they only work with the date part of it.

*SetToWeekDayInSameWeek* (p. 370)  
*GetWeekDayInSameWeek* (p. 371)  
*SetToNextWeekDay* (p. 371)  
*GetNextWeekDay* (p. 371)  
*SetToPrevWeekDay* (p. 371)  
*GetPrevWeekDay* (p. 371)  
*SetToWeekDay* (p. 371)  
*GetWeekDay* (p. 372)  
*SetToLastWeekDay* (p. 372)  
*GetLastWeekDay* (p. 372)  
*SetToWeekOfYear* (p. 372)  
*SetToLastMonthDay* (p. 372)  
*GetLastMonthDay* (p. 372)  
*SetToYearDay* (p. 373)  
*GetYearDay* (p. 373)

### **Astronomical/historical functions**

Some degree of support for the date units used in astronomy and/or history is provided. You can construct a *wxDateTime* object from a *JDN* (p. 361) and you may also get its *JDN*, *MJD* (p. 373) or *Rata Die number* (p. 373) from it.

*wxDateTime(double jdn)* (p. 360)  
*Set(double jdn)* (p. 361)  
*GetJulianDayNumber* (p. 373)  
*GetJDN* (p. 373)  
*GetModifiedJulianDayNumber* (p. 373)  
*GetMJD* (p. 373)  
*GetRataDie* (p. 373)

### **Time zone and DST support**

Please see the *time zone overview* (p. 2055) for more information about time zones. Normally, these functions should be rarely used.

*FromTimezone* (p. 374)  
*ToTimezone* (p. 374)  
*MakeTimezone* (p. 374)  
*MakeFromTimezone* (p. 374)  
*ToUTC* (p. 374)  
*MakeUTC* (p. 374)  
*GetBeginDST* (p. 356)  
*GetEndDST* (p. 357)  
*IsDST* (p. 374)

### **wxDateTime::ConvertYearToBC**

**static int ConvertYearToBC(int year)**

Converts the year in absolute notation (i.e. a number which can be negative, positive or zero) to the year in BC/AD notation. For the positive years, nothing is done, but the year 0 is year 1 BC and so for other years there is a difference of 1.

This function should be used like this:

```
wxDateTime dt(...);
int y = dt.GetYear();
printf("The year is %d%s", wxDateTime::ConvertYearToBC(y), y > 0 ?
"AD" : "BC");
```

### **wxDateTime::GetAmPmStrings**

**static void GetAmPmStrings(wxString \*am, wxString \*pm)**

Returns the translations of the strings AM and PM used for time formatting for the current locale. Either of the pointers may be NULL if the corresponding value is not needed.

### **wxDateTime::GetBeginDST**

**static wxDateTime GetBeginDST(int year = Inv\_Year, Country country = Country\_Default)**

Get the beginning of DST for the given country in the given year (current one by default). This function suffers from limitations described in *DST overview* (p. 2056).

**See also**

*GetEndDST* (p. 357)

### **wxDateTime::GetCountry**

**static Country GetCountry()**

Returns the current default country. The default country is used for DST calculations, for example.

**See also**

*SetCountry* (p. 359)

**wxDateTime::GetCurrentYear**

**static int GetCurrentYear(Calendar cal = *Gregorian*)**

Get the current year in given calendar (only Gregorian is currently supported).

**wxDateTime::GetCurrentMonth**

**static Month GetCurrentMonth(Calendar cal = *Gregorian*)**

Get the current month in given calendar (only Gregorian is currently supported).

**wxDateTime::GetCentury**

**static int GetCentury(int year)**

Get the current century, i.e. first two digits of the year, in given calendar (only Gregorian is currently supported).

**wxDateTime::GetEndDST**

**static wxDateTime GetEndDST(int year = *Inv\_Year*, Country country = *Country\_Default*)**

Returns the end of DST for the given country in the given year (current one by default).

**See also**

*GetBeginDST* (p. 356)

**wxDateTime::GetMonthName**

**static wxString GetMonthName(Month month, NameFlags flags = *Name\_Full*)**

Gets the full (default) or abbreviated (specify *Name\_Abbbr* name of the given month.

**See also**

*GetWeekDayName* (p. 358)

**wxDateTime::GetNumberOfDays**

**static wxDateTime\_t GetNumberOfDays(int year, Calendar cal = Gregorian)**

**static wxDateTime\_t GetNumberOfDays(Month month, int year = Inv\_Year, Calendar cal = Gregorian)**

Returns the number of days in the given year or in the given month of the year.

The only supported value for *cal* parameter is currently *Gregorian*.

**wxPython note:** These two methods are named *GetNumberOfDaysInYear* and *GetNumberOfDaysInMonth* in wxPython.

### **wxDateTime::GetTimeNow**

**static time\_t GetTimeNow()**

Returns the current time.

### **wxDateTime::GetTmNow**

**static struct tm \* GetTmNow(struct tm \*tm)**

Returns the current time broken down, uses the buffer whose address is passed to the function via *tm* to store the result.

### **wxDateTime::GetTmNow**

**static struct tm \* GetTmNow()**

Returns the current time broken down. Note that this function returns a pointer to a static buffer that's reused by calls to this function and certain C library functions (e.g. *localtime*). If there is any chance your code might be used in a multi-threaded application, you really should use the flavour of function *wxDateTime::GetTmNow* (p. 358) taking a parameter.

### **wxDateTime::GetWeekDayName**

**static wxString GetWeekDayName(WeekDay weekday, NameFlags flags = Name\_Full)**

Gets the full (default) or abbreviated (specify *Name\_Abbbr* name of the given week day.

#### **See also**

*GetMonthName* (p. 357)

### **wxDateTime::IsLeapYear**

**static bool IsLeapYear(int year = Inv\_Year, Calendar cal = Gregorian)**

Returns *true* if the *year* is a leap one in the specified calendar.

This functions supports Gregorian and Julian calendars.

### **wxDateTime::IsWestEuropeanCountry**

**static bool IsWestEuropeanCountry(Country country = Country\_Default)**

This function returns `true` if the specified (or default) country is one of Western European ones. It is used internally by `wxDateTime` to determine the DST convention and date and time formatting rules.

### **wxDateTime::IsDSTApplicable**

**static bool IsDSTApplicable(int year = Inv\_Year, Country country = Country\_Default)**

Returns `true` if DST was used in the given year (the current one by default) in the given country.

### **wxDateTime::Now**

**static wxDateTime Now()**

Returns the object corresponding to the current time.

Example:

```
wxDateTime now = wxDateTime::Now();
printf("Current time in Paris:\t%s\n", now.Format("%c",
wxDateTime::CET).c_str());
```

Note that this function is accurate up to second: `wxDateTime::UNow` (p. 360) should be used for better precision (but it is less efficient and might not be available on all platforms).

### **See also**

*Today* (p. 359)

### **wxDateTime::SetCountry**

**static void SetCountry(Country country)**

Sets the country to use by default. This setting influences the DST calculations, date formatting and other things.

The possible values for *country* parameter are enumerated in *wxDateTime constants section* (p. 348).

### **See also**

*GetCountry* (p. 356)

### **wxDateTime::Today**

**static wxDateTime Today()**

Returns the object corresponding to the midnight of the current day (i.e. the same as *Now()* (p. 359), but the time part is set to 0).

**See also**

*Now* (p. 359)

**wxDateTime::UNow****static wxDateTime UNow()**

Returns the object corresponding to the current time including the milliseconds if a function to get time with such precision is available on the current platform (supported under most Unices and Win32).

**See also**

*Now* (p. 359)

**wxDateTime::wxDateTime****wxDateTime()**

Default constructor. Use one of *Set ( )* functions to initialize the object later.

**wxDateTime::wxDateTime****wxDateTime& wxDateTime(time\_t *timet*)**

Same as *Set* (p. 360).

**wxPython note:** This constructor is named *wxDateTimeFromTimeT* in wxPython.

**wxDateTime::wxDateTime****wxDateTime& wxDateTime(const struct tm& *tm*)**

Same as *Set* (p. 360)

**wxPython note:** Unsupported.

**wxDateTime::wxDateTime****wxDateTime& wxDateTime(double *jdn*)**

Same as *Set* (p. 360)

**wxPython note:** This constructor is named *wxDateTimeFromJDN* in wxPython.



**wxDateTime::wxDateTime**

**wxDateTime& wxDateTime**(wxDateTime\_t *hour*, wxDateTime\_t *minute* = 0, wxDateTime\_t *second* = 0, wxDateTime\_t *millisec* = 0)

Same as *Set* (p. 361)

**wxPython note:** This constructor is named `wxDateTimeFromHMS` in wxPython.

**wxDateTime::wxDateTime**

**wxDateTime& wxDateTime**(wxDateTime\_t *day*, Month *month* = *Inv\_Month*, int *Inv\_Year*, wxDateTime\_t *hour* = 0, wxDateTime\_t *minute* = 0, wxDateTime\_t *second* = 0, wxDateTime\_t *millisec* = 0)

Same as *Set* (p. 362)

**wxPython note:** This constructor is named `wxDateTimeFromDMY` in wxPython.

**wxDateTime::SetToCurrent**

**wxDateTime& SetToCurrent**()

Sets the date and time of to the current values. Same as assigning the result of *Now()* (p. 359) to this object.

**wxDateTime::Set**

**wxDateTime& Set**(time\_t *timet*)

Constructs the object from *timet* value holding the number of seconds since Jan 1, 1970.

**wxPython note:** This method is named `SetTimeT` in wxPython.

**wxDateTime::Set**

**wxDateTime& Set**(const struct tm& *tm*)

Sets the date and time from the broken down representation in the standard `tm` structure.

**wxPython note:** Unsupported.

**wxDateTime::Set**

**wxDateTime& Set**(double *jdn*)

Sets the date from the so-called *Julian Day Number*.

By definition, the Julian Day Number, usually abbreviated as JDN, of a particular instant is the fractional number of days since 12 hours Universal Coordinated Time (Greenwich mean noon) on January 1 of the year -4712 in the Julian proleptic calendar.

**wxPython note:** This method is named `SetJDN` in wxPython.

### **wxDateTime::Set**

**wxDateTime& Set**(wxDateTime\_t *hour*, wxDateTime\_t *minute* = 0, wxDateTime\_t *second* = 0, wxDateTime\_t *millisec* = 0)

Sets the date to be equal to *Today* (p. 359) and the time from supplied parameters.

**wxPython note:** This method is named `SetHMS` in wxPython.

### **wxDateTime::Set**

**wxDateTime& Set**(wxDateTime\_t *day*, Month *month* = *Inv\_Month*, int *year* = *Inv\_Year*, wxDateTime\_t *hour* = 0, wxDateTime\_t *minute* = 0, wxDateTime\_t *second* = 0, wxDateTime\_t *millisec* = 0)

Sets the date and time from the parameters.

### **wxDateTime::ResetTime**

**wxDateTime& ResetTime**()

Reset time to midnight (00:00:00) without changing the date.

### **wxDateTime::SetYear**

**wxDateTime& SetYear**(int *year*)

Sets the year without changing other date components.

### **wxDateTime::SetMonth**

**wxDateTime& SetMonth**(Month *month*)

Sets the month without changing other date components.

### **wxDateTime::SetDay**

**wxDateTime& SetDay**(wxDateTime\_t *day*)

Sets the day without changing other date components.

### **wxDateTime::SetHour**

**wxDateTime& SetHour**(wxDateTime\_t *hour*)

Sets the hour without changing other date components.

### **wxDateTime::SetMinute**

**wxDatetime& SetMinute(wxDatetime\_t minute)**

Sets the minute without changing other date components.

**wxDatetime::SetSecond**

**wxDatetime& SetSecond(wxDatetime\_t second)**

Sets the second without changing other date components.

**wxDatetime::SetMillisecond**

**wxDatetime& SetMillisecond(wxDatetime\_t millisecond)**

Sets the millisecond without changing other date components.

**wxDatetime::operator=**

**wxDatetime& operator(time\_t time)**

Same as *Set* (p. 361).

**wxDatetime::operator=**

**wxDatetime& operator(const struct tm& tm)**

Same as *Set* (p. 361).

**wxDatetime::IsValid**

**bool IsValid() const**

Returns *true* if the object represents a valid time moment.

**wxDatetime::GetDateOnly**

**wxDatetime GetDateOnly() const**

Returns the object having the same date component as this one but time of 00:00:00.

This function is new since wxWidgets version 2.8.2

**See also**

*ResetTime* (p. 362)

**wxDatetime::GetTm**

**Tm GetTm(const TimeZone& tz = Local) const**

Returns broken down representation of the date and time.

**wxDateTime::GetTicks****time\_t GetTicks() const**

Returns the number of seconds since Jan 1, 1970. An assert failure will occur if the date is not in the range covered by `time_t` type.

**wxDateTime::GetCentury****int GetCentury(const TimeZone& tz = Local) const**

Returns the century of this date.

**wxDateTime::GetYear****int GetYear(const TimeZone& tz = Local) const**

Returns the year in the given timezone (local one by default).

**wxDateTime::GetMonth****Month GetMonth(const TimeZone& tz = Local) const**

Returns the month in the given timezone (local one by default).

**wxDateTime::GetDay****wxDateTime\_t GetDay(const TimeZone& tz = Local) const**

Returns the day in the given timezone (local one by default).

**wxDateTime::GetWeekDay****WeekDay GetWeekDay(const TimeZone& tz = Local) const**

Returns the week day in the given timezone (local one by default).

**wxDateTime::GetHour****wxDateTime\_t GetHour(const TimeZone& tz = Local) const**

Returns the hour in the given timezone (local one by default).

**wxDateTime::GetMinute****wxDateTime\_t GetMinute(const TimeZone& tz = Local) const**

Returns the minute in the given timezone (local one by default).

**wxDateTime::GetSecond**

**wxDateTime\_t GetSecond(const TimeZone& tz = Local) const**

Returns the seconds in the given timezone (local one by default).

**wxDateTime::GetMillisecond**

**wxDateTime\_t GetMillisecond(const TimeZone& tz = Local) const**

Returns the milliseconds in the given timezone (local one by default).

**wxDateTime::GetDayOfYear**

**wxDateTime\_t GetDayOfYear(const TimeZone& tz = Local) const**

Returns the day of the year (in 1...366 range) in the given timezone (local one by default).

**wxDateTime::GetWeekOfYear**

**wxDateTime\_t GetWeekOfYear(WeekFlags flags = Monday\_First, const TimeZone& tz = Local) const**

Returns the number of the week of the year this date is in. The first week of the year is, according to international standards, the one containing Jan 4 or, equivalently, the first week which has Thursday in this year. Both of these definitions are the same as saying that the first week of the year must contain more than half of its days in this year. Accordingly, the week number will always be in 1...53 range (52 for non-leap years).

The function depends on the *week start* (p. 348) convention specified by the *flags* argument but its results for `Sunday_First` are not well-defined as the ISO definition quoted above applies to the weeks starting on Monday only.

**wxDateTime::GetWeekOfMonth**

**wxDateTime\_t GetWeekOfMonth(WeekFlags flags = Monday\_First, const TimeZone& tz = Local) const**

Returns the ordinal number of the week in the month (in 1...5 range).

As *GetWeekOfYear* (p. 365), this function supports both conventions for the week start. See the description of these *week start* (p. 348) conventions.

**wxDateTime::IsWorkDay**

**bool IsWorkDay(Country country = Country\_Default) const**

Returns `true` if this day is not a holiday in the given country.

**wxDateTime::IsGregorianDate**

**bool IsGregorianDate(GregorianAdoption country = Gr\_Standard) const**

Returns `true` if the given date is later than the date of adoption of the Gregorian calendar in the given country (and hence the Gregorian calendar calculations make sense for it).

### **wxDatetime::SetFromDOS**

**wxDatetime& Set(unsigned long ddt)**

Sets the date from the date and time inDOS

([http://developer.novell.com/ndk/doc/smscomp/index.html?page=/ndk/doc/smscomp/sms\\_docs/data/hc2vlu5i.html](http://developer.novell.com/ndk/doc/smscomp/index.html?page=/ndk/doc/smscomp/sms_docs/data/hc2vlu5i.html))format.

### **wxDatetime::GetAsDOS**

**unsigned long GetAsDOS() const**

Returns the date and time inDOS

([http://developer.novell.com/ndk/doc/smscomp/index.html?page=/ndk/doc/smscomp/sms\\_docs/data/hc2vlu5i.html](http://developer.novell.com/ndk/doc/smscomp/index.html?page=/ndk/doc/smscomp/sms_docs/data/hc2vlu5i.html))format.

### **wxDatetime::IsEqualTo**

**bool IsEqualTo(const wxDateTime& datetime) const**

Returns `true` if the two dates are strictly identical.

### **wxDatetime::IsEarlierThan**

**bool IsEarlierThan(const wxDateTime& datetime) const**

Returns `true` if this date precedes the given one.

### **wxDatetime::IsLaterThan**

**bool IsLaterThan(const wxDateTime& datetime) const**

Returns `true` if this date is later than the given one.

### **wxDatetime::IsStrictlyBetween**

**bool IsStrictlyBetween(const wxDateTime& t1, const wxDateTime& t2) const**

Returns `true` if this date lies strictly between the two others,

**See also**

*IsBetween* (p. 366)

### **wxDatetime::IsBetween**

**bool IsBetween(const wxDateTime& t1, const wxDateTime& t2) const**

Returns `true` if *IsStrictlyBetween* (p. 366) is `true` or if the date is equal to one of the limit values.

**See also**

*IsStrictlyBetween* (p. 366)

**wxDateTime::IsSameDate**

**bool IsSameDate(const wxDateTime& dt) const**

Returns `true` if the date is the same without comparing the time parts.

**wxDateTime::IsSameTime**

**bool IsSameTime(const wxDateTime& dt) const**

Returns `true` if the time is the same (although dates may differ).

**wxDateTime::IsEqualUpTo**

**bool IsEqualUpTo(const wxDateTime& dt, const wxTimeSpan& ts) const**

Returns `true` if the date is equal to another one up to the given time interval, i.e. if the absolute difference between the two dates is less than this interval.

**wxDateTime::Add**

**wxDateTime Add(const wxTimeSpan& diff) const**

**wxDateTime& Add(const wxTimeSpan& diff)**

**wxDateTime& operator+=(const wxTimeSpan& diff)**

Adds the given time span to this object.

**wxPython note:** This method is named `AddTS` in wxPython.

**wxDateTime::Add**

**wxDateTime Add(const wxDateSpan& diff) const**

**wxDateTime& Add(const wxDateSpan& diff)**

**wxDateTime& operator+=(const wxDateSpan& diff)**

Adds the given date span to this object.

**wxPython note:** This method is named `AddDS` in wxPython.

**wxDateTime::Subtract**

**wxDatetime Subtract(const wxTimeSpan& diff) const**

**wxDatetime& Subtract(const wxTimeSpan& diff)**

**wxDatetime& operator-=(const wxTimeSpan& diff)**

Subtracts the given time span from this object.

**wxPython note:** This method is named `SubtractTS` in wxPython.

**wxDatetime::Subtract**

**wxDatetime Subtract(const wxDateSpan& diff) const**

**wxDatetime& Subtract(const wxDateSpan& diff)**

**wxDatetime& operator-=(const wxDateSpan& diff)**

Subtracts the given date span from this object.

**wxPython note:** This method is named `SubtractDS` in wxPython.

**wxDatetime::Subtract**

**wxTimeSpan Subtract(const wxDateTime& dt) const**

Subtracts another date from this one and returns the difference between them as `wxTimeSpan`.

**wxDatetime::ParseRfc822Date**

**const wxChar \* ParseRfc822Date(const wxChar\* date)**

Parses the string *date* looking for a date formatted according to the RFC 822 in it. The exact description of this format may, of course, be found in the RFC (section 5), but, briefly, this is the format used in the headers of Internet email messages and one of the most common strings expressing date in this format may be something like "Sat, 18 Dec 1999 00:48:30 +0100".

Returns `NULL` if the conversion failed, otherwise return the pointer to the character immediately following the part of the string which could be parsed. If the entire string contains only the date in RFC 822 format, the returned pointer will be pointing to a `NUL` character.

This function is intentionally strict, it will return an error for any string which is not RFC 822 compliant. If you need to parse date formatted in more free ways, you should use *ParseDateTime* (p. 369) or *ParseDate* (p. 369) instead.

**wxDatetime::ParseFormat**

**const wxChar \* ParseFormat(const wxChar \*date, const wxChar \*format = wxDefaultDateTimeFormat, const wxDateTime& dateDef = wxDefaultDateTime)**



This function parses the string *date* according to the given *format*. The system `strptime(3)` function is used whenever available, but even if it is not, this function is still implemented, although support for locale-dependent format specifiers such as `"%c"`, `"%x"` or `"%X"` may not be perfect and GNU extensions such as `"%z"` and `"%Z"` are not implemented. This function does handle the month and weekday names in the current locale on all platforms, however.

Please see the description of the ANSI C function `strptime(3)` for the syntax of the format string.

The *dateDef* parameter is used to fill in the fields which could not be determined from the format string. For example, if the format is `"%d"` (the day of the month), the month and the year are taken from *dateDef*. If it is not specified, *Today* (p. 359) is used as the default date.

Returns `NULL` if the conversion failed, otherwise return the pointer to the character which stopped the scan.

### **wxDateTime::ParseDateTime**

**const wxChar \* ParseDateTime(const wxChar \*datetime)**

Parses the string *datetime* containing the date and time in free format. This function tries as hard as it can to interpret the given string as date and time. Unlike *ParseRfc822Date* (p. 368), it will accept anything that may be accepted and will only reject strings which can not be parsed in any way at all.

Returns `NULL` if the conversion failed, otherwise return the pointer to the character which stopped the scan.

### **wxDateTime::ParseDate**

**const wxChar \* ParseDate(const wxChar \*date)**

This function is like *ParseDateTime* (p. 369), but it only allows the date to be specified. It is thus less flexible than *ParseDateTime* (p. 369), but also has less chances to misinterpret the user input.

Returns `NULL` if the conversion failed, otherwise return the pointer to the character which stopped the scan.

### **wxDateTime::ParseTime**

**const wxChar \* ParseTime(const wxChar \*time)**

This function is like *ParseDateTime* (p. 369), but only allows the time to be specified in the input string.

Returns `NULL` if the conversion failed, otherwise return the pointer to the character which stopped the scan.

**wxDateTime::Format**

**wxString Format(const wxChar \*format = wxDefaultDateTimeFormat, const TimeZone& tz = Local) const**

This function does the same as the standard ANSI C `strftime(3)` function. Please see its description for the meaning of *format* parameter.

It also accepts a few wxWidgets-specific extensions: you can optionally specify the width of the field to follow using `printf(3)`-like syntax and the format specification `%l` can be used to get the number of milliseconds.

**See also**

*ParseFormat* (p. 368)

**wxDateTime::FormatDate**

**wxString FormatDate() const**

Identical to calling *Format()* (p. 370) with "`%x`" argument (which means 'preferred date representation for the current locale').

**wxDateTime::FormatTime**

**wxString FormatTime() const**

Identical to calling *Format()* (p. 370) with "`%X`" argument (which means 'preferred time representation for the current locale').

**wxDateTime::FormatISODate**

**wxString FormatISODate() const**

This function returns the date representation in the ISO 8601 format (YYYY-MM-DD).

**wxDateTime::FormatISOTime**

**wxString FormatISOTime() const**

This function returns the time representation in the ISO 8601 format (HH:MM:SS).

**wxDateTime::SetToWeekDayInSameWeek**

**wxDateTime& SetToWeekDayInSameWeek(WeekDay weekday, WeekFlags flags = Monday\_First)**

Adjusts the date so that it will still lie in the same week as before, but its week day will be the given one.

Returns the reference to the modified object itself.

**wxDateTime::GetWeekDayInSameWeek**

**wxDateTime GetWeekDayInSameWeek(WeekDay weekday, WeekFlags flags = Monday\_First) const**

Returns the copy of this object to which *SetToWeekDayInSameWeek* (p. 370) was applied.

**wxDateTime::SetToNextWeekDay**

**wxDateTime& SetToNextWeekDay(WeekDay weekday)**

Sets the date so that it will be the first *weekday* following the current date.

Returns the reference to the modified object itself.

**wxDateTime::GetNextWeekDay**

**wxDateTime GetNextWeekDay(WeekDay weekday) const**

Returns the copy of this object to which *SetToNextWeekDay* (p. 371) was applied.

**wxDateTime::SetToPrevWeekDay**

**wxDateTime& SetToPrevWeekDay(WeekDay weekday)**

Sets the date so that it will be the last *weekday* before the current date.

Returns the reference to the modified object itself.

**wxDateTime::GetPrevWeekDay**

**wxDateTime GetPrevWeekDay(WeekDay weekday) const**

Returns the copy of this object to which *SetToPrevWeekDay* (p. 371) was applied.

**wxDateTime::SetToWeekDay**

**bool SetToWeekDay(WeekDay weekday, int n = 1, Month month = Inv\_Month, int year = Inv\_Year)**

Sets the date to the *n*-th *weekday* in the given month of the given year (the current month and year are used by default). The parameter *n* may be either positive (counting from the beginning of the month) or negative (counting from the end of it).

For example, *SetToWeekDay(2, wxDateTime::Wed)* will set the date to the second Wednesday in the current month and *SetToWeekDay(-1, wxDateTime::Sun)* -- to the last Sunday in it.

Returns *true* if the date was modified successfully, *false* otherwise meaning that the specified date doesn't exist.

**wxDateTime::GetWeekDay**

**wxDateTime GetWeekDay(WeekDay weekday, int n = 1, Month month = Inv\_Month, int year = Inv\_Year) const**

Returns the copy of this object to which *SetToWeekDay* (p. 371) was applied.

**wxDateTime::SetToLastWeekDay**

**bool SetToLastWeekDay(WeekDay weekday, Month month = Inv\_Month, int year = Inv\_Year)**

The effect of calling this function is the same as of calling *SetToWeekDay*(-1, weekday, month, year). The date will be set to the last *weekday* in the given month and year (the current ones by default).

Always returns `true`.

**wxDateTime::GetLastWeekDay**

**wxDateTime GetLastWeekDay(WeekDay weekday, Month month = Inv\_Month, int year = Inv\_Year)**

Returns the copy of this object to which *SetToLastWeekDay* (p. 372) was applied.

**wxDateTime::SetToWeekOfYear**

**static wxDateTime SetToWeekOfYear(int year, wxDateTime\_t numWeek, WeekDay weekday = Mon)**

Set the date to the given *weekday* in the week number *numWeek* of the given *year*. The number should be in range 1...53.

Note that the returned date may be in a different year than the one passed to this function because both the week 1 and week 52 or 53 (for leap years) contain days from different years. See *GetWeekOfYear* (p. 365) for the explanation of how the year weeks are counted.

**wxDateTime::SetToLastMonthDay**

**wxDateTime& SetToLastMonthDay(Month month = Inv\_Month, int year = Inv\_Year)**

Sets the date to the last day in the specified month (the current one by default).

Returns the reference to the modified object itself.

**wxDateTime::GetLastMonthDay**

**wxDateTime GetLastMonthDay(Month month = Inv\_Month, int year = Inv\_Year) const**

Returns the copy of this object to which *SetToLastMonthDay* (p. 372) was applied.

**wxDateTime::SetToYearDay****wxDateTime& SetToYearDay(wxDateTime\_t yday)**

Sets the date to the day number *yday* in the same year (i.e., unlike the other functions, this one does not use the current year). The day number should be in the range 1...366 for the leap years and 1...365 for the other ones.

Returns the reference to the modified object itself.

**wxDateTime::GetYearDay****wxDateTime GetYearDay(wxDateTime\_t yday) const**

Returns the copy of this object to which *SetToYearDay* (p. 373) was applied.

**wxDateTime::GetJulianDayNumber****double GetJulianDayNumber() const**

Returns the *JDN* (p. 361) corresponding to this date. Beware of rounding errors!

**See also**

*GetModifiedJulianDayNumber* (p. 373)

**wxDateTime::GetJDN****double GetJDN() const**

Synonym for *GetJulianDayNumber* (p. 373).

**wxDateTime::GetModifiedJulianDayNumber****double GetModifiedJulianDayNumber() const**

Returns the *Modified Julian Day Number* (MJD) which is, by definition, equal to JDN - 2400000.5. The MJDs are simpler to work with as the integral MJDs correspond to midnights of the dates in the Gregorian calendar and not th noons like JDN. The MJD 0 is Nov 17, 1858.

**wxDateTime::GetMJD****double GetMJD() const**

Synonym for *GetModifiedJulianDayNumber* (p. 373).

**wxDateTime::GetRataDie****double GetRataDie() const**

Return the *Rata Die number* of this date.

By definition, the Rata Die number is a date specified as the number of days relative to a base date of December 31 of the year 0. Thus January 1 of the year 1 is Rata Die day 1.

### **wxDateTime::FromTimezone**

**wxDateTime FromTimezone(const TimeZone& tz, bool noDST = false) const**

Transform the date from the given time zone to the local one. If *noDST* is `true`, no DST adjustments will be made.

Returns the date in the local time zone.

### **wxDateTime::ToTimezone**

**wxDateTime ToTimezone(const TimeZone& tz, bool noDST = false) const**

Transform the date to the given time zone. If *noDST* is `true`, no DST adjustments will be made.

Returns the date in the new time zone.

### **wxDateTime::MakeTimezone**

**wxDateTime& MakeTimezone(const TimeZone& tz, bool noDST = false)**

Modifies the object in place to represent the date in another time zone. If *noDST* is `true`, no DST adjustments will be made.

### **wxDateTime::MakeFromTimezone**

**wxDateTime& MakeFromTimezone(const TimeZone& tz, bool noDST = false)**

Same as *FromTimezone* (p. 374) but modifies the object in place.

### **wxDateTime::ToUTC**

**wxDateTime ToUTC(bool noDST = false) const**

This is the same as calling *ToTimezone* (p. 374) with the argument `GMT0`.

### **wxDateTime::MakeUTC**

**wxDateTime& MakeUTC(bool noDST = false)**

This is the same as calling *MakeTimezone* (p. 374) with the argument `GMT0`.

### **wxDateTime::IsDST**

**int IsDST(Country country = Country\_Default) const**

Returns `true` if the DST is applied for this date in the given country.

**See also**

*GetBeginDST* (p. 356) and *GetEndDST* (p. 357)

## **wxDateTimeHolidayAuthority**

TODO

## **wxDateTimeWorkDays**

TODO

## **wxDb**

A `wxDb` instance is a connection to an ODBC datasource which may be opened, closed, and re-opened an unlimited number of times. A database connection allows function to be performed directly on the datasource, as well as allowing access to any tables/views defined in the datasource to which the user has sufficient privileges.

See the *database classes overview* (p. 2152) for an introduction to using the ODBC classes.

**Include files**

<wx/db.h>

**Helper classes and data structures**

The following classes and structs are defined in `db.cpp/.h` for use with the `wxDb` class.

- *wxDbColFor* (p. 407)
- *wxDbCollnf* (p. 408)
- *wxDbTableInf* (p. 451)
- *wxDbInf* (p. 415)

**Constants**

NOTE: In a future release, all ODBC class constants will be prefaced with 'wx'.

<code>wxDB_PATH_MAX</code>	Maximum path length allowed to be passed
to	
	the ODBC driver to indicate where the
data	file(s) are located.

DB_MAX_COLUMN_NAME_LEN	Maximum supported length for the name of a column
DB_MAX_ERROR_HISTORY	Maximum number of error messages retained in the queue before being overwritten by new errors.
DB_MAX_ERROR_MSG_LEN	Maximum supported length of an error message returned by the ODBC classes
DB_MAX_STATEMENT_LEN	Maximum supported length for a complete SQL statement to be passed to the ODBC driver
DB_MAX_TABLE_NAME_LEN	Maximum supported length for the name of a table
DB_MAX_WHERE_CLAUSE_LEN	Maximum supported WHERE clause length that can be passed to the ODBC driver
DB_TYPE_NAME_LEN	Maximum length of the name of a column's data type

## Enumerated types

### Enumerated types

#### *enum wxDbSqlLogState*

sqlLogOFF, sqlLogON

#### *enum wxDBMS*

These are the databases currently tested and working with the ODBC classes. A call to `wxDb::Dbms` (p. 386) will return one of these enumerated values listed below.

- DB2
- DBase (IV, V)\*\*
- Firebird
- INFORMIX
- Interbase
- MS SQL Server (v7 - minimal testing)
- MS Access (97, 2000, 2002, and 2003)



- MySQL (2.x and 3.5 - use the 2.5x drivers though)
- Oracle (v7, v8, v8i)
- Pervasive SQL
- PostgreSQL
- Sybase (ASA and ASE)
- XBase Sequiter
- VIRTUOSO

See the remarks in *wxD::Dbms* (p. 386) for exceptions/issues with each of these database engines.

### Public member variables

#### *SWORD wxDb::cbErrorMsg*

This member variable is populated as a result of calling *wxD::GetNextError* (p. 394). Contains the count of bytes in the *wxD::errorMsg* string.

#### *int wxDb::DB\_STATUS*

The last ODBC error/status that occurred on this data connection. Possible codes are:

DB_ERR_GENERAL_WARNING	// SqlState = '01000'
DB_ERR_DISCONNECT_ERROR	// SqlState = '01002'
DB_ERR_DATA_TRUNCATED	// SqlState = '01004'
DB_ERR_PRIV_NOT_REVOKED	// SqlState = '01006'
DB_ERR_INVALID_CONN_STR_ATTR	// SqlState = '01S00'
DB_ERR_ERROR_IN_ROW	// SqlState = '01S01'
DB_ERR_OPTION_VALUE_CHANGED	// SqlState = '01S02'
DB_ERR_NO_ROWS_UPD_OR_DEL	// SqlState = '01S03'
DB_ERR_MULTI_ROWS_UPD_OR_DEL	// SqlState = '01S04'
DB_ERR_WRONG_NO_OF_PARAMS	// SqlState = '07001'
DB_ERR_DATA_TYPE_ATTR_VIOL	// SqlState = '07006'
DB_ERR_UNABLE_TO_CONNECT	// SqlState = '08001'
DB_ERR_CONNECTION_IN_USE	// SqlState = '08002'
DB_ERR_CONNECTION_NOT_OPEN	// SqlState = '08003'
DB_ERR_REJECTED_CONNECTION	// SqlState = '08004'
DB_ERR_CONN_FAIL_IN_TRANS	// SqlState = '08007'
DB_ERR_COMM_LINK_FAILURE	// SqlState = '08S01'
DB_ERR_INSERT_VALUE_LIST_MISMATCH	// SqlState = '21S01'
DB_ERR_DERIVED_TABLE_MISMATCH	// SqlState = '21S02'
DB_ERR_STRING_RIGHT_TRUNC	// SqlState = '22001'
DB_ERR_NUMERIC_VALUE_OUT_OF_RNG	// SqlState = '22003'
DB_ERR_ERROR_IN_ASSIGNMENT	// SqlState = '22005'
DB_ERR_DATETIME_FLD_OVERFLOW	// SqlState = '22008'
DB_ERR_DIVIDE_BY_ZERO	// SqlState = '22012'
DB_ERR_STR_DATA_LENGTH_MISMATCH	// SqlState = '22026'
DB_ERR_INTEGRITY_CONSTRAINT_VIOL	// SqlState = '23000'
DB_ERR_INVALID_CURSOR_STATE	// SqlState = '24000'

---

DB_ERR_INVALID_TRANS_STATE	// SqlState = '25000'
DB_ERR_INVALID_AUTH_SPEC	// SqlState = '28000'
DB_ERR_INVALID_CURSOR_NAME	// SqlState = '34000'
DB_ERR_SYNTAX_ERROR_OR_ACCESS_VIOL	// SqlState = '37000'
DB_ERR_DUPLICATE_CURSOR_NAME	// SqlState = '3C000'
DB_ERR_SERIALIZATION_FAILURE	// SqlState = '40001'
DB_ERR_SYNTAX_ERROR_OR_ACCESS_VIOL2	// SqlState = '42000'
DB_ERR_OPERATION_ABORTED	// SqlState = '70100'
DB_ERR_UNSUPPORTED_FUNCTION	// SqlState = 'IM001'
DB_ERR_NO_DATA_SOURCE	// SqlState = 'IM002'
DB_ERR_DRIVER_LOAD_ERROR	// SqlState = 'IM003'
DB_ERR_SQLALLOCENV_FAILED	// SqlState = 'IM004'
DB_ERR_SQLALLOCCONNECT_FAILED	// SqlState = 'IM005'
DB_ERR_SQLSETCONNECTOPTION_FAILED	// SqlState = 'IM006'
DB_ERR_NO_DATA_SOURCE_DLG_PROHIB	// SqlState = 'IM007'
DB_ERR_DIALOG_FAILED	// SqlState = 'IM008'
DB_ERR_UNABLE_TO_LOAD_TRANSLATION_DLL	// SqlState = 'IM009'
DB_ERR_DATA_SOURCE_NAME_TOO_LONG	// SqlState = 'IM010'
DB_ERR_DRIVER_NAME_TOO_LONG	// SqlState = 'IM011'
DB_ERR_DRIVER_KEYWORD_SYNTAX_ERROR	// SqlState = 'IM012'
DB_ERR_TRACE_FILE_ERROR	// SqlState = 'IM013'
DB_ERR_TABLE_OR_VIEW_ALREADY_EXISTS	// SqlState = 'S0001'
DB_ERR_TABLE_NOT_FOUND	// SqlState = 'S0002'
DB_ERR_INDEX_ALREADY_EXISTS	// SqlState = 'S0011'
DB_ERR_INDEX_NOT_FOUND	// SqlState = 'S0012'
DB_ERR_COLUMN_ALREADY_EXISTS	// SqlState = 'S0021'
DB_ERR_COLUMN_NOT_FOUND	// SqlState = 'S0022'
DB_ERR_NO_DEFAULT_FOR_COLUMN	// SqlState = 'S0023'
DB_ERR_GENERAL_ERROR	// SqlState = 'S1000'
DB_ERR_MEMORY_ALLOCATION_FAILURE	// SqlState = 'S1001'
DB_ERR_INVALID_COLUMN_NUMBER	// SqlState = 'S1002'
DB_ERR_PROGRAM_TYPE_OUT_OF_RANGE	// SqlState = 'S1003'
DB_ERR_SQL_DATA_TYPE_OUT_OF_RANGE	// SqlState = 'S1004'
DB_ERR_OPERATION_CANCELLED	// SqlState = 'S1008'
DB_ERR_INVALID_ARGUMENT_VALUE	// SqlState = 'S1009'
DB_ERR_FUNCTION_SEQUENCE_ERROR	// SqlState = 'S1010'
DB_ERR_OPERATION_INVALID_AT_THIS_TIME	// SqlState = 'S1011'
DB_ERR_INVALID_TRANS_OPERATION_CODE	// SqlState = 'S1012'
DB_ERR_NO_CURSOR_NAME_AVAIL	// SqlState = 'S1015'
DB_ERR_INVALID_STR_OR_BUF_LEN	// SqlState = 'S1090'
DB_ERR_DESCRIPTOR_TYPE_OUT_OF_RANGE	// SqlState = 'S1091'
DB_ERR_OPTION_TYPE_OUT_OF_RANGE	// SqlState = 'S1092'
DB_ERR_INVALID_PARAM_NO	// SqlState = 'S1093'
DB_ERR_INVALID_SCALE_VALUE	// SqlState = 'S1094'
DB_ERR_FUNCTION_TYPE_OUT_OF_RANGE	// SqlState = 'S1095'
DB_ERR_INF_TYPE_OUT_OF_RANGE	// SqlState = 'S1096'
DB_ERR_COLUMN_TYPE_OUT_OF_RANGE	// SqlState = 'S1097'
DB_ERR_SCOPE_TYPE_OUT_OF_RANGE	// SqlState = 'S1098'
DB_ERR_NULLABLE_TYPE_OUT_OF_RANGE	// SqlState = 'S1099'
DB_ERR_UNIQUENESS_OPTION_TYPE_OUT_OF_RANGE	// SqlState = 'S1100'
DB_ERR_ACCURACY_OPTION_TYPE_OUT_OF_RANGE	// SqlState = 'S1101'
DB_ERR_DIRECTION_OPTION_OUT_OF_RANGE	// SqlState = 'S1103'
DB_ERR_INVALID_PRECISION_VALUE	// SqlState = 'S1104'
DB_ERR_INVALID_PARAM_TYPE	// SqlState = 'S1105'
DB_ERR_FETCH_TYPE_OUT_OF_RANGE	// SqlState = 'S1106'
DB_ERR_ROW_VALUE_OUT_OF_RANGE	// SqlState = 'S1107'
DB_ERR_CONCURRENCY_OPTION_OUT_OF_RANGE	// SqlState = 'S1108'

---

---

```

DB_ERR_INVALID_CURSOR_POSITION        // SqlState = 'S1109'
DB_ERR_INVALID_DRIVER_COMPLETION      // SqlState = 'S1110'
DB_ERR_INVALID_BOOKMARK_VALUE         // SqlState = 'S1111'
DB_ERR_DRIVER_NOT_CAPABLE              // SqlState = 'S1C00'
DB_ERR_TIMEOUT_EXPIRED                 // SqlState = 'S1T00'

```

**struct wxDb::dbInf**

This structure is internal to the wxDb class and contains details of the ODBC datasource that the current instance of the wxDb is connected to in its members. When the datasource is opened, all of the information contained in the dbInf structure is queried from the datasource. This information is used almost exclusively within the ODBC class library. Where there may be a need for particular portions of this information outside of the class library, member functions (e.g. *wxDbTable::IsCursorClosedOnCommit* (p. 435)) have been added for ease of use.

```

wxChar dbmsName[40]           - Name of the dbms product
wxChar dbmsVer[64]            - Version # of the dbms product
wxChar driverName[40]         - Driver name
wxChar odbcVer[60]            - ODBC version of the driver
wxChar drvMgrOdbcVer[60]      - ODBC version of the driver manager
wxChar driverVer[60]          - Driver version
wxChar serverName[80]         - Server Name, typically a connect string
wxChar databaseName[128]      - Database filename
wxChar outerJoins[2]          - Does datasource support outer joins
wxChar procedureSupport[2]    - Does datasource support stored
                               procedures
UWORD  maxConnections         - Maximum # of connections datasource
                               supports
UWORD  maxStmts               - Maximum # of HSTMTs per HDBC
UWORD  apiConflvl             - ODBC API conformance level
UWORD  cliConflvl             - Is datasource SAG compliant
UWORD  sqlConflvl             - SQL conformance level
UWORD  cursorCommitBehavior   - How cursors are affected on db commit
UWORD  cursorRollbackBehavior - How cursors are affected on db
                               rollback
UWORD  supportNotNullClause    - Does datasource support NOT NULL
                               clause
wxChar supportIEF[2]          - Integrity Enhancement Facility (Ref.
                               Integrity)
UDWORD txnIsolation           - Transaction isolation level supported
by
                               driver
UDWORD txnIsolationOptions    - Transaction isolation level options
                               available
UDWORD fetchDirections        - Fetch directions supported
UDWORD lockTypes              - Lock types supported in SQLSetPos
UDWORD posOperations           - Position operations supported in
                               SQLSetPos
UDWORD posStmts               - Position statements supported
UDWORD scrollConcurrency       - Scrollable cursor concurrency options
                               supported
UDWORD scrollOptions           - Scrollable cursor options supported
UDWORD staticSensitivity      - Can additions/deletions/updates be
                               detected

```

UWORD	txnCapable	- Indicates if datasource supports transactions
UDWORD	loginTimeout	- Number seconds to wait for a login request

*wxChar* **wxDdb::errorList**[DB\_MAX\_ERROR\_HISTORY][DB\_MAX\_ERROR\_MSG\_LEN]

The last n ODBC errors that have occurred on this database connection.

*wxChar* **wxDdb::errorMsg**[SQL\_MAX\_MESSAGE\_LENGTH]

This member variable is populated as a result of calling *wxDdb::GetNextError* (p. 394). It contains the ODBC error message text.

*SDWORD* **wxDdb::nativeError**

Set by *wxDdb::DispAllErrors*, *wxDdb::GetNextError*, and *wxDdb::DispNextError*. It contains the datasource-specific error code returned by the datasource to the ODBC driver. Used for reporting ODBC errors.

*wxChar* **wxDdb::sqlState**[20]

Set by *wxDdb::TranslateSqlState*(). Indicates the error state after a failed ODBC operation. Used for reporting ODBC errors.

### Remarks

Default cursor scrolling is defined by *wxODBC\_FWD\_ONLY\_CURSORS* in *setup.h* when the *wxWidgets* library is built. This behavior can be overridden when an instance of a *wxDdb* is created (see *wxDdb constructor* (p. 382)). Default setting of this value true, as not all databases/drivers support both types of cursors.

### See also

*wxDdbColFor* (p. 407), *wxDdbColInf* (p. 408), *wxDdbTable* (p. 415), *wxDdbTableInf* (p. 451), *wxDdbInf* (p. 415)

### Associated non-class functions

The following functions are used in conjunction with the *wxDdb* class.

**void wxDbCloseConnections()**

### Remarks

Closes all cached connections that have been made through use of the *wxDdbGetConnection* (p. 380) function.

NOTE: These connections are closed regardless of whether they are in use or not. This function should only be called after the program has finished using the connections and all *wxDdbTable* instances that use any of the connections have been closed.

This function performs a *wxDdb::CommitTrans* (p. 384) on the connection before closing it to commit any changes that are still pending, as well as to avoid any function sequence

errors upon closing each connection.

**int wxDbConnectionsInUse()**

**Remarks**

Returns a count of how many database connections are currently free ( not being used) that have been cached through use of the *wxDGetConnection* (p. 380)function.

**bool wxDbFreeConnection(wxD \*pDb)**

**Remarks**

Searches the list of cached database connections connection for one matching the passed in wxDb instance. If found, that cached connection is freed.

Freeing a connection means that it is marked as available (free) in the cache of connections, so that a call to *wxDGetConnection* (p. 380)is able to return a pointer to the wxDb instance for use. Freeing a connection does NOT close the connection, it only makes the connection available again.

**wxD \* wxDbGetConnection(wxDConnectInf \*pDbConfig,bool  
FwdOnlyCursors=(bool)wxODBC\_FWD\_ONLY\_CURSORS)**

**Remarks**

This function is used to request a "new" wxDb instance for use by the program. The wxDb instance returned is also opened (see *wxD::Open* (p. 399)).

This function (along with *wxDFreeConnection()* and *wxDCloseConnection()*) maintain a cache of wxDb instances for user/re-use by a program. When a program needs a wxDb instance, it may call this function to obtain a wxDb instance. If there is a wxDb instance in the cache that is currently unused that matches the connection requirements specified in '*pDbConfig*' then that cached connection is marked as no longer being free, and a pointer to the wxDb instance is returned.

If there are no connections available in the cache that meet the requirements given in '*pDbConfig*', then a new wxDb instance is created to connect to the datasource specified in '*pDbConfig*' using the userID and password given in '*pDbConfig*'.

NOTE: The caching routine also uses the *wxD::Open* (p. 399)connection datatype copying code. If the call to *wxDGetConnection()* requests a connection to a datasource, and there is not one available in the cache, a new connection is created. But when the connection is opened, instead of polling the datasource over again for its datatypes, if a connection to the same datasource (using the same userID/password) has already been done previously, the new connection skips querying the datasource for its datatypes, and uses the same datatypes determined previously by the other connection(s) for that same datasource. This cuts down greatly on network traffic, database load, and connection creation time.

When the program is done using a connection created through a call to *wxDGetConnection()*, the program should call *wxDFreeConnection()* to release the wxDb instance back to the cache. DO NOT DELETE THE wxDb INSTANCE! Deleting the wxDb instance returned can cause a crash/memory corruption later in the program when

the cache is cleaned up.

When exiting the program, call `wxDbCloseConnections()` to close all the cached connections created by calls to `wxDbGetConnection()`.

**const wxChar \* wxDbLogExtendedErrorMsg(const wxChar \*userText, wxDb \*pDb, wxChar \*ErrFile, int ErrLine)**

Writes a message to the wxLog window (stdout usually) when an internal error situation occurs.

**bool wxDbSqlLog(wxDbSqlLogState state, const wxString &filename = SQL\_LOG\_FILENAME)**

#### Remarks

This function sets the sql log state for all open wxDb objects

**bool wxDbGetDataSource(HENV henv, wxChar \*Dsn, SWORD DsnMax, wxChar \*DsDesc, SWORD DsDescMax, UWORD direction = SQL\_FETCH\_NEXT)**

#### Remarks

This routine queries the ODBC driver manager for a list of available datasources. Repeatedly call this function to obtain all the datasources available through the ODBC driver manager on the current workstation.

```
wxArrayString strArray;

while (wxDbGetDataSource(DbConnectInf.GetHenv(), Dsn,
SQL_MAX_DSN_LENGTH+1, DsDesc, 255))
    strArray.Add(Dsn);
```

### **wxDb::wxDb**

#### **wxDb()**

Default constructor.

**wxDb(const HENV &aHenv, bool FwdOnlyCursors=(bool)wxODBC\_FWD\_ONLY\_CURSORS)**

Constructor, used to create an ODBC connection to a datasource.

#### Parameters

*aHenv*

Environment handle used for this connection. See `wxDConnectInf::AllocHenv` (p. 411)

*FwdOnlyCursors*

Will cursors created for use with this datasource connection only allow forward scrolling cursors.

### Remarks

This is the constructor for the wxDb class. The wxDb object must be created and opened before any database activity can occur.

### Example

```
wxDbConnectInf ConnectInf;
....Set values for member variables of ConnectInf here

wxDb sampleDB(ConnectInf.GetHenv());
if (!sampleDB.Open(ConnectInf.GetDsn(), ConnectInf.GetUserID(),
                  ConnectInf.GetPassword()))
{
    // Error opening datasource
}
```

### See also

*wxDbGetConnection* (p. 380)

### wxDb::Catalog

**bool Catalog(wxChar \* *userID*, const wxString &*fileName* = SQL\_CATALOG\_FILENAME)**

Allows a data "dictionary" of the datasource to be created, dumping pertinent information about all data tables to which the user specified in *userID* has access.

### Parameters

*userID*

Database user name to use in accessing the database. All tables to which this user has rights will be evaluated in the catalog.

*fileName*

*OPTIONAL.* Name of the text file to create and write the DB catalog to. Default is SQL\_CATALOG\_FILENAME.

### Return value

Returns true if the catalog request was successful, or false if there was some reason that the catalog could not be generated.

### Example

=====	=====	=====	=====	=====
TABLE NAME	COLUMN NAME	DATA TYPE	PRECISION	LENGTH
=====	=====	=====	=====	=====
EMPLOYEE	RECID	(0008)NUMBER	15	8

EMPLOYEE	USER_ID	(0012)VARCHAR2	13	13
EMPLOYEE	FULL_NAME	(0012)VARCHAR2	26	26
EMPLOYEE	PASSWORD	(0012)VARCHAR2	26	26
EMPLOYEE	START_DATE	(0011)DATE	19	16

**wxDB::Close****void Close()**

Closes the database connection.

**Remarks**

At the end of your program, when you have finished all of your database work, you must close the ODBC connection to the datasource. There are actually four steps involved in doing this as illustrated in the example.

Any wxDbTable instances which use this connection must be deleted before closing the database connection.

**Example**

```
// Commit any open transactions on the datasource
sampleDB.CommitTrans();

// Delete any remaining wxDbTable objects allocated with new
delete parts;

// Close the wxDb connection when finished with it
sampleDB.Close();
```

**wxDB::CommitTrans****bool CommitTrans()**

Permanently "commits" changes (insertions/deletions/updates) to the database.

**Return value**

Returns true if the commit was successful, or false if the commit failed.

**Remarks**

Transactions begin implicitly as soon as you make a change to the database with an insert/update/delete, or any other direct SQL command that performs one of these operations against the datasource. At any time thereafter, to save the changes to disk permanently, "commit" them by calling this function.

Calling this member function commits ALL open transactions on this ODBC connection. For example, if three different wxDbTable instances used the same connection to the datasource, committing changes made on one of those wxDbTable instances commits any pending transactions on all three wxDbTable instances.



Until a call to `wxDdb::CommitTrans()` is made, no other user or cursor is able to see any changes made to the row(s) that have been inserted/modified/deleted.

**Special Note : Cursors**

It is important to understand that different database/ODBC driver combinations handle transactions differently. One thing in particular that you must pay attention to is cursors, in regard to transactions. Cursors are what allow you to scroll through records forward and backward and to manipulate records as you scroll through them. When you issue a query, a cursor is created behind the scenes. The cursor keeps track of the query and keeps track of the current record pointer. After you commit or rollback a transaction, the cursor may be closed automatically. This is database dependent, and with some databases this behavior can be controlled through management functions. This means you would need to requery the datasource before you can perform any additional work using this cursor. This is only necessary however if the datasource closes the cursor after a commit or rollback. Use the `wxDdbTable::IsCursorClosedOnCommit` (p. 435) member function to determine the datasource's transaction behavior. Note, in many situations it is very inefficient to assume the cursor is closed and always requery. This could put a significant, unnecessary load on datasources that leave the cursors open after a transaction.

**wxDdb::CreateView**

**bool CreateView(const wxString & viewName, const wxString & collList, const wxString & pSqlStmt)**

Creates a SQL VIEW of one or more tables in a single datasource. Note that this function will only work against databases which support views (currently only Oracle as of November 21 2000).

**Parameters**

*viewName*

The name of the view. e.g. PARTS\_V

*collList*

*OPTIONAL* Pass in a comma delimited list of column names if you wish to explicitly name each column in the result set. If not desired, pass in an empty string and the column names from the associated table(s) will be used.

*pSqlStmt*

Pointer to the select statement portion of the CREATE VIEW statement. Must be a complete, valid SQL SELECT statement.

**Remarks**

A 'view' is a logical table that derives columns from one or more other tables or views. Once the view is created, it can be queried exactly like any other table in the database.

NOTE: Views are not available with all datasources. Oracle is one example of a datasource which does support views.

**Example**

```
// Incomplete code sample
db.CreateView("PARTS_SD1", "PN, PD, QTY",
              "SELECT PART_NUM, PART_DESC, QTY_ON_HAND * 1.1 FROM
PARTS \
              WHERE STORAGE_DEVICE = 1");

// PARTS_SD1 can now be queried just as if it were a data table.
// e.g. SELECT PN, PD, QTY FROM PARTS_SD1
```

**wxDdb::Dbms****wxDDBMS Dbms()****Remarks**

The return value will be of the enumerated type wxDBMS. This enumerated type contains a list of all the currently tested and supported databases.

Additional databases may work with these classes, but the databases returned by this function have been tested and confirmed to work with these ODBC classes.

Possible values returned by this function can be viewed in the *Enumerated types* (p. 376) section of wxDb.

There are known issues with conformance to the ODBC standards with several datasources supported by the wxWidgets ODBC classes. Please see the overview for specific details on which datasource have which issues.

**Return value**

The return value will indicate which of the supported datasources is currently connected to by this connection. In the event that the datasource is not recognized, a value of 'dbmsUNIDENTIFIED' is returned.

**wxDdb::DispAllErrors**

**bool DispAllErrors(HENV aHenv, HDBC aHdbc = SQL\_NULL\_HDBC, HSTMT aHstmt = SQL\_NULL\_HSTMT)**

Used to log all database errors that occurred as a result of an executed database command. This logging is automatic and also includes debug logging when compiled in debug mode via *wxLogDebug* (p. 1973). If logging is turned on via *wxDdb::SetSqlLogging* (p. 402), then an entry is also logged to the defined log file.

**Parameters**

*aHenv*

Handle to the ODBC environment.

*aHdbc*

Handle to the ODBC connection. Pass this in if the ODBC function call that erred required a `hdbc` or `hstmt` argument.

#### *aHstmt*

Handle to the ODBC statement being executed against. Pass this in if the ODBC function call that failed required a `hstmt` argument.

#### Remarks

This member function will log all of the ODBC error messages for the last ODBC function call that was made. This function is normally used internally within the ODBC class library, but can be used programmatically after calling ODBC functions directly (i.e. `SQLFreeEnv()`).

#### Return value

The function always returns false, so a call to this function can be made in the return statement of a code block in the event of a failure to perform an action (see the example below).

#### See also

`wxDb::SetSqlLogging` (p. 402), `wxDbSqlLog`

#### Example

```
if (SQLExecDirect(hstmt, (UCHAR FAR *) pSqlStmt, SQL_NTS) !=
    SQL_SUCCESS)
    // Display all ODBC errors for this stmt
    return(db.DispAllErrors(db.henv, db.hdbc, hstmt));
```

### **wxDb::DispNextError**

#### **void DispNextError()**

#### Remarks

This function is normally used internally within the ODBC class library. It could be used programmatically after calling ODBC functions directly. This function works in conjunction with `wxDb::GetNextError` (p. 394) when errors (or sometimes informational messages) returned from ODBC need to be analyzed rather than simply displaying them as an error. `GetNextError()` retrieves the next ODBC error from the ODBC error queue. The `wxDb` member variables `"sqlState"`, `"nativeError"` and `"errorMsg"` could then be evaluated. To display the error retrieved, `DispNextError()` could then be called. The combination of `GetNextError()` and `DispNextError()` can be used to iteratively step through the errors returned from ODBC evaluating each one in context and displaying the ones you choose.

#### Example

```
// Drop the table before attempting to create it
sprintf(sqlStmt, "DROP TABLE %s", tableName);
// Execute the drop table statement
if (SQLExecDirect(hstmt, (UCHAR FAR *)sqlStmt, SQL_NTS) !=
```

```
SQL_SUCCESS)
{
    // Check for sqlState = S0002, "Table or view not found".
    // Ignore this error, bomb out on any other error.
    pDb->GetNextError(henv, hdbc, hstmt);
    if (wxStrcmp(pDb->sqlState, "S0002"))
    {
        pDb->DispNextError(); // Displayed error retrieved
        pDb->DispAllErrors(henv, hdbc, hstmt); // Display all other
errors, if any
        pDb->RollbackTrans(); // Rollback the transaction
        CloseCursor();        // Close the cursor
        return(false);        // Return Failure
    }
}
```

### **wxDb::DropView**

**bool DropView(const wxString &viewName)**

Drops the data table view named in 'viewName'.

#### **Parameters**

*viewName*

Name of the view to be dropped.

#### **Remarks**

If the view does not exist, this function will return true. Note that views are not supported with all datasources.

### **wxDb::EscapeSqlChars**

**wxString EscapeSqlChars(const wxString& value)**

This function is used internally by wxWidgets while building SQL statements. It has been provided to help users who wish to explicitly construct SQL statements to be sent to the server. The function takes the value passed and returns it with any special characters escaped. Which characters are considered special depends on what type of datasource the object is connected to. For example, most database servers use a backslash as the escape character; if the value passed contains a backslash it will be replaced with a double backslash before it is passed to the server. This function can be used to avoid passing statements with syntax errors to the server as well as prevent SQL injection attacks.

#### **Parameters**

*value*

The value to be escaped.

### **wxDb::ExecSql**

**bool ExecSql(const wxString &pSqlStmt)**

**bool ExecSql(const wxString &pSqlStmt, wxDbColInf \*\*columns, short &numcols)**

Allows a native SQL command to be executed directly against the datasource. In addition to being able to run any standard SQL command, use of this function allows a user to (potentially) utilize features specific to the datasource they are connected to that may not be available through ODBC. The ODBC driver will pass the specified command directly to the datasource.

To get column amount and column names or other information about returned columns, pass 'columns' and 'numcols' parameters to the function also.

### Parameters

*pSqlStmt*

Pointer to the SQL statement to be executed.

*columns*

On success, this function will set this pointer to point to array of *wxDbColInf* (p. 408) objects, holding information about columns returned by the query. You need to call `delete[]` for the pointer you pass here after you don't use it anymore to prevent memory leak.

*numcols*

Reference to variable where amount of objects in 'columns'-parameter will be set.

### Remarks

This member extends the *wxDb* class and allows you to build and execute ANY VALID SQL statement against the datasource. This allows you to extend the class library by being able to issue any SQL statement that the datasource is capable of processing.

### See also

*wxDb::GetData* (p. 392), *wxDb::GetNext* (p. 394)

### **wxDb::FwdOnlyCursors**

**bool IsFwdOnlyCursors()**

Older form (pre-2.3/2.4 of *wxWidgets*) of the *wxDb::IsFwdOnlyCursors* (p. 396). This method is provided for backward compatibility only. The method *wxDb::IsFwdOnlyCursors* (p. 396) should be used in place of this method.

**wxDbInf \* GetCatalog(const wxChar \*userID)**

### **wxDb::GetCatalog**

**wxDbInf \* GetCatalog(const wxChar \*userID)**

Returns a *wxDblnf* (p. 415) pointer that points to the catalog (datasource) name, schema, number of tables accessible to the current user, and a *wxDbTableInf* pointer to all data pertaining to all tables in the users catalog.

### Parameters

*userID*

Owner/Schema of the table. Specify a *userID* when the datasource you are connected to allows multiple unique tables with the same name to be owned by different users. *userID* is evaluated as follows:

```
userID == NULL    ... UserID is ignored (DEFAULT)
userID == ""      ... UserID set equal to 'this->uid'
userID != ""      ... UserID set equal to 'userID'
```

### Remarks

The returned catalog will only contain catalog entries for tables to which the user specified in 'userID' has sufficient privileges. If no user is specified (NULL passed in), a catalog pertaining to all tables in the datasource accessible to the connected user (permissions apply) via this connection will be returned.

### wxDb::GetColumnCount

**int GetColumnCount(const wxString &tableName, const wxChar \*userID)**

### Parameters

*tableName*

The table name you wish to obtain column information about.

*userID*

Name of the user that owns the table(s) (also referred to as schema). Required for some datasources for situations where there may be multiple tables with the same name in the datasource, but owned by different users. *userID* is evaluated in the following manner:

```
userID == NULL    ... UserID is ignored (DEFAULT)
userID == ""      ... UserID set equal to 'this->uid'
userID != ""      ... UserID set equal to 'userID'
```

### Return value

Returns a count of how many columns are in the specified table. If an error occurs retrieving the number of columns, this function will return a -1.

### wxDb::GetColumns

**wxDbCollnf \* GetColumns(const wxString &tableName, UWORD \*numCols, const wxChar \*userID=NULL)**

**wxDdbColInf \* GetColumns(wxChar \*tableName[], const wxChar \*userID)**

### Parameters

*tableName*

The table name you wish to obtain column information about.

*numCols*

Pointer to a UWORD which will hold a count of the number of columns returned by this function

*tableName[]*

An array of pointers to table names you wish to obtain column information about. The last element of this array must be a NULL string.

*userID*

Name of the user that owns the table(s) (also referred to as schema). Required for some datasources for situations where there may be multiple tables with the same name in the datasource, but owned by different users. *userID* is evaluated in the following manner:

```
userID == NULL    ... UserID is ignored (DEFAULT)
userID == ""      ... UserID set equal to 'this->uid'
userID != ""      ... UserID set equal to 'userID'
```

### Return value

This function returns a pointer to an array of *wxDdbColInf* (p. 408) structures, allowing you to obtain information regarding the columns of the named table(s). If no columns were found, or an error occurred, this pointer will be NULL.

THE CALLING FUNCTION IS RESPONSIBLE FOR DELETING THE *wxDdbColInf* MEMORY WHEN IT IS FINISHED WITH IT.

ALL column bindings associated with this *wxDdb* instance are unbound by this function, including those used by any *wxDdbTable* instances that use this *wxDdb* instance. This function should use its own *wxDdb* instance to avoid undesired unbinding of columns.

### See also

*wxDdbColInf* (p. 408)

### Example

```
wxChar *tableList[] = {"PARTS", 0};
wxDdbColInf *colInf = pDb->GetColumns(tableList);
if (colInf)
{
    // Use the column inf
    .....
    // Destroy the memory
```

```
        delete [] colInf;  
    }
```

### **wxDdb::GetData**

**bool** GetData(**UWORD** *colNumber*, **SWORD** *cType*, **PTR** *pData*, **SDWORD** *maxLen*, **SDWORD FAR** \* *cbReturned*)

Used to retrieve result set data without binding column values to memory variables (i.e. not using a wxDbTable instance to access table data).

#### **Parameters**

*colNumber*

Ordinal number of the desired column in the result set to be returned.

*cType*

The C data type that is to be returned. See a partial list in *wxDdbTable::SetColDefs* (p. 443)

*pData*

Memory buffer which will hold the data returned by the call to this function.

*maxLen*

Maximum size of the buffer '*pData*' in characters. NOTE: Not UNICODE safe. If this is a numeric field, a value of 0 may be passed for this parameter, as the API knows the size of the expected return value.

*cbReturned*

Pointer to the buffer containing the length of the actual data returned. If this value comes back as SQL\_NULL\_DATA, then the *wxDdb::GetData* (p. 392) call has failed.

#### **See also**

*wxDdb::GetNext* (p. 394), *wxDdb::ExecSql* (p. 388)

#### **Example**

```
SDWORD cb;  
ULONG reqQty;  
wxString sqlStmt;  
sqlStmt = "SELECT SUM(REQUIRED_QTY - PICKED_QTY) FROM ORDER_TABLE  
WHERE \  
                PART_RECID = 1450 AND REQUIRED_QTY > PICKED_QTY";  
  
// Perform the query  
if (!pDb->ExecSql(sqlStmt.c_str()))  
{  
    // ERROR  
    return(0);  
}
```



```
    }

    // Request the first row of the result set
    if (!pDb->GetNext())
    {
        // ERROR
        return(0);
    }

    // Read column #1 of the row returned by the call to ::GetNext()
    // and return the value in 'reqQty'
    if (!pDb->GetData(1, SQL_C_ULONG, &reqQty, 0, &cb))
    {
        // ERROR
        return(0);
    }

    // Check for a NULL result
    if (cb == SQL_NULL_DATA)
        return(0);
```

### Remarks

When requesting multiple columns to be returned from the result set (for example, the SQL query requested 3 columns be returned), the calls to this function must request the columns in ordinal sequence (1,2,3 or 1,3 or 2,3).

### **wxDb::GetDatabaseName**

**const wxChar \* GetDatabaseName()**

Returns the name of the database engine.

### **wxDb::GetDatasourceName**

**const wxString & GetDatasourceName()**

Returns the ODBC datasource name.

### **wxDb::GetHDBC**

**HDBC GetHDBC()**

Returns the ODBC handle to the database connection.

### **wxDb::GetHENV**

**HENV GetHENV()**

Returns the ODBC environment handle.

### **wxDb::GetHSTMT**

**HSTMT GetHSTMT()**

Returns the ODBC statement handle associated with this database connection.

**wxDdb::GetKeyFields**

**int GetKeyFields(const wxString &tableName, wxDbCollnf \*collnf, UWORD numColumns)**

Used to determine which columns are members of primary or non-primary indexes on the specified table. If a column is a member of a foreign key for some other table, that information is detected also.

This function is primarily for use by the *wxDdb::GetColumns* (p. 390) function, but may be called if desired from the client application.

**Parameters**

*tableName*

Name of the table for which the columns will be evaluated as to their inclusion in any indexes.

*collnf*

Data structure containing the column definitions (obtained with *wxDdb::GetColumns* (p. 390)). This function populates the *PkCol*, *PkTableName*, and *FkTableName* members of the *collnf* structure.

*numColumns*

Number of columns defined in the instance of *collnf*.

**Return value**

Currently always returns true.

**See also**

*wxDdbCollnf* (p. 408), *wxDdb::GetColumns* (p. 390)

**wxDdb::GetNext**

**bool GetNext()**

Called after executing a query, this function requests the next row in the result set after the current position of the cursor.

**See also**

*wxDdb::ExecSql* (p. 388), *wxDdb::GetData* (p. 392)

**wxDdb::GetNextError**

```
bool GetNextError(HENV aHenv,HDBC aHdbc = SQL_NULL_HDBC, HSTMT aHstmt =  
SQL_NULL_HSTMT)
```

### Parameters

*aHenv*

A handle to the ODBC environment.

*aHdbc*

*OPTIONAL.* A handle to the ODBC connection. Pass this in if the ODBC function call that failed required a hdbc or hstmt argument.

*aHstmt*

*OPTIONAL.* A handle to the ODBC statement being executed against. Pass this in if the ODBC function call that failed requires a hstmt argument.

### Example

```
    if (SQLExecDirect(hstmt, (UCHAR FAR *) pSqlStmt, SQL_NTS) !=  
        SQL_SUCCESS)  
    {  
        return(db.GetNextError(db.henv, db.hdbc, hstmt));  
    }
```

### See also

*wxDdb::DispNextError* (p. 387), *wxDdb::DispAllErrors* (p. 386)

### **wxDdb::GetPassword**

```
const wxString & GetPassword()
```

Returns the password used to establish this connection to the datasource.

### **wxDdb::GetTableCount**

```
int GetTableCount()
```

Returns the number of *wxDdbTable()* instances currently using this datasource connection.

### **wxDdb::GetUsername**

```
const wxString & GetUsername()
```

Returns the user name (uid) used to establish this connection to the datasource.

### **wxDdb::Grant**

```
bool Grant(int privileges, const wxString &tableName, const wxString &userList =  
"PUBLIC")
```

Use this member function to GRANT privileges to users for accessing tables in the datasource.

### Parameters

#### *privileges*

Use this argument to select which privileges you want to grant. Pass DB\_GRANT\_ALL to grant all privileges. To grant individual privileges pass one or more of the following OR'd together:

```
DB_GRANT_SELECT   = 1
DB_GRANT_INSERT   = 2
DB_GRANT_UPDATE   = 4
DB_GRANT_DELETE   = 8
DB_GRANT_ALL      = DB_GRANT_SELECT | DB_GRANT_INSERT |
                    DB_GRANT_UPDATE | DB_GRANT_DELETE
```

#### *tableName*

The name of the table you wish to grant privileges on.

#### *userList*

*OPTIONAL.* A comma delimited list of users to grant the privileges to. If this argument is not passed in, the privileges will be given to the general PUBLIC.

### Remarks

Some databases require user names to be specified in all capital letters (i.e. Oracle). This function does not automatically capitalize the user names passed in the comma-separated list. This is the responsibility of the calling routine.

The currently logged in user must have sufficient grantor privileges for this function to be able to successfully grant the indicated privileges.

### Example

```
db.Grant(DB_GRANT_SELECT | DB_GRANT_INSERT, "PARTS", "mary,
sue");
```

### **wxDb::IsFwdOnlyCursors**

#### **bool IsFwdOnlyCursors()**

This setting indicates whether this database connection was created as being capable of using only forward scrolling cursors.

This function does NOT indicate if the ODBC driver or datasource supports backward scrolling cursors. There is no standard way of detecting if the driver or datasource can support backward scrolling cursors.

If a wxDb instance was created as being capable of only forward scrolling cursors, then even if the datasource and ODBC driver support backward scrolling cursors, tables using

this database connection would only be able to use forward scrolling cursors.

The default setting of whether a `wxDdb` connection to a database allows forward-only or also backward scrolling cursors is defined in `setup.h` by the value of `wxODBC_FWD_ONLY_CURSORS`. This default setting can be overridden when the `wxDdb` connection is initially created (see *wxDdb constructor* (p. 382) and *wxDdbGetConnection* (p. 380)).

### Return value

Returns true if this datasource connection is defined as using only forward scrolling cursors, or false if the connection is defined as being allowed to use backward scrolling cursors and their associated functions (see note above).

### Remarks

Added as of wxWidgets v2.4 release, this function is a renamed version of `wxDdb::FwdOnlyCursors()` to match the normal wxWidgets naming conventions for class member functions.

This function is not available in versions prior to v2.4. You should use *wxDdb::FwdOnlyCursors* (p. 389) for wxWidgets versions prior to 2.4.

### See also

*wxDdb constructor* (p. 382), *wxDdbGetConnection* (p. 380)

## **wxDdb::IsOpen**

### **bool IsOpen()**

Indicates whether the database connection to the datasource is currently opened.

### Remarks

This function may indicate that the database connection is open, even if the call to *wxDdb::Open* (p. 399) may have failed to fully initialize the connection correctly. The connection to the database is open and can be used via the direct SQL commands, if this function returns true. Other functions which depend on the *wxDdb::Open* (p. 399) to have completed correctly may not function as expected. The return result from *wxDdb::Open* (p. 399) is the only way to know if complete initialization of this `wxDdb` connection was successful or not. See *wxDdb::Open* (p. 399) for more details on partial failures to open a connection instance.

## **wxDdb::LogError**

**void LogError(const wxString &errMsg const wxString &SQLState= "")**

*errMsg*

Free-form text to display describing the error/text to be logged.

*SQLState*

*OPTIONAL.* Native SQL state error. Default is 0.

### Remarks

Calling this function will enter a log message in the error list maintained for the database connection. This log message is free form and can be anything the programmer wants to enter in the error list.

If SQL logging is turned on, the call to this function will also log the text into the SQL log file.

### See also

*wxDdb::WriteSqlLog* (p. 406)

### **wxDdb::ModifyColumn**

**void ModifyColumn(const wxString &tableName const wxString &columnNameint  
dataType ULONG columnLength=0 const wxString &optionalParam="")**

Used to change certain properties of a column such as the length, or whether a column allows NULLs or not.

*tableName*

Name of the table that the column to be modified is in.

*columnName*

Name of the column to be modified. NOTE: Name of column cannot be changed with this function.

*dataType*

Any one of DB\_DATA\_TYPE\_VARCHAR, DB\_DATA\_TYPE\_INTEGER, DB\_DATA\_TYPE\_FLOAT, DB\_DATA\_TYPE\_DATE.

*columnLength*

New size of the column. Valid only for DB\_DATA\_TYPE\_VARCHAR dataType fields. Default is 0.

*optionalParam*

Default is "".

### Remarks

Cannot be used to modify the precision of a numeric column, therefore 'columnLength' is ignored unless the dataType is DB\_DATA\_TYPE\_VARCHAR.

Some datasources do not allow certain properties of a column to be changed if any rows currently have data stored in that column. Those datasources that do allow columns to be changed with data in the rows many handle truncation and/or expansion in different ways.

Please refer to the reference material for the datasource being used for behavioral descriptions.

### Example

```
ok = pDb->ModifyColumn( "CONTACTS", "ADDRESS2",  
                        DB_, colDefs[j].SzDataObj,  
                        wxT( "NOT NULL" ) );
```

### wxDB::Open

**bool Open(const wxString &Dsn, const wxString &Uid, const wxString &AuthStr,  
bool failOnDataTypeUnsupported)**

**bool Open(const wxString &inConnectStr, bool failOnDataTypeUnsupported)**

**bool Open(wxDBConnectInf \*dbConnectInf, bool failOnDataTypeUnsupported)**

**bool Open(wxDB \*copyDb)**

Opens a connection to the datasource, sets certain behaviors of the datasource to confirm to the accepted behaviors (e.g. cursor position maintained on commits), and queries the datasource for its representations of the basic datatypes to determine the form in which the data going to/from columns in the data tables are to be handled.

The second form of this function, which accepts a "wxDB \*" as a parameter, can be used to avoid the overhead (execution time, database load, network traffic) which are needed to determine the data types and representations of data that are necessary for cross-datasource support by these classes.

Normally the first form of the wxDb::Open() function will open the connection and then send a series of queries to the datasource asking it for its representation of data types, and all the features it supports. If one connection to the datasource has already been made previously, the information gathered when that connection was created can just be copied to any new connections to the same datasource by passing a pointer to the first connection in as a parameter to the wxDb::Open() function. Note that this new connection created from the first connections information will use the same Dsn/Uid/AuthStr as the first connection used.

### Parameters

#### *Dsn*

datasource name. The name of the ODBC datasource as assigned when the datasource is initially set up through the ODBC data source manager.

#### *Uid*

User ID. The name (ID) of the user you wish to connect as to the datasource. The user name (ID) determines what objects you have access to in the datasource and what datasource privileges you have. Privileges include being able to create new objects, update objects, delete objects and so on. Users and privileges are normally administered by the database administrator.

***AuthStr***

The password associated with the *Uid*.

***failOnDataTypeUnsupporte***

As part of connecting to a database, the `wxD::Open()` function will query the database to find out the native types that it supports. With some databases, some data types may not be supported, or not sufficiently supported, for use with the `wxODBC` classes. Normally a program should fail in this case, so as not to try to use a data type that is not supported. This parameter allows the programmer to override the failure if they wish and continue on using the connection.

***dbConnectInf***

Contains a DSN, *Uid*, Password, or a connection string to be used in opening a new connection to the database. If a connection string is present, then the connection string will be used. If there is no connection string present, then the DSN, *Uid*, and Password are used.

***inConnectStr***

A valid ODBC connection string used to connect to a database

***copyDb***

Already completely configured and opened datasource connection from which all Dsn, *Uid*, *AuthStr*, connection string, and data typing information is to be copied from for use by this datasource connection. If 'copyDb' used a connection string to create its connection originally, then the connection being made by this call to `wxD::Open()` will use that same connection string.

**Remarks**

After a `wxD` instance is created, it must then be opened. When opening a datasource, there must be three pieces of information passed. The data source name, user name (ID) and the password for the user. No database activity on the datasource can be performed until the connection is opened. This is normally done at program startup and the datasource remains open for the duration of the program/module run.

It is possible to have connections to multiple datasources open at the same time to support distributed database connections by having separate instances of `wxD` objects that use either the same or different Dsn/*Uid*/*AuthStr* settings.

If this function returns a value of false, it does not necessarily mean that the connection to the datasource was not opened. It may mean that some portion of the initialization of the connection failed (such as a datatype not being able to be determined how the datasource represents it). To determine if the connection to the database failed, use the `wxD::IsOpen` (p. 397) function after receiving a false result back from this function to determine if the connection was opened or not. If this function returns false, but `wxD::IsOpen` (p. 397) returns true, then direct SQL commands may be passed to the database connection and can be successfully executed, but use of the datatypes (such as by a `wxDTable` instance) that are normally determined during open will not be possible.



The *Dsn*, *Uid*, and *AuthStr* string pointers that are passed in are copied. NOT the strings themselves, only the pointers. The calling routine must maintain the memory for these three strings for the life of the wxDb instance.

### Example

```
wxD b sampleDB(DbConnectInf.GetHenv());
if (!sampleDB.Open("Oracle 7.1 HP/UX", "gtasker", "myPassword"))
{
    if (sampleDb.IsOpen())
    {
        // Connection is open, but the initialization of
        // datatypes and parameter settings failed
    }
    else
    {
        // Error opening datasource
    }
}
```

### wxD b::RollbackTrans

#### bool RollbackTrans()

Function to "undo" changes made to the database. After an insert/update/delete, the operation may be "undone" by issuing this command any time before a *wxD b::CommitTrans* (p. 384) is called on the database connection.

#### Remarks

Transactions begin implicitly as soon as you make a change to the database. The transaction continues until either a commit or rollback is executed. Calling *wxD b::RollbackTrans()* will result in ALL changes done using this database connection that have not already been committed to be "undone" back to the last commit/rollback that was successfully executed.

Calling this member function rolls back ALL open (uncommitted) transactions on this ODBC connection, including all *wxD bTable* instances that use this connection.

#### See also

*wxD b::CommitTrans* (p. 384) for a special note on cursors

### wxD b::SetDebugErrorMessages

#### void SetDebugErrorMessages(bool state)

*state*

Either true (debug messages are logged) or false (debug messages are not logged).

#### Remarks

Turns on/off debug error messages from the ODBC class library. When this function is passed true, errors are reported to the user/logged automatically in a text or pop-up dialog when an ODBC error occurs. When passed false, errors are silently handled.

When compiled in release mode (FINAL=1), this setting has no affect.

### See also

*wxDb* constructor (p. 382)

### **wxDb::SetSqlLogging**

**bool SetSqlLogging(wxDbSqlLogState state, const wxString &filename = SQL\_LOG\_FILENAME, bool append = false)**

#### Parameters

*state*

Either *sqlLogOFF* or *sqlLogON* (see *enum wxDbSqlLogState* (p. 407)). Turns logging of SQL commands sent to the datasource OFF or ON.

*filename*

*OPTIONAL*. Name of the file to which the log text is to be written. Default is *SQL\_LOG\_FILENAME*.

*append*

*OPTIONAL*. Whether the file is appended to or overwritten. Default is false.

#### Remarks

When called with *sqlLogON*, all commands sent to the datasource engine are logged to the file specified by *filename*. Logging is done by embedded *wxDb::WriteSqlLog* (p. 406) calls in the database member functions, or may be manually logged by adding calls to *wxDb::WriteSqlLog* (p. 406) in your own source code.

When called with *sqlLogOFF*, the logging file is closed, and any calls to *wxDb::WriteSqlLog* (p. 406) are ignored.

### **wxDb::SQLColumnName**

**const wxString SQLColumnName(const wxChar \* colName)**

Returns the column name in a form ready for use in SQL statements. In most cases, the column name is returned verbatim. But some databases (e.g. MS Access, SQL Server, MSDE) allow for spaces in column names, which must be specially quoted. For example, if the datasource allows spaces in the column name, the returned string will have the correct enclosing marks around the name to allow it to be properly included in a SQL statement for the DBMS that is currently connected to with this connection.

#### Parameters

*colName*

Native name of the column in the table that is to be evaluated to determine if any special quoting marks needed to be added to it before including the column name in a SQL statement

**See also**

*wxDb::SQLTableName* (p. 403)

**wxDb::SQLTableName**

**const wxString SQLTableName(const wxChar \* *tableName*)**

Returns the table name in a form ready for use in SQL statements. In most cases, the table name is returned verbatim. But some databases (e.g. MS Access, SQL Server, MSDE) allow for spaces in table names, which must be specially quoted. For example, if the datasource allows spaces in the table name, the returned string will have the correct enclosing marks around the name to allow it to be properly included in a SQL statement for the data source that is currently connected to with this connection.

**Parameters**

*tableName*

Native name of the table that is to be evaluated to determine if any special quoting marks needed to be added to it before including the table name in a SQL statement

**See also**

*wxDb::SQLColumnName* (p. 402)

**wxDb::TableExists**

**bool TableExists(const wxString &*tableName*, const wxChar \**userID*=NULL, const wxString &*path*="")**

Checks the ODBC datasource for the existence of a table. If a *userID* is specified, then the table must be accessible by that user (user must have at least minimal privileges to the table).

**Parameters**

*tableName*

Name of the table to check for the existence of.

*userID*

Owner of the table (also referred to as schema). Specify a *userID* when the datasource you are connected to allows multiple unique tables with the same name to be owned by different users. *userID* is evaluated as follows:

`userID == NULL ... UserID is ignored (DEFAULT)`

```
userID == ""    ... UserID set equal to 'this->uid'
userID != ""    ... UserID set equal to 'userID'
```

### Remarks

*tableName* may refer to a table, view, alias or synonym.

This function does not indicate whether or not the user has privileges to query or perform other functions on the table. Use the *wxDdb::TablePrivileges* (p. 404) to determine if the user has sufficient privileges or not.

### See also

*wxDdb::TablePrivileges* (p. 404)

### wxDdb::TablePrivileges

**bool TablePrivileges(const wxString &tableName, const wxString &priv, const wxChar \*userID=NULL, const wxChar \*schema=NULL, const wxString &path="")**

Checks the ODBC datasource for the existence of a table. If a *userID* is specified, then the table must be accessible by that user (user must have at least minimal privileges to the table).

### Parameters

*tableName*

Name of the table on which to check privileges. *tableName* may refer to a table, view, alias or synonym.

*priv*

The table privilege being evaluated. May be one of the following (or a datasource specific privilege):

SELECT	: The connected user is permitted to retrieve data for one or more columns of the table.
INSERT	: The connected user is permitted to insert new rows containing data for one or more columns into the table.
UPDATE	: The connected user is permitted to update the data
in	one or more columns of the table.
DELETE	: The connected user is permitted to delete rows of data from the table.
REFERENCES	: Is the connected user permitted to refer to one or more columns of the table within a constraint (for example, a unique, referential, or table check constraint).

*userID*

*OPTIONAL.* User for which to determine if the privilege specified to be checked is granted or not. Default is "". *userID* is evaluated as follows:

```
userID == NULL    ... NOT ALLOWED!
userID == " "     ... UserID set equal to 'this->uid'
userID != " "     ... UserID set equal to 'userID'
```

*schema*

*OPTIONAL.* Owner of the table. Specify a *userID* when the datasource you are connected to allows multiple unique tables with the same name to be owned by different users. Specifying the table owner makes determination of the users privileges MUCH faster. Default is NULL. *userID* is evaluated as follows:

```
schema == NULL    ... Any owner (DEFAULT)
schema == " "     ... Owned by 'this->uid'
schema != " "     ... Owned by userID specified in 'schema'
```

*path*

*OPTIONAL.* Path to the table. Default is "". Currently unused.

**Remarks**

The scope of privilege allowed to the connected user by a given table privilege is datasource dependent.

For example, the privilege UPDATE might allow the connected user to update all columns in a table on one datasource, but only those columns for which the grantor (the user that granted the connected user) has the UPDATE privilege on another datasource.

Looking up a user's privileges to a table can be time consuming depending on the datasource and ODBC driver. This time can be minimized by passing a *schema* as a parameter. With some datasources/drivers, the difference can be several seconds of time difference.

**wxDdb::TranslateSqlState**

**int TranslateSqlState(const wxString &SQLState)**

Converts an ODBC sqlstate to an internal error code.

**Parameters**

*SQLState*

State to be converted.

**Return value**

Returns the internal class DB\_ERR code. See *wxDdb::DB\_STATUS* (p. 375) definition.

## **wxDdb::WriteSqlLog**

**bool WriteSqlLog(const wxString &logMsg)**

### **Parameters**

*logMsg*

Free form string to be written to the log file.

### **Remarks**

Very useful debugging tool that may be turned on/off during run time (see *wxDdb::SetSqlLogging* (p. 402) for details on turning logging on/off). The passed in string *logMsg* will be written to a log file if SQL logging is turned on.

### **Return value**

If SQL logging is off when a call to *WriteSqlLog()* is made, or there is a failure to write the log message to the log file, the function returns false without performing the requested log, otherwise true is returned.

### **See also**

*wxDdb::SetSqlLogging* (p. 402)

## **wxDdbColDataPtr**

Pointer to dynamic column definitions for use with a *wxDdbTable* instance. Currently there are no member functions for this class.

See the *database classes overview* (p. 2152) for an introduction to using the ODBC classes.

```
void    *PtrDataObj;  
int      SzDataObj;  
SWORD    SqlCtype;
```

## **wxDdbColDef**

This class is used to hold information about the columns bound to an instance of a *wxDdbTable* object.

Each instance of this class describes one column in the *wxDdbTable* object. When calling the *wxDdb constructor* (p. 382), a parameter passed in indicates the number of columns that will be defined for the *wxDdbTable* object. The constructor uses this information to allocate adequate memory for all of the column descriptions in your *wxDdbTable* object. Private member *wxDdbTable::colDefs* is a pointer to this chunk of memory maintained by the *wxDdbTable* class (and can be retrieved using the *wxDdbTable::GetColDefs* (p. 429) function). To access the *n*th column definition of your *wxDdbTable* object, just reference *wxDdbColDefs* element [*n* - 1].

Typically, `wxDbTable::SetColDefs` (p. 443) is used to populate an array of these data structures for the `wxDbTable` instance.

Currently there are no accessor functions for this class, so all members are public.

```

wxChar  ColName[DB_MAX_COLUMN_NAME_LEN+1];  // Column Name
int      DbDataType;      - Logical Data Type;
                        e.g. DB_DATA_TYPE_INTEGER
SWORD    SqlCtype;        - C data type; e.g. SQL_C_LONG
void     *PtrDataObj;     - Address of the data object
int      SzDataObj;       - Size, in bytes, of the data object
bool     KeyField;        - Is column part of the PRIMARY KEY for the
                        table? -- Date fields should NOT be
                        KeyFields
bool     Updateable;      - Column is updateable?
bool     InsertAllowed;   - Column included in INSERT statements?
bool     DerivedCol;      - Column is a derived value?
SDWORD   CbValue;         - !!!Internal use only!!!
bool     Null;            - NOT FULLY IMPLEMENTED
                        Allows NULL values in Inserts and
                        Updates

```

### See also

*database classes overview* (p. 2152), `wxDbTable::GetColDefs` (p. 429), `wxDb` constructor (p. 382)

### Include files

<wx/db.h>

### wxDbColDef::Initialize

Simply initializes all member variables to a cleared state. Called by the constructor automatically.

### wxDbColFor

Beginning support for handling international formatting specifically on dates and floats.

```

wxString      s_Field;      // Formatted String for Output
wxString      s_Format[7];  // Formatted Objects - TIMESTAMP has
                        the biggest (7)
wxString      s_Amount[7];  // Formatted Objects - amount of
                        things that can be formatted
int           i_Amount[7];  // Formatted Objects -
                        TT MM YYYY HH MM SS m
int           i_Nation;     // 0 = timestamp
                        1 = EU
                        2 = UK
                        3 = International
                        4 = US

```

```
int          i_dbDataType; // conversion of the 'sqlDataType'
                                to the generic data type used by
                                these classes
SWORD        i_sqlDataType;
```

The constructor for this class initializes all the values to zero or NULL.

The destructor does nothing at this time.

Only one function is provided with this class currently.

See the *database classes overview* (p. 2152) for an introduction to using the ODBC classes.

### Include files

<wx/db.h>

### wxDbColFor::Format

**int Format(int Nation, int dbDataType, SWORD sqlDataType, short columnSize, short decimalDigits)**

Work in progress, and should be inter-related with wxLocale eventually.

### wxDbColFor::Initialize

Simply initializes all member variables to a cleared state. Called by the constructor automatically.

### wxDbColInf

Used with the *wxDb::GetColumns* (p. 390) functions for obtaining all retrievable information about a column's definition.

```
wxChar        catalog[128+1];
wxChar        schema[128+1];
wxChar        tableName[DB_MAX_TABLE_NAME_LEN+1];
wxChar        colName[DB_MAX_COLUMN_NAME_LEN+1];
SWORD         sqlDataType;
wxChar        typeName[128+1];
SWORD         columnSize;
SWORD         bufferLength;
short         decimalDigits;
short         numPrecRadix;
short         nullable;
wxChar        remarks[254+1];
int           dbDataType; // conversion of the 'sqlDataType'
                                // to the generic data type used by
                                // these classes
int           PkCol;      // Primary key column
```



```
                                0 = No
                                1 = First Key
                                2 = Second Key, etc...
wxChar      PkTableName[DB_MAX_TABLE_NAME_LEN+1];
                                // Tables that use this PKey as a FKey
int          FkCol;             // Foreign key column
                                0 = No
                                1 = First Key
                                2 = Second Key, etc...
wxChar      FkTableName[DB_MAX_TABLE_NAME_LEN+1];
                                // Foreign key table name
wxDbColFor *pColFor;           // How should this column be formatted
```

The constructor for this class initializes all the values to zero, "", or NULL.

The destructor for this class takes care of deleting the pColFor member if it is non-NULL.

See the *database classes overview* (p. 2152) for an introduction to using the ODBC classes.

#### Include files

<wx/db.h>

#### wxDbCollnf::Initialize

Simply initializes all member variables to a cleared state. Called by the constructor automatically.

### wxDbConnectInf

This class is used for holding the data necessary for connecting to the ODBC datasource. That information includes: SQL environment handle, datasource name, user ID, password and default directory path (used with dBase). Other optional fields held in this class are and file type, both for future functions planned to be added for creating/manipulating datasource definitions.

#### wxDbConnectInf::wxDbConnectInf

##### wxDbConnectInf()

Default constructor.

```
wxDbConnectInf(HENV henv, const wxString &dsn,const wxString &userID="",  
const wxString &password,const wxString &defaultDir="", const wxString  
&description="",const wxString &fileType="")
```

Constructor which allows initial settings of all the classes member variables.

See the special note below on the henv parameter for forcing this constructor to create a

SQL environment handle automatically, rather than needing to pass one in to the function.

### Parameters

#### *henv*

Environment handle used for this connection. See *wxDConnectInf::AllocHenv* (p. 411) for how to create an SQL environment handle. NOTE: Passing in a NULL for this parameter will inform the constructor that it should create its own SQL environment handle. If NULL is passed for this parameter, the constructor will call *wxDConnectInf::AllocHenv* (p. 411) internally. A flag is set internally also to indicate that the HENV was created by the constructor so that when the default class destructor is called, the destructor will call *wxDConnectInf::FreeHenv* (p. 411) to free the environment handle automatically.

#### *dsn*

Name of the datasource to be used in creating wxDb instances for creating connection(s) to a datasource.

#### *userID*

*OPTIONAL* Many datasources allow (or even require) use of a username to determine privileges that connecting user is allowed to have when accessing the datasource or the data tables. Default is "".

#### *password*

*OPTIONAL* Password to be associated with the user ID specified in 'userID'. Default is "".

#### *defaultDir*

*OPTIONAL* Used for datasources which require the path to where the data file is stored to be specified. dBase is one example of the type of datasource which requires this information. Default is "".

#### *description*

*OPTIONAL FUTURE USE* Default is "".

#### *fileType*

*OPTIONAL FUTURE USE* Default is "".

### Remarks

It is strongly recommended that programs use the longer form of the constructor and allow the constructor to create the SQL environment handle automatically, and manage the destruction of the handle.

### Example

```
wxDConnectInf *DbConnectInf;
```

```
        DbConnectInf = new wxDbConnectInf(0, "MY_DSN", "MY_USER",
        "MY_PASSWORD" );

        ....the rest of the program

        delete DbConnectInf;
```

**See also**

*wxDConnectInf::AllocHenv* (p. 411), *wxDConnectInf::FreeHenv* (p. 411)

**wxDbConnectInf::~~wxDbConnectInf****~wxDbConnectInf()**

Handles the default destruction of the instance of the class. If the long form of the *wxDConnectInf* (p. 409) was used, then this destructor also takes care of calling *wxDConnectInf::FreeHenv* (p. 411) to free the SQL environment handle.

**wxDbConnectInf::AllocHenv****bool AllocHenv()**

Allocates a SQL environment handle that will be used to interface with an ODBC datasource.

**Remarks**

This function can be automatically called by the long form of the *wxDbConnectInf* (p. 409) constructor.

**wxDbConnectInf::FreeHenv****void FreeHenv()**

Frees the SQL environment handle being managed by the instance of this class.

**Remarks**

If the SQL environment handle was created using the long form of the *wxDbConnectInf* (p. 409) constructor, then the flag indicating that the HENV should be destroyed when the classes destructor is called is reset to be false, so that any future handles created using the *wxDbConnectInf::AllocHenv* (p. 411) function must be manually released with a call to this function.

**wxDbConnectInf::Initialize**

Simply initializes all member variables to a cleared state. Called by the constructor automatically.

**wxDbConnectInf::GetAuthStr**

**const wxChar \* GetAuthStr()**

Accessor function to return the password assigned for this class instance that will be used with the user ID.

Synonymous with *wxDbConnectInf::GetPassword* (p. 412)

**wxDbConnectInf::GetDefaultDir****const wxChar \* GetDefaultDir()**

Accessor function to return the default directory in which the datasource's data table is stored. This directory is only used for file based datasources like dBase. MS-Access does not require this to be set, as the path is set in the ODBC Administrator for MS-Access.

**wxDbConnectInf::GetDescription****const wxChar \* GetDescription()**

Accessor function to return the description assigned for this class instance.

NOTE: Description is a FUTURE USE item and is unused currently.

**wxDbConnectInf::GetDsn****const wxChar \* GetDsn()**

Accessor function to return the datasource name assigned for this class instance.

**wxDbConnectInf::GetFileType****const wxChar \* GetFileType()**

Accessor function to return the filetype of the ODBC datasource assigned for this class instance.

NOTE: FileType is a FUTURE USE item and is unused currently.

**wxDbConnectInf::GetHenv****const HENV GetHenv()**

Accessor function to return the SQL environment handle being managed by this class instance.

**wxDbConnectInf::GetPassword****const wxChar \* GetPassword()**

Accessor function to return the password assigned for this class instance that will be used with the user ID.

Synonymous with *wxDConnectInf::GetAuthStr* (p. 411)

### **wxDConnectInf::GetUid**

**const wxChar \* GetUid()**

Accessor function to return the user ID assigned for this class instance.

### **wxDConnectInf::GetUserID**

**const wxChar \* GetUserID()**

Accessor function to return the user ID assigned for this class instance.

### **wxDConnectInf::SetAuthStr**

**SetAuthStr**(const wxString &authstr)

Accessor function to assign the password for this class instance that will be used with the user ID.

Synonymous with *wxDConnectInf::SetPassword* (p. 414)

### **wxDConnectInf::SetDefaultDir**

**SetDefaultDir**(const wxString &defDir)

Accessor function to assign the default directory in which the datasource's data table is stored. This directory is only used for file based datasources like dBase. MS-Access does not require this to be set, as the path is set in the ODBC Administrator for MS-Access.

### **wxDConnectInf::SetDescription**

**SetDescription**(const wxString &desc)

Accessor function to assign the description assigned for this class instance.

NOTE: Description is a FUTURE USE item and is unused currently.

### **wxDConnectInf::SetDsn**

**SetDsn**(const wxString &dsn)

Accessor function to assign the datasource name for this class instance.

### **wxDConnectInf::SetFileType**

**SetFileType**(const wxString &)

Accessor function to return the filetype of the ODBC datasource assigned for this class

instance.

NOTE: FileType is a FUTURE USE item and is unused currently.

### **wxDBConnectInf::SetHenv**

**void SetHenv(const HENV henv)**

Accessor function to set the SQL environment handle for this class instance.

### **wxDBConnectInf::SetPassword**

**SetPassword(const wxString &password)**

Accessor function to assign the password for this class instance that will be used with the user ID.

Synonymous with *wxDBConnectInf::SetAuthStr* (p. 413)

### **wxDBConnectInf::SetUid**

**SetUid(const wxString &uid)**

Accessor function to set the user ID for this class instance.

### **wxDBConnectInf::SetUserID**

**SetUserID(const wxString &userID)**

Accessor function to assign the user ID for this class instance.

## **wXDbIdxDef**

Used in creation of non-primary indexes. Currently there are no member functions for this class.

```
wxChar  ColName[DB_MAX_COLUMN_NAME_LEN+1]
                                     // Name of column
bool     Ascending                  // Is index maintained in
                                     ASCENDING sequence?
```

There are no constructors/destructors as of this time, and no member functions.

See the *database classes overview* (p. 2152) for an introduction to using the ODBC classes.

### **Include files**

<wx/db.h>

## wxDblnf

Contains information regarding the database connection (datasource name, number of tables, etc). A pointer to a wxDbTableInf is included in this class so a program can create a wxDbTableInf array instance to maintain all information about all tables in the datasource to have all the datasource's information in one memory structure.

Primarily, this class is used internally by the wxWidgets ODBC classes.

```
wxChar      catalog[128+1];
wxChar      schema[128+1]; // typically means owner of table(s)
int         numTables;      // How many tables does this
                           //      datasource have
wxDbTableInf *pTableInf;    // Equals a new
                           //      wxDbTableInf[numTables];
```

The constructor for this class initializes all the values to zero, "", or NULL.

The destructor for this class takes care of deleting the pTableInf member if it is non-NULL.

See the *database classes overview* (p. 2152) for an introduction to using the ODBC classes.

### Include files

<wx/db.h>

### wxDblnf::Initialize

Simply initializes all member variables to a cleared state. Called by the constructor automatically.

## wxDbTable

A wxDbTable instance provides re-usable access to rows of data in a table contained within the associated ODBC datasource

See the *database classes overview* (p. 2152) for an introduction to using the ODBC classes.

### Include files

<wx/dbtable.h>  
<wx/db.h>

### Helper classes and data structures

The following classes and structs are defined in dbtable.cpp/.h for use with the wxDbTable class.

- *wxDdbColDef* (p. 406)
- *wxDdbColDataPtr* (p. 406)
- *wxDdbIdxDef* (p. 414)

### Constants

<code>wxDB_DEFAULT_CURSOR</code>	Primary cursor normally used for cursor based operations.
<code>wxDB_QUERY_ONLY</code>	Used to indicate whether a table that is opened is for query only, or if insert/update/deletes will be performed on the table. Less overhead (cursors and memory) are allocated for query only tables, plus read access times are faster with some datasources.
<code>wxDB_ROWID_LEN</code>	[Oracle only] - Used when <code>CanUpdateByRowID()</code> is true. Optimizes updates so they are faster by updating on the Oracle-specific ROWID column rather than some other index.
<code>wxDB_DISABLE_VIEW</code>	Use to indicate when a database view should not be if a table is normally set up to use a view. [Currently unsupported.]

### **wxDdbTable::wxDdbTable**

**wxDdbTable(wxDdb \*pwxDb, const wxString &tblName, const UWORD numColumns, const wxString &qryTblName = "", bool qryOnly = !wxDB\_QUERY\_ONLY, const wxString &tblPath = "")**

Default constructor.

#### **Parameters**

*pwxDb*

Pointer to the wxDb instance to be used by this wxDbTable instance.

*tblName*

The name of the table in the RDBMS.



*numColumns*

The number of columns in the table. (Do NOT include the ROWID column in the count if using Oracle).

*qryTblName*

*OPTIONAL.* The name of the table or view to base your queries on. This argument allows you to specify a table/view other than the base table for this object to base your queries on. This allows you to query on a view for example, but all of the INSERT, UPDATE and DELETES will still be performed on the base table for this wxDbTable object. Basing your queries on a view can provide a substantial performance increase in cases where your queries involve many tables with multiple joins. Default is "".

*qryOnly*

*OPTIONAL.* Indicates whether the table will be accessible for query purposes only, or should the table create the necessary cursors to be able to insert, update, and delete data from the table. Default is !wxDB\_QUERY\_ONLY.

*tblPath*

*OPTIONAL.* Some datasources (such as dBase) require a path to where the table is stored on the system. Default is "".

**wxDbTable::wxDbTable****virtual ~wxDbTable()**

Virtual default destructor.

**wxDbTable::BuildDeleteStmt**

**void BuildDeleteStmt(wxString &pSqlStmt, int typeOfDel, const wxString &pWhereClause= "")**

Constructs the full SQL statement that can be used to delete all rows matching the criteria in the pWhereClause.

**Parameters***pSqlStmt*

Pointer to buffer for the SQL statement retrieved. To be sure you have adequate space allocated for the SQL statement, allocate DB\_MAX\_STATEMENT\_LEN bytes.

*typeOfDel*

The type of delete statement being performed. Can be one of three values: DB\_DEL\_KEYFIELDS, DB\_DEL\_WHERE or DB\_DEL\_MATCHING

*pWhereClause*

*OPTIONAL.* If the `typeOfDel` is `DB_DEL_WHERE`, then you must also pass in a SQL WHERE clause in this argument. Default is "".

#### Remarks

This member function constructs a SQL DELETE statement. This can be used for debugging purposes if you are having problems executing your SQL statement.

WHERE and FROM clauses specified using `wxDbTable::SetWhereClause` (p. 448) and `wxDbTable::SetFromClause` (p. 446) are ignored by this function.

#### **wxDbTable::BuildSelectStmt**

**void BuildSelectStmt(wxString &pSqlStmt, int typeOfSelect, bool distinct)**

Constructs the full SQL statement that can be used to select all rows matching the criteria in the `pWhereClause`. This function is called internally in the `wxDbTable` class whenever the function `wxDbTable::Query` (p. 437) is called.

NOTE: Only the columns specified in `wxDbTable::SetColDefs` (p. 443) statements are included in the list of columns returned by the SQL statement created by a call to this function.

#### Parameters

*pSqlStmt*

Pointer to storage for the SQL statement retrieved. To be sure you have adequate space allocated for the SQL statement, allocate `DB_MAX_STATEMENT_LEN` bytes.

*typeOfSelect*

The type of select statement being performed. Can be one of four values: `DB_SELECT_KEYFIELDS`, `DB_SELECT_WHERE`, `DB_SELECT_MATCHING` or `DB_SELECT_STATEMENT`.

*distinct*

Whether to select distinct records only.

#### Remarks

This member function constructs a SQL SELECT statement. This can be used for debugging purposes if you are having problems executing your SQL statement.

WHERE and FROM clauses specified using `wxDbTable::SetWhereClause` (p. 448) and `wxDbTable::SetFromClause` (p. 446) are ignored by this function.

#### **wxDbTable::BuildUpdateStmt**

**void BuildUpdateStmt(wxString &pSqlStmt, int typeOfUpd, const wxString &pWhereClause= "")**

Constructs the full SQL statement that can be used to update all rows matching the criteria in the pWhereClause.

If typeOfUpdate is DB\_UPD\_KEYFIELDS, then the current values in the bound columns are used to determine which row(s) in the table are to be updated. The exception to this is when a datasource supports ROW IDs (Oracle). The ROW ID column is used for efficiency purposes when available.

NOTE: Only the columns specified in wxDbTable::SetColDefs (p. 443) statements are included in the list of columns updated by the SQL statement created by a call to this function. Any column definitions that were defined as being non-updateable will be excluded from the SQL UPDATE statement created by this function.

### Parameters

*pSqlStmt*

Pointer to storage for the SQL statement retrieved. To be sure you have adequate space allocated for the SQL statement, allocate DB\_MAX\_STATEMENT\_LEN bytes.

*typeOfUpdate*

The type of update statement being performed. Can be one of two values: DB\_UPD\_KEYFIELDS or DB\_UPD\_WHERE.

*pWhereClause*

OPTIONAL. If the typeOfUpdate is DB\_UPD\_WHERE, then you must also pass in a SQL WHERE clause in this argument. Default is "".

### Remarks

This member function allows you to see what the SQL UPDATE statement looks like that the ODBC class library builds. This can be used for debugging purposes if you are having problems executing your SQL statement.

WHERE and FROM clauses specified using wxDbTable::SetWhereClause (p. 448) and wxDbTable::SetFromClause (p. 446) are ignored by this function.

### wxDBTable::BuildWhereClause

**void BuildWhereClause(wxString &pWhereClause, int typeOfWhere, const wxString &qualTableName= "", bool useLikeComparison=false)**

Constructs the portion of a SQL statement which would follow the word 'WHERE' in a SQL statement to be passed to the datasource. The returned string does NOT include the word 'WHERE'.

### Parameters

*pWhereClause*

Pointer to storage for the SQL statement retrieved. To be sure you have adequate space allocated for the SQL statement, allocate DB\_MAX\_STATEMENT\_LEN bytes.

*typeOfWhere*

The type of where clause to generate. Can be one of two values: DB\_WHERE\_KEYFIELDS or DB\_WHERE\_MATCHING.

*qualTableName*

*OPTIONAL*. Prepended to all base table column names. For use when a FROM clause has been specified with the *wxDbTable::SetFromClause* (p. 446), to clarify which table a column name reference belongs to. Default is "".

*useLikeComparison*

*OPTIONAL*. Should the constructed WHERE clause utilize the LIKE comparison operator. If false, then the '=' operator is used. Default is false.

**Remarks**

This member function allows you to see what the SQL WHERE clause looks like that the ODBC class library builds. This can be used for debugging purposes if you are having problems executing your own SQL statements.

If using 'typeOfWhere' set to DB\_WHERE\_MATCHING, any bound columns currently containing a NULL value are not included in the WHERE clause's list of columns to use in the comparison.

**wxDbTable::CanSelectForUpdate****bool CanSelectForUpdate()**

Use this function to determine if the datasource supports SELECT ... FOR UPDATE. When the keywords "FOR UPDATE" are included as part of your SQL SELECT statement, all records *retrieved* (not just queried, but actually retrieved using *wxDbTable::GetNext* (p. 432), etc) from the result set are locked.

**Remarks**

Not all datasources support the "FOR UPDATE" clause, so you must use this member function to determine if the datasource currently connected to supports this behavior or not before trying to select using "FOR UPDATE".

If the *wxDbTable* instance was created with the parameter *wxDB\_QUERY\_ONLY*, then this function will return false. For all known databases which do not support the FOR UPDATE clause, this function will return false also.

**wxDbTable::CanUpdateByROWID**

**bool CanUpdateByROWID()**

CURRENTLY ONLY POSSIBLE IF USING ORACLE.

--- CURRENTLY DISABLED FOR \*ALL\* DATASOURCES --- NOV 1 2000 - gt

Every Oracle table has a hidden column named ROWID. This is a pointer to the physical location of the record in the datasource and allows for very fast updates and deletes. The key is to retrieve this ROWID during your query so it is available during an update or delete operation.

Use of the ROWID feature is always handled by the class library except in the case of *wxDBTable::QueryBySqlStmt* (p. 439). Since you are passing in the SQL SELECT statement, it is up to you to include the ROWID column in your query. If you do not, the application will still work, but may not be as optimized. The ROWID is always the last column in the column list in your SQL SELECT statement. The ROWID is not a column in the normal sense and should not be considered part of the column definitions for the *wxDBTable* object.

**Remarks**

The decision to include the ROWID in your SQL SELECT statement must be deferred until runtime since it depends on whether you are connected to an Oracle datasource or not.

**Example**

```
// Incomplete code sample
wxDBTable parts;
.....
if (parts.CanUpdateByROWID())
{
    // Note that the ROWID column must always be the last column
    selected
    sqlStmt = "SELECT PART_NUM, PART_DESC, ROWID" FROM PARTS";
}
else
    sqlStmt = "SELECT PART_NUM, PART_DESC FROM PARTS";
```

**wxDBTable::ClearMemberVar**

**void ClearMemberVar(UWORD colNumber, bool setToNull=false)**

Same as *wxDBTable::ClearMemberVars* (p. 422) except that this function clears only the specified column of its values, and optionally sets the column to be a NULL column.

*colNumber*

Column number that is to be cleared. This number (between 0 and (numColumns-1)) is the index of the column definition created using the *wxDBTable::SetColDefs* (p. 443) function.

*setToNull*

*OPTIONAL.* Indicates whether the column should be flagged as being a NULL value

stored in the bound memory variable. If true, then any value stored in the bound member variable is cleared. Default is false.

### **wxDbTable::ClearMemberVars**

**void ClearMemberVars**(bool *setToNull*=false)

Initializes all bound columns of the wxDbTable instance to zero. In the case of a string, zero is copied to the first byte of the string.

*setToNull*

*OPTIONAL.* Indicates whether all columns should be flagged as having a NULL value stored in the bound memory variable. If true, then any value stored in the bound member variable is cleared. Default is false.

### **Remarks**

This is useful before calling functions such as `wxDbTable::QueryMatching` (p. 440) or `wxDbTable::DeleteMatching` (p. 426) since these functions build their WHERE clauses from non-zero columns. To call either `wxDbTable::QueryMatching` (p. 440) or `wxDbTable::DeleteMatching` (p. 426) use this sequence:

- 1) `ClearMemberVars()`
- 2) Assign columns values you wish to match on
- 3) Call `wxDbTable::QueryMatching()` or `wxDbTable::DeleteMatching()`

### **wxDbTable::CloseCursor**

**bool CloseCursor**(HSTMT *cursor*)

Closes the specified cursor associated with the wxDbTable object.

### **Parameters**

*cursor*

The cursor to be closed.

### **Remarks**

Typically handled internally by the ODBC class library, but may be used by the programmer if desired.

DO NOT CLOSE THE wxDB\_DEFAULT\_CURSOR!

### **wxDbTable::Count**

**ULONG Count**(const wxString &*args*="\*\*")

Returns the number of records which would be in the result set using the current query parameters specified in the WHERE and FROM clauses.

## Parameters

*args*

*OPTIONAL.* This argument allows the use of the DISTINCT keyword against a column name to cause the returned count to only indicate the number of rows in the result set that have a unique value in the specified column. An example is shown below. Default is "", meaning a count of the total number of rows matching is returned, regardless of uniqueness.

## Remarks

This function can be called before or after an actual query to obtain the count of records in the result set. Count() uses its own cursor, so result set cursor positioning is not affected by calls to Count().

WHERE and FROM clauses specified using *wxDbTable::SetWhereClause* (p. 448) and *wxDbTable::SetFromClause* (p. 446) ARE used by this function.

## Example

```
USERS TABLE

FIRST_NAME      LAST_NAME
-----
John            Doe
Richard         Smith
Michael         Jones
John            Carpenter

// Incomplete code sample
wxDbTable users;
.....
users.SetWhereClause("");

// This Count() will return 4, as there are four users listed above
// that match the query parameters
totalNumberOfUsers = users.Count();

// This Count() will return 3, as there are only 3 unique first
names
// in the table above - John, Richard, Michael.
totalNumberOfUniqueFirstNames = users.Count("DISTINCT
FIRST_NAME");
```

## wxDbTable::CreateIndex

**bool CreateIndex(const wxString &IndexName, bool unique, UWORD  
numIndexColumns, wxDbIdxDef \*pIndexDefs, bool attemptDrop=true)**

This member function allows you to create secondary (non-primary) indexes on your tables. You first create your table, normally specifying a primary index, and then create any secondary indexes on the table. Indexes in relational model are not required. You do not need indexes to look up records in a table or to join two tables together. In the

relational model, indexes, if available, provide a quicker means to look up data in a table. To enjoy the performance benefits of indexes, the indexes must be defined on the appropriate columns and your SQL code must be written in such a way as to take advantage of those indexes.

### Parameters

#### *IndexName*

Name of the Index. Name must be unique within the table space of the datasource.

#### *unique*

Indicates if this index is unique.

#### *numIndexColumns*

Number of columns in the index.

#### *pIndexDefs*

A pointer to an array *wxDblIdxDef* (p. 414) structures.

#### *attemptDrop*

*OPTIONAL*. Indicates if the function should try to execute a *wxDblTable::DropIndex* (p. 428) on the index name provided before trying to create the index name. Default is true.

### Remarks

The first parameter, index name, must be unique and should be given a meaningful name. Common practice is to include the table name as a prefix in the index name (e.g. For table PARTS, you might want to call your index PARTS\_Index1). This will allow you to easily view all of the indexes defined for a given table grouped together alphabetically.

The second parameter indicates if the index is unique or not. Uniqueness is enforced at the RDBMS level preventing rows which would have duplicate indexes from being inserted into the table when violating a unique index's uniqueness.

In the third parameter, specify how many columns are in your index. This number must match the number of columns defined in the 'pIndexDefs' parameter.

The fourth parameter specifies which columns make up the index using the *wxDblIdxDef* (p. 414) structure. For each column in the index, you must specify two things, the column name and the sort order (ascending / descending). See the example below to see how to build and pass in the *wxDblIdxDef* (p. 414) structure.

The fifth parameter is provided to handle the differences in datasources as to whether they will automatically overwrite existing indexes with the same name or not. Some datasources require that the existing index must be dropped first, so this is the default behavior.

Some datasources (MySQL, and possibly others) require columns which are to be part of



an index to be defined as NOT NULL. When this function is called, if a column is not defined to be NOT NULL, a call to this function will modify the column definition to change any columns included in the index to be NOT NULL. In this situation, if a NULL value already exists in one of the columns that is being modified, creation of the index will fail.

PostGres is unable to handle index definitions which specify whether the index is ascending or descending, and defaults to the system default when the index is created.

It is not necessary to call `wxDdb::CommitTrans` (p. 384) after executing this function.

### Example

```
// Create a secondary index on the PARTS table
wxDdbIdxDef IndexDef[2]; // 2 columns make up the index

wxStrncpy(IndexDef[0].ColName, "PART_DESC"); // Column 1
IndexDef[0].Ascending = true;

wxStrncpy(IndexDef[1].ColName, "SERIAL_NO"); // Column 2
IndexDef[1].Ascending = false;

// Create a name for the index based on the table's name
wxString indexName;
indexName.Printf("%s_Index1", parts->GetTableName());
parts->CreateIndex(indexName, true, 2, IndexDef);
```

### **wxDdbTable::CreateTable**

**bool CreateTable**(bool *attemptDrop=true*)

Creates a table based on the definitions previously defined for this `wxDdbTable` instance.

#### Parameters

*attemptDrop*

*OPTIONAL.* Indicates whether the driver should attempt to drop the table before trying to create it. Some datasources will not allow creation of a table if the table already exists in the table space being used. Default is true.

#### Remarks

This function creates the table and primary index (if any) in the table space associated with the connected datasource. The owner of these objects will be the user id that was given when `wxDdb::Open` (p. 399) was called. The objects will be created in the default schema/table space for that user.

In your derived `wxDdbTable` object constructor, the columns and primary index of the table are described through the `wxDdbColDef` (p. 406) structure. `wxDdbTable::CreateTable` (p. 425) uses this information to create the table and to add the primary index. See `wxDdbTable` (p. 415) ctor and `wxDdbColDef` description for additional information on describing the columns of the table.

It is not necessary to call `wxDdb::CommitTrans` (p. 384) after executing this function.

**wxDbTable::DB\_STATUS****bool DB\_STATUS()**

Accessor function that returns the wxDb private member variable DB\_STATUS for the database connection used by this instance of wxDbTable.

**wxDbTable::Delete****bool Delete()**

Deletes the row from the table indicated by the current cursor.

**Remarks**

Use *wxDbTable::GetFirst* (p. 430), *wxDbTable::GetLast* (p. 431), *wxDbTable::GetNext* (p. 432) or *wxDbTable::GetPrev* (p. 432) to position the cursor to a valid record. Once positioned on a record, call this function to delete the row from the table.

A *wxDb::CommitTrans* (p. 384) or *wxDb::RollbackTrans* (p. 401) must be called after use of this function to commit or rollback the deletion.

NOTE: Most datasources have a limited size "rollback" segment. This means that it is only possible to insert/update/delete a finite number of rows without performing a *wxDb::CommitTrans* (p. 384) or *wxDb::RollbackTrans* (p. 401). Size of the rollback segment varies from database to database, and is user configurable in most databases. Therefore it is usually best to try to perform a commit or rollback at relatively small intervals when processing a larger number of actions that insert/update/delete rows in a table.

**wxDbTable::DeleteCursor****bool DeleteCursor(HSTMT \*hstmtDel)**

Allows a program to delete a cursor.

**Parameters**

*hstmtDel*

Handle of the cursor to delete.

**Remarks**

For default cursors associated with the instance of wxDbTable, it is not necessary to specifically delete the cursors. This is automatically done in the wxDbTable destructor.

NOTE: If the cursor could not be deleted for some reason, an error is logged indicating the reason. Even if the cursor could not be deleted, the HSTMT that is passed in is deleted, and the pointer is set to NULL.

DO NOT DELETE THE wxDB\_DEFAULT\_CURSOR!

**wxDbTable::DeleteMatching**

**bool DeleteMatching()**

This member function allows you to delete records from your wxDbTable object by specifying the data in the columns to match on.

**Remarks**

To delete all users with a first name of "JOHN", do the following:

1. Clear all "columns" using wxDbTable::ClearMemberVars().
2. Set the FIRST\_NAME column equal to "JOHN".
3. Call wxDbTable::DeleteMatching().

The WHERE clause is built by the ODBC class library based on all non-NULL columns. This allows deletion of records by matching on any column(s) in your wxDbTable instance, without having to write the SQL WHERE clause.

A wxDb::CommitTrans (p. 384) or wxDb::RollbackTrans (p. 401) must be called after use of this function to commit or rollback the deletion.

NOTE: Row(s) should be locked before deleting them to make sure they are not already in use. This can be achieved by calling wxDbTable::QueryMatching (p. 440), and then retrieving the records, locking each as you go (assuming FOR UPDATE is allowed on the datasource). After the row(s) have been successfully locked, call this function.

NOTE: Most datasources have a limited "rollback" segment. This means that it is only possible to insert/update/delete a finite number of rows without performing a wxDb::CommitTrans (p. 384) or wxDb::RollbackTrans (p. 401). Size of the rollback segment varies from database to database, and is user configurable in most databases. Therefore it is usually best to try to perform a commit or rollback at relatively small intervals when processing a larger number of actions that insert/update/delete rows in a table.

**Example**

```
// Incomplete code sample to delete all users with a first name
// of "JOHN"
users.ClearMemberVars();
wxStrcpy(users.FirstName, "JOHN");
users.DeleteMatching();
```

**wxDATABASE::DeleteWhere****bool DeleteWhere(const wxString &pWhereClause)**

Deletes all rows from the table which match the criteria specified in the WHERE clause that is passed in.

**Parameters**

*pWhereClause*

SQL WHERE clause. This WHERE clause determines which records will be deleted

from the table interfaced through the `wxDbTable` instance. The WHERE clause passed in must be compliant with the SQL 92 grammar. Do not include the keyword 'WHERE'

### Remarks

This is the most powerful form of the `wxDbTable` delete functions. This function gives access to the full power of SQL. This function can be used to delete records by passing a valid SQL WHERE clause. Sophisticated deletions can be performed based on multiple criteria using the full functionality of the SQL language.

A `wxDb::CommitTrans` (p. 384) must be called after use of this function to commit the deletions.

Note: This function is limited to deleting records from the table associated with this `wxDbTable` object only. Deletions on joined tables is not possible.

NOTE: Most datasources have a limited size "rollback" segment. This means that it is only possible to insert/update/delete a finite number of rows without performing a `wxDb::CommitTrans` (p. 384) or `wxDb::RollbackTrans` (p. 401). Size of the rollback segment varies from database to database, and is user configurable in most databases. Therefore it is usually best to try to perform a commit or rollback at relatively small intervals when processing a larger number of actions that insert/update/delete rows in a table.

WHERE and FROM clauses specified using `wxDbTable::SetWhereClause` (p. 448) and `wxDbTable::SetFromClause` (p. 446) are ignored by this function.

### Example

```
// Delete parts 1 thru 10 from containers 'X', 'Y' and 'Z' that
// are magenta in color
parts.DeleteWhere("(PART_NUMBER BETWEEN 1 AND 10) AND \
                  CONTAINER IN ('X', 'Y', 'Z') AND \
                  UPPER(COLOR) = 'MAGENTA'");
```

### `wxDbTable::DropIndex`

**`bool DropIndex(const wxString &IndexName)`**

Allows an index on the associated table to be dropped (deleted) if the user login has sufficient privileges to do so.

### Parameters

*IndexName*

Name of the index to be dropped.

### Remarks

If the index specified in the 'IndexName' parameter does not exist, an error will be logged, and the function will return a result of false.

It is not necessary to call `wxDb::CommitTrans` (p. 384) after executing this function.

**wxDATABASE::DropTable****bool DropTable()**

Deletes the associated table if the user has sufficient privileges to do so.

**Remarks**

This function returns true if the table does not exist, but only for supported databases (see *wxDATABASE::Dbms* (p. 386)). If a datasource is not specifically supported, and this function is called, the function will return false.

Most datasources/ODBC drivers will delete any indexes associated with the table automatically, and others may not. Check the documentation for your database to determine the behavior.

It is not necessary to call *wxDATABASE::CommitTrans* (p. 384) after executing this function.

**wxDATABASE::From****const wxString & From()****void From(const wxString &From)**

Accessor function for the private class member *wxDATABASE::from*. Can be used as a synonym for *wxDATABASE::GetFromClause* (p. 431) (the first form of this function) or *wxDATABASE::SetFromClause* (p. 446) (the second form of this function).

**Parameters***From*

A comma separated list of table names that are to be inner joined with the base table's columns so that the joined table's columns may be returned in the result set or used as a portion of a comparison with the base table's columns. NOTE that the base table's name must NOT be included in the FROM clause, as it is automatically included by the *wxDATABASE* class in constructing query statements.

**Return value**

The first form of this function returns the current value of the *wxDATABASE* member variable *from*.

The second form of the function has no return value, as it will always set the from clause successfully.

**See also**

*wxDATABASE::GetFromClause* (p. 431), *wxDATABASE::SetFromClause* (p. 446)

**wxDATABASE::GetColDefs****wxDATABASE \* GetColDefs()**

Accessor function that returns a pointer to the array of column definitions that are bound to the columns that this `wxDbTable` instance is associated with.

To determine the number of elements pointed to by the returned `wxDbColDef` (p. 406) pointer, use the `wxDbTable::GetNumberOfColumns` (p. 432) function.

**Remarks**

These column definitions must not be manually redefined after they have been set.

**wxDbTable::GetCursor****HSTMT GetCursor()**

Returns the HSTMT value of the current cursor for this `wxDbTable` object.

**Remarks**

This function is typically used just before changing to use a different cursor so that after the program is finished using the other cursor, the current cursor can be set back to being the cursor in use.

**See also**

`wxDbTable::SetCursor` (p. 445), `wxDbTable::GetNewCursor` (p. 431)

**wxDbTable::GetDb****wxDb \* GetDb()**

Accessor function for the private member variable `pDb` which is a pointer to the datasource connection that this `wxDbTable` instance uses.

**wxDbTable::GetFirst****bool GetFirst()**

Retrieves the FIRST row in the record set as defined by the current query. Before retrieving records, a query must be performed using `wxDbTable::Query` (p. 437), `wxDbTable::QueryOnKeyFields` (p. 441), `wxDbTable::QueryMatching` (p. 440) or `wxDbTable::QueryBySqlStmt` (p. 439).

**Remarks**

This function can only be used if the datasource connection used by the `wxDbTable` instance was created with `FwdOnlyCursors` set to false. If the connection does not allow backward scrolling cursors, this function will return false, and the data contained in the bound columns will be undefined.

**See also**

`wxDb::IsFwdOnlyCursors` (p. 396)

**wxDbTable::GetFromClause****const wxString & GetFromClause()**

Accessor function that returns the current FROM setting assigned with the *wxDbTable::SetFromClause* (p. 446).

**See also**

*wxDbTable::From* (p. 429)

**wxDbTable::GetLast****bool GetLast()**

Retrieves the LAST row in the record set as defined by the current query. Before retrieving records, a query must be performed using *wxDbTable::Query* (p. 437), *wxDbTable::QueryOnKeyFields* (p. 441), *wxDbTable::QueryMatching* (p. 440) or *wxDbTable::QueryBySqlStmt* (p. 439).

**Remarks**

This function can only be used if the datasource connection used by the *wxDbTable* instance was created with *FwdOnlyCursors* set to false. If the connection does not allow backward scrolling cursors, this function will return false, and the data contained in the bound columns will be undefined.

**See also**

*wxDb::IsFwdOnlyCursors* (p. 396)

**wxDbTable::GetNewCursor****HSTMT \* GetNewCursor(bool setCursor=false, bool bindColumns=true)**

This function will create a new cursor that can be used to access the table being referenced by this *wxDbTable* instance, or to execute direct SQL commands on without affecting the cursors that are already defined and possibly positioned.

**Parameters**

*setCursor*

*OPTIONAL*. Should this new cursor be set to be the current cursor after successfully creating the new cursor. Default is false.

*bindColumns*

*OPTIONAL*. Should this new cursor be bound to all the memory variables that the default cursor is bound to. Default is true.

**Remarks**

This new cursor must be closed using `wxDbTable::DeleteCursor` (p. 426) by the calling program before the `wxDbTable` instance is deleted, or both memory and resource leaks will occur.

### **wxDbTable::GetNext**

#### **bool GetNext()**

Retrieves the NEXT row in the record set after the current cursor position as defined by the current query. Before retrieving records, a query must be performed using `wxDbTable::Query` (p. 437), `wxDbTable::QueryOnKeyFields` (p. 441), `wxDbTable::QueryMatching` (p. 440) or `wxDbTable::QueryBySqlStmt` (p. 439).

#### **Return value**

This function returns false when the current cursor has reached the end of the result set. When false is returned, data in the bound columns is undefined.

#### **Remarks**

This function works with both forward and backward scrolling cursors.

**See also** `wxDbTable::++` (p. 451)

### **wxDbTable::GetNumberOfColumns**

#### **UWORD GetNumberOfColumns()**

Accessor function that returns the number of columns that are statically bound for access by the `wxDbTable` instance.

### **wxDbTable::GetOrderByClause**

#### **const wxString & GetOrderByClause()**

Accessor function that returns the current ORDER BY setting assigned with the `wxDbTable::SetOrderByClause` (p. 447).

#### **See also**

`wxDbTable::OrderBy` (p. 437)

### **wxDbTable::GetPrev**

#### **bool GetPrev()**

Retrieves the PREVIOUS row in the record set before the current cursor position as defined by the current query. Before retrieving records, a query must be performed using `wxDbTable::Query` (p. 437), `wxDbTable::QueryOnKeyFields` (p. 441), `wxDbTable::QueryMatching` (p. 440) or `wxDbTable::QueryBySqlStmt` (p. 439).

#### **Return value**



This function returns false when the current cursor has reached the beginning of the result set and there are now other rows prior to the cursors current position. When false is returned, data in the bound columns is undefined.

**Remarks**

This function can only be used if the datasource connection used by the wxDbTable instance was created with FwdOnlyCursors set to false. If the connection does not allow backward scrolling cursors, this function will return false, and the data contained in the bound columns will be undefined.

**See also**

*wxDb::IsFwdOnlyCursors* (p. 396), *wxDbTable::--* (p. 451)

**wxDbTable::GetQueryTableName****const wxString & GetQueryTableName()**

Accessor function that returns the name of the table/view that was indicated as being the table/view to query against when this wxDbTable instance was created.

**See also**

*wxDbTable constructor* (p. 416)

**wxDbTable::GetRowNum****UWORD GetRowNum()**

Returns the ODBC row number for performing positioned updates and deletes.

**Remarks**

This function is not being used within the ODBC class library and may be a candidate for removal if no use is found for it.

Row number with some datasources/ODBC drivers is the position in the result set, while in others it may be a physical position in the database. Check your database documentation to find out which behavior is supported.

**wxDbTable::GetTableName****const wxString & GetTableName()**

Accessor function that returns the name of the table that was indicated as being the table that this wxDbTable instance was associated with.

**wxDbTable::GetTablePath****const wxString & GetTablePath()**

Accessor function that returns the path to the data table that was indicated during creation of this `wxDbTable` instance.

### Remarks

Currently only applicable to dBase and MS-Access datasources.

### `wxDbTable::GetWhereClause`

#### `const wxString & GetWhereClause()`

Accessor function that returns the current WHERE setting assigned with `thewxDbTable::SetWhereClause` (p. 448)

### See also

`wxDbTable::Where` (p. 450)

### `wxDbTable::Insert`

#### `int Insert()`

Inserts a new record into the table being referenced by this `wxDbTable` instance. The values in the member variables of the `wxDbTable` instance are inserted into the columns of the new row in the database.

### Return value

<code>DB_SUCCESS</code>	Record inserted successfully (value = 1)
<code>DB_FAILURE</code>	Insert failed (value = 0)
<code>DB_ERR_INTEGRITY_CONSTRAINT_VIOL</code>	The insert failed due to an integrity constraint violation (duplicate
<code>non-unique</code>	index entry) is attempted.

### Remarks

A `wxDb::CommitTrans` (p. 384) or `wxDb::RollbackTrans` (p. 401) must be called after use of this function to commit or rollback the insertion.

### Example

```
// Incomplete code snippet
wxStrcpy(parts->PartName, "10");
wxStrcpy(parts->PartDesc, "Part #10");
parts->Qty = 1000;
RETCODE retcode = parts->Insert();
switch(retcode)
{
    case DB_SUCCESS:
        parts->GetDb()->CommitTrans();
```

```
        return(true);
    case DB_ERR_INTEGRITY_CONSTRAINT_VIOL:
        // Current data would result in a duplicate key
        // on one or more indexes that do not allow duplicates
        parts->GetDb()->RollbackTrans();
        return(false);
    default:
        // Insert failed for some unexpected reason
        parts->GetDb()->RollbackTrans();
        return(false);
}
```

### **wxDbTable::IsColNull**

**bool IsColNull(UWORD colNumber) const**

Used primarily in the ODBC class library to determine if a column value is set to "NULL". Works for all data types supported by the ODBC class library.

#### **Parameters**

*colNumber*

The column number of the bound column as defined by the *wxDbTable::SetColDefs* (p. 443) calls which defined the columns accessible to this *wxDbTable* instance.

#### **Remarks**

NULL column support is currently not fully implemented as of wxWidgets 2.4.

### **wxDbTable::IsCursorClosedOnCommit**

**bool IsCursorClosedOnCommit()**

Accessor function to return information collected during the opening of the datasource connection that is used by this *wxDbTable* instance. The result returned by this function indicates whether an implicit closing of the cursor is done after a commit on the database connection.

#### **Return value**

Returns true if the cursor associated with this *wxDbTable* object is closed after a commit or rollback operation. Returns false otherwise.

#### **Remarks**

If more than one *wxDbTable* instance used the same database connection, all cursors which use the database connection are closed on the commit if this function indicates true.

### **wxDbTable::IsQueryOnly**

**bool IsQueryOnly()**

Accessor function that returns a value indicating if this `wxDbTable` instance was created to allow only queries to be performed on the bound columns. If this function returns true, then no actions may be performed using this `wxDbTable` instance that would modify (insert/delete/update) the table's data.

### **wxDbTable::Open**

**bool** **Open**(**bool** *checkPrivileges=false*, **bool** *checkTableExists=true*)

Every `wxDbTable` instance must be opened before it can be used. This function checks for the existence of the requested table, binds columns, creates required cursors, (insert/select and update if connection is not `wxDB_QUERY_ONLY`) and constructs the insert statement that is to be used for inserting data as a new row in the datasource.

NOTE: To retrieve data into an opened table, the of the table must be bound to the variables in the program via call(s) to `wxDbTable::SetColDefs` (p. 443) before calling `Open()`.

See the *database classes overview* (p. 2152) for an introduction to using the ODBC classes.

#### **Parameters**

*checkPrivileges*

Indicates whether the `Open()` function should check whether the current connected user has at least `SELECT` privileges to access the table to which they are trying to open. Default is false.

*checkTableExists*

Indicates whether the `Open()` function should check whether the table exists in the database or not before opening it. Default is true.

#### **Remarks**

If the function returns a false value due to the table not existing, a log entry is recorded for the datasource connection indicating the problem that was detected when checking for table existence. Note that it is usually best for the calling routine to check for the existence of the table and for sufficient user privileges to access the table in the mode (`wxDB_QUERY_ONLY` or `!wxDB_QUERY_ONLY`) before trying to open the table for the best possible explanation as to why a table cannot be opened.

Checking the user's privileges on a table can be quite time consuming during the open phase. With most applications, the programmer already knows that the user has sufficient privileges to access the table, so this check is normally not required.

For best performance, open the table, and then use the `wxDb::TablePrivileges` (p. 404) function to check the users privileges. Passing a schema to the `TablePrivileges()` function can significantly speed up the privileges checks.

#### **See also**

*wxDdb::TableExists* (p. 403), *wxDdb::TablePrivileges* (p. 404) *wxDdbTable::SetColDefs* (p. 443)

### **wxDdbTable::OrderBy**

**const wxString & OrderBy()**

**void OrderBy(const wxString &OrderBy)**

Accessor function for the private class member *wxDdbTable::orderBy*. Can be used as a synonym for *wxDdbTable::GetOrderByClause* (p. 432) (the first form of this function) or *wxDdbTable::SetOrderByClause* (p. 447) (the second form of this function).

#### **Parameters**

*OrderBy*

A comma separated list of column names that indicate the alphabetized/numeric sorting sequence that the result set is to be returned in. If a FROM clause has also been specified, each column name specified in the ORDER BY clause should be prefaced with the table name to which the column belongs using DOT notation (TABLE\_NAME.COLUMN\_NAME).

#### **Return value**

The first form of this function returns the current value of the *wxDdbTable* member variable *::orderBy*.

The second form of the function has no return value.

#### **See also**

*wxDdbTable::GetOrderByClause* (p. 432), *wxDdbTable::SetFromClause* (p. 446)

### **wxDdbTable::Query**

**virtual bool Query(bool forUpdate=false, bool distinct=false)**

#### **Parameters**

*forUpdate*

*OPTIONAL*. Gives you the option of locking records as they are retrieved. If the RDBMS is not capable of the FOR UPDATE clause, this argument is ignored. See *wxDdbTable::CanSelectForUpdate* (p. 420) for additional information regarding this argument. Default is false.

*distinct*

*OPTIONAL*. Allows selection of only distinct values from the query (SELECT DISTINCT ... FROM ...). The notion of DISTINCT applies to all columns returned in the result set, not individual columns. Default is false.

## Remarks

This function queries records from the datasource based on the three `wxDbTable` members: "where", "orderBy", and "from". Use `wxDbTable::SetWhereClause` (p. 448) to filter on records to be retrieved (e.g. All users with a first name of "JOHN"). Use `wxDbTable::SetOrderByClause` (p. 447) to change the sequence in which records are returned in the result set from the datasource (e.g. Ordered by LAST\_NAME). Use `wxDbTable::SetFromClause` (p. 446) to allow inner joining of the base table (the one being associated with this instance of `wxDbTable`) with other tables which share a related field.

After each of these clauses are set/cleared, call `wxDbTable::Query()` to fetch the result set from the datasource.

This scheme has an advantage if you have to requery your record set frequently in that you only have to set your WHERE, ORDER BY, and FROM clauses once. Then to refresh the record set, simply call `wxDbTable::Query()` as frequently as needed.

Note that repeated calls to `wxDbTable::Query()` may tax the database server and make your application sluggish if done too frequently or unnecessarily.

The base table name is automatically prepended to the base column names in the event that the FROM clause has been set (is non-null) using `wxDbTable::SetFromClause` (p. 446).

The cursor for the result set is positioned *before* the first record in the result set after the query. To retrieve the first record, call either `wxDbTable::GetFirst` (p. 430) (only if backward scrolling cursors are available) or `wxDbTable::GetNext` (p. 432). Typically, no data from the result set is returned to the client driver until a request such as `wxDbTable::GetNext` (p. 432) is performed, so network traffic and database load are not overwhelmed transmitting data until the data is actually requested by the client. This behavior is solely dependent on the ODBC driver though, so refer to the ODBC driver's reference material for information on its behaviors.

Values in the bound columns' memory variables are undefined after executing a call to this function and remain that way until a row in the result set is requested to be returned.

The `wxDbTable::Query()` function is defined as "virtual" so that it may be overridden for application specific purposes.

Be sure to set the `wxDbTable`'s "where", "orderBy", and "from" member variables to "" if they are not to be used in the query. Otherwise, the results returned may have unexpected results (or no results) due to improper or incorrect query parameters constructed from the uninitialized clauses.

## Example

```
// Incomplete code sample
parts->SetWhereClause("DESCRIPTION = 'FOOD'");
parts->SetOrderByClause("EXPIRATION_DATE");
parts->SetFromClause("");
// Query the records based on the where, orderBy and from clauses
// specified above
parts->Query();
```

```
// Display all records queried
while(parts->GetNext())
    dispPart(parts); // user defined function
```

### **wxDbTable::QueryBySqlStmt**

#### **bool QueryBySqlStmt(const wxString &pSqlStmt)**

Performs a query against the datasource by accepting and passing verbatim the SQL SELECT statement passed to the function.

#### **Parameters**

*pSqlStmt*

Pointer to the SQL SELECT statement to be executed.

#### **Remarks**

This is the most powerful form of the query functions available. This member function allows a programmer to write their own custom SQL SELECT statement for requesting data from the datasource. This gives the programmer access to the full power of SQL for performing operations such as scalar functions, aggregate functions, table joins, and sub-queries, as well as datasource specific function calls.

The requirements of the SELECT statement are the following:

1. Must return the correct number of columns. In the derived wxDbTable constructor, it is specified how many columns are in the wxDbTable object. The SELECT statement must return exactly that many columns.
2. The columns must be returned in the same sequence as specified when defining the bounds columns *wxDbTable::SetColDefs* (p. 443), and the columns returned must be of the proper data type. For example, if column 3 is defined in the wxDbTable bound column definitions to be a float, the SELECT statement must return a float for column 3 (e.g. PRICE \* 1.10 to increase the price by 10
3. The ROWID can be included in your SELECT statement as the **last** column selected, if the datasource supports it. Use *wxDbTable::CanUpdateByROWID()* to determine if the ROWID can be selected from the datasource. If it can, much better performance can be achieved on updates and deletes by including the ROWID in the SELECT statement.

Even though data can be selected from multiple tables (joins) in your select statement, only the base table associated with this wxDbTable object is automatically updated through the ODBC class library. Data from multiple tables can be selected for display purposes however. Include columns in the wxDbTable object and mark them as non-updateable (See *wxDbColDef* (p. 406) for details). This way columns can be selected and displayed from other tables, but only the base table will be updated automatically when performed through the *wxDbTable::Update* (p. 449) function after using this type of query. To update tables other than the base table, use the *wxDbTable::Update* (p. 449) function passing a SQL statement.

After this function has been called, the cursor is positioned before the first record in the record set. To retrieve the first record, call either `wxDbTable::GetFirst` (p. 430) or `wxDbTable::GetNext` (p. 432).

### Example

```
// Incomplete code samples
wxString sqlStmt;
sqlStmt = "SELECT * FROM PARTS WHERE STORAGE_DEVICE = 'SD98' \
          AND CONTAINER = 12";
// Query the records using the SQL SELECT statement above
parts->QueryBySqlStmt(sqlStmt);
// Display all records queried
while(parts->GetNext())
    dispPart(&parts);

Example SQL statements
-----

// Table Join returning 3 columns
SELECT PART_NUM, part_desc, sd_name
    from parts, storage_devices
    where parts.storage_device_id =
           storage_devices.storage_device_id

// Aggregate function returning total number of
// parts in container 99
SELECT count(*) from PARTS where container = 99

// Order by clause; ROWID, scalar function
SELECT PART_NUM, substring(part_desc, 1, 10), qty_on_hand + 1,
ROWID
    from parts
    where warehouse = 10
    order by PART_NUM desc           // descending order

// Subquery
SELECT * from parts
    where container in (select container
                        from storage_devices
                        where device_id = 12)
```

### **wxDbTable::QueryMatching**

**virtual bool QueryMatching(bool forUpdate=false, bool distinct=false)**

`QueryMatching` allows querying of records from the table associated with the `wxDbTable` object by matching "columns" to values.

For example: To query the datasource for the row with a `PART_NUMBER` column value of "32", clear all column variables of the `wxDbTable` object, set the `PartNumber` variable that is bound to the `PART_NUMBER` column in the `wxDbTable` object to "32", and then call `wxDbTable::QueryMatching()`.

### Parameters



*forUpdate*

*OPTIONAL.* Gives you the option of locking records as they are queried (SELECT ... FOR UPDATE). If the RDBMS is not capable of the FOR UPDATE clause, this argument is ignored. See *wxDbTable::CanSelectForUpdate* (p. 420) for additional information regarding this argument. Default is false.

*distinct*

*OPTIONAL.* Allows selection of only distinct values from the query (SELECT DISTINCT ... FROM ...). The notion of DISTINCT applies to all columns returned in the result set, not individual columns. Default is false.

**Remarks**

The SQL WHERE clause is built by the ODBC class library based on all non-zero/non-NULL columns in your *wxDbTable* object. Matches can be on one, many or all of the *wxDbTable*'s columns. The base table name is prepended to the column names in the event that the *wxDbTable*'s FROM clause is non-null.

This function cannot be used to perform queries which will check for columns that are 0 or NULL, as the automatically constructed WHERE clause only will contain comparisons on column member variables that are non-zero/non-NULL.

The primary difference between this function and *wxDbTable::QueryOnKeyFields* (p. 441) is that this function can query on any column(s) in the *wxDbTable* object. Note however that this may not always be very efficient. Searching on non-indexed columns will always require a full table scan.

The cursor is positioned before the first record in the record set after the query is performed. To retrieve the first record, the program must call either *wxDbTable::GetFirst* (p. 430) or *wxDbTable::GetNext* (p. 432).

WHERE and FROM clauses specified using *wxDbTable::SetWhereClause* (p. 448) and *wxDbTable::SetFromClause* (p. 446) are ignored by this function.

**Example**

```
// Incomplete code sample
parts->ClearMemberVars();           // Set all columns to zero
wxStrncpy(parts->PartNumber,"32"); // Set columns to query on
parts->OnHold = true;
parts->QueryMatching();              // Query
// Display all records queried
while(parts->GetNext())
    dispPart(parts); // Some application defined function
```

**wxDbTable::QueryOnKeyFields**

**bool** QueryOnKeyFields(**bool** forUpdate=false,**bool** distinct=false)

QueryOnKeyFields provides an easy mechanism to query records in the table associated with the *wxDbTable* object by the primary index column(s). Simply assign the primary

index column(s) values and then call this member function to retrieve the record.

Note that since primary indexes are always unique, this function implicitly always returns a single record from the database. The base table name is prepended to the column names in the event that the `wxDbTable`'s `FROM` clause is non-null.

### Parameters

#### *forUpdate*

*OPTIONAL.* Gives you the option of locking records as they are queried (`SELECT ... FOR UPDATE`). If the RDBMS is not capable of the `FOR UPDATE` clause, this argument is ignored. See `wxDbTable::CanSelectForUpdate` (p. 420) for additional information regarding this argument. Default is false.

#### *distinct*

*OPTIONAL.* Allows selection of only distinct values from the query (`SELECT DISTINCT ... FROM ...`). The notion of `DISTINCT` applies to all columns returned in the result set, not individual columns. Default is false.

### Remarks

The cursor is positioned before the first record in the record set after the query is performed. To retrieve the first record, the program must call either `wxDbTable::GetFirst` (p. 430) or `wxDbTable::GetNext` (p. 432).

`WHERE` and `FROM` clauses specified using `wxDbTable::SetWhereClause` (p. 448) and `wxDbTable::SetFromClause` (p. 446) are ignored by this function.

### Example

```
// Incomplete code sample
wxStrcpy(parts->PartNumber, "32");
parts->QueryOnKeyFields();
// Display all records queried
while(parts->GetNext())
    dispPart(parts); // Some application defined function
```

### **wxDbTable::Refresh**

#### **bool Refresh()**

This function re-reads the bound columns into the memory variables, setting them to the current values stored on the disk.

The cursor position and result set are unaffected by calls to this function. (The one exception is in the case where the record to be refreshed has been deleted by some other user or transaction since it was originally retrieved as part of the result set. For most datasources, the default behavior in this situation is to return the value that was originally queried for the result set, even though it has been deleted from the database. But this is datasource dependent, and should be tested before relying on this behavior.)

### Remarks

This routine is only guaranteed to work if the table has a unique primary index defined for it. Otherwise, more than one record may be fetched and there is no guarantee that the correct record will be refreshed. The table's columns are refreshed to reflect the current data in the database.

### **wxDbTable::SetColDefs**

**bool SetColDefs(UWORD index, const wxString &fieldName, int dataType, void \*pData, SWORD cType, int size, bool keyField = false, bool updateable = true, bool insertAllowed = true, bool derivedColumn = false)**

**wxDbColDataPtr \* SetColDefs(wxDbColInf \*colInfs, UWORD numCols)**

#### **Parameters**

*index*

Column number (0 to n-1, where n is the number of columns specified as being defined for this wxDbTable instance when the wxDbTable constructor was called.

*fieldName*

Column name from the associated data table.

*dataType*

Logical data type. Valid logical types include:

DB_DATA_TYPE_VARCHAR	: strings
DB_DATA_TYPE_INTEGER	: non-floating point numbers
DB_DATA_TYPE_FLOAT	: floating point numbers
DB_DATA_TYPE_DATE	: dates
DB_DATA_TYPE_BLOB	: binary large objects
DB_DATA_TYPE_MEMO	: large strings

*pData*

Pointer to the data object that will hold the column's value when a row of data is returned from the datasource.

*cType*

SQL C Type. This defines the data type that the SQL representation of the data is converted to to be stored in *pData*. Other valid types are available also, but these are the most common ones:

```
SQL_C_CHAR      // string - deprecated: use SQL_C_WXCHAR
SQL_C_WXCHAR    // string - Used transparently in unicode or
non-unicode builds
SQL_C_LONG
SQL_C_ULONG
SQL_C_SHORT
SQL_C_USHORT
SQL_C_FLOAT
SQL_C_DOUBLE
```

```
SQL_C_NUMERIC
SQL_C_TIMESTAMP

SQL_C_BOOLEAN    // defined in db.h
SQL_C_ENUM       // defined in db.h
```

***size***

Maximum size in bytes of the *pData* object.

***keyField***

*OPTIONAL*. Indicates if this column is part of the primary index. Default is false.

***updateable***

*OPTIONAL*. Are updates allowed on this column? Default is true.

***insertAllowed***

*OPTIONAL*. Inserts allowed on this column? Default is true.

***derivedColumn***

*OPTIONAL*. Is this a derived column (non-base table column for query only)? Default is false.

***collNfs***

Pointer to an array of *wxDbCollNf* instances which contains all the information necessary to create *numCols* column definitions.

***numCols***

Number of elements of *wxDbCollNf* type that are pointed to by *collNfs*, which are to have column definitions created from them.

**Remarks**

If *pData* is to hold a string of characters, be sure to include enough space for the NULL terminator in *pData* and in the byte count of *size*.

Using the first form of this function, if the column definition is not able to be created, a value of false is returned. If the specified index of the column exceeds the number of columns defined in the *wxDbTable* instance, an assert is thrown and logged (in debug builds) and a false is returned.

A failure to create the column definition in the second form results in a value of NULL being returned.

Both forms of this function provide a shortcut for defining the columns in your *wxDbTable* object. Use this function in any derived *wxDbTable* constructor when describing the column/columns in the *wxDbTable* object.

The second form of this function is primarily used when the *wxDb::GetColumns* (p. 390)

function was used to query the datasource for the column definitions, so that the column definitions are already stored in `wxDboCollnf` form. One example use of using `wxDbo::GetColumns` (p. 390) then using this function is if a data table existed in one datasource, and the table's column definitions were to be copied over to another datasource or table.

### Example

```
// Long way not using this function
wxStrncpy(colDefs[0].ColName, "PART_NUM");
colDefs[0].DbType         = DB_DATA_TYPE_VARCHAR;
colDefs[0].PtrDataObj     = PartNumber;
colDefs[0].SqlCtype       = SQL_C_WXCHAR;
colDefs[0].SzDataObj      = PART_NUMBER_LEN;
colDefs[0].KeyField       = true;
colDefs[0].Updateable     = false;
colDefs[0].InsertAllowed = true;
colDefs[0].DerivedCol     = false;

// Shortcut using this function
SetColDefs(0, "PART_NUM", DB_DATA_TYPE_VARCHAR, PartNumber,
           SQL_C_WXCHAR, PART_NUMBER_LEN, true, false, true,
false);
```

### `wxDboTable::SetCursor`

**void SetCursor(HSTMT \*hstmtActivate = (void \*\*) wxDB\_DEFAULT\_CURSOR)**

#### Parameters

*hstmtActivate*

*OPTIONAL.* Pointer to the cursor that is to become the current cursor. Passing no cursor handle will reset the cursor back to the `wxDboTable`'s default (original) cursor that was created when the `wxDboTable` instance was first created. Default is `wxDB_DEFAULT_CURSOR`.

#### Remarks

When swapping between cursors, the member variables of the `wxDboTable` object are automatically refreshed with the column values of the row that the current cursor is positioned at (if any). If the cursor is not positioned, then the data in member variables is undefined.

The only way to return back to the cursor that was in use before this function was called is to programmatically determine the current cursor's HSTMT **BEFORE** calling this function using `wxDboTable::GetCursor` (p. 430) and saving a pointer to that cursor.

#### See also

`wxDboTable::GetNewCursor` (p. 431), `wxDboTable::GetCursor` (p. 430), `wxDboTable::SetCursor` (p. 445)

**wxDbTable::SetFromClause****void SetFromClause(const wxString &From)**

Accessor function for setting the private class member wxDbTable::from that indicates what other tables should be inner joined with the wxDbTable's base table for access to the columns in those other tables.

Synonym to this function is one form of *wxDbTable::From* (p. 429)

**Parameters***From*

A comma separated list of table names that are to be inner joined with the base table's columns so that the joined table's columns may be returned in the result set or used as a portion of a comparison with the base table's columns. NOTE that the base table's name must NOT be included in the FROM clause, as it is automatically included by the wxDbTable class in constructing query statements.

**Remarks**

Used by the *wxDbTable::Query* (p. 437) and *wxDbTable::Count* (p. 422) member functions to allow inner joining of records from multiple tables.

Do **not** include the keyword "FROM" when setting the FROM clause.

If using the FROM clause when performing a query, be certain to include in the corresponding WHERE clause a comparison of a column from either the base table or one of the other joined tables to each other joined table to ensure the datasource knows on which column values the tables should be joined on.

**Example**

```
...
// Base table is the "LOCATION" table, and it is being
// inner joined to the "PART" table via the field "PART_NUMBER"
// that can be related between the two tables.
location->SetWhereClause( "LOCATION.PART_NUMBER =
PART.PART_NUMBER" )
location->SetFromClause( "PART" );
...
```

**See also**

*wxDbTable::From* (p. 429), *wxDbTable::GetFromClause* (p. 431)

**wxDbTable::SetColNull****bool SetColNull(UWORD colNumber, bool set=true)****bool SetColNull(const wxString &colName, bool set=true)**

Both forms of this function allow a member variable representing a column in the table

associated with this `wxDbTable` object to be set to `NULL`.

The first form allows the column to be set by the index into the column definitions used to create the `wxDbTable` instance, while the second allows the actual column name to be specified.

### Parameters

*colNumber*

Index into the column definitions used when first defining this `wxDbTable` object.

*colName*

Actual data table column name that is to be set to `NULL`.

*set*

Whether the column is set to `NULL` or not. Passing `true` sets the column to `NULL`, passing `false` sets the column to be non-`NULL`. Default is `true`.

### Remarks

No database updates are done by this function. It only operates on the member variables in memory. Use `insert` or `update` function to store this value to disk.

### `wxDbTable::SetOrderByClause`

**`void SetOrderByClause(const wxString &OrderBy)`**

Accessor function for setting the private class member `wxDbTable::orderBy` which determines sequence/ordering of the rows returned in the result set of a query.

A synonym to this function is one form of the function `wxDbTable::OrderBy` (p. 437)

### Parameters

*OrderBy*

A comma separated list of column names that indicate the alphabetized sorting sequence that the result set is to be returned in. If a `FROM` clause has also been specified, each column name specified in the `ORDER BY` clause should be prefaced with the table name to which the column belongs using `DOT` notation (`TABLE_NAME.COLUMN_NAME`).

### Remarks

Do **not** include the keywords `"ORDER BY"` when setting the `ORDER BY` clause.

### Example

```
...
parts->SetOrderByClause("PART_DESCRIP, QUANTITY");
...
```

```
...
location->SetOrderByClause( "LOCATION.POSITION,
PART.PART_NUMBER) ;
...
```

**See also**

*wxDATABASE::OrderBy* (p. 437), *wxDATABASE::GetOrderByClause* (p. 432)

**wxDATABASE::SetQueryTimeout**

**bool SetQueryTimeout(UDWORD *nSeconds*)**

Allows a time period to be set as the timeout period for queries.

**Parameters**

*nSeconds*

The number of seconds to wait for the query to complete before timing out.

**Remarks**

Neither Oracle or Access support this function as of yet. Other databases should be evaluated for support before depending on this function working correctly.

**wxDATABASE::SetWhereClause**

**void SetWhereClause(const wxString &*Where*)**

Accessor function for setting the private class member *wxDATABASE::where* that determines which rows are returned in the result set by the datasource.

A synonym to this function is one form of the function *wxDATABASE::Where* (p. 450)

**Parameters**

*Where*

SQL "where" clause. This clause can contain any SQL language that is legal in standard where clauses. If a FROM clause has also been specified, each column name specified in the ORDER BY clause should be prefaced with the table name to which the column belongs using DOT notation (TABLE\_NAME.COLUMN\_NAME).

**Remarks**

Do **not** include the keywords "WHERE" when setting the WHERE clause.

**Example**

```
...
// Simple where clause
parts->SetWhereClause( "PART_NUMBER = '32' " );
...
```



```
// Any comparison operators
parts->SetWhereClause("PART_DESCRIP LIKE 'HAMMER%'");
...
// Multiple comparisons, including a function call
parts->Where("QTY > 0 AND {fn UCASE(PART_DESCRIP)} LIKE
'%DRILL%'");
...
// Using parameters and multiple logical combinations
parts->Where("((QTY > 10) OR (ON_ORDER > 0)) AND ON_HOLD = 0");
...
// This query uses an inner join (requiring a FROM clause also)
// that joins the PART and LOCATION table on the common field
// PART_NUMBER.
parts->Where("PART.ON_HOLD = 0 AND \
PART.PART_NUMBER = LOCATION.PART_NUMBER AND \
LOCATION.PART_NUMBER > 0");
```

**See also**

*wxDbTable::Where* (p. 450), *wxDbTable::GetWhereClause* (p. 434)

**wxDbTable::Update**

**bool Update()**

**bool Update(const wxString &pSqlStmt)**

The first form of this function will update the row that the current cursor is currently positioned at with the values in the memory variables that are bound to the columns. The actual SQL statement to perform the update is automatically created by the ODBC class, and then executed.

The second form of the function allows full access through SQL statements for updating records in the database. Write any valid SQL UPDATE statement and submit it to this function for execution. Sophisticated updates can be performed using the full power of the SQL dialect. The full SQL statement must have the exact syntax required by the driver/datasource for performing the update. This usually is in the form of:

```
UPDATE tablename SET col1=X, col2=Y, ... where ...
```

**Parameters**

*pSqlStmt*

Pointer to SQL UPDATE statement to be executed.

**Remarks**

A *wxDb::CommitTrans* (p. 384) or *wxDb::RollbackTrans* (p. 401) must be called after use of this function to commit or rollback the update.

**Example**

```
wxString sqlStmt;
sqlStmt = "update PART set QTY = 0 where PART_NUMBER = '32'";
```

**wxDbTable::UpdateWhere****bool UpdateWhere(const wxString &pWhereClause)**

Performs updates to the base table of the wxDbTable object, updating only the rows which match the criteria specified in the *pWhereClause*.

All columns that are bound to member variables for this wxDbTable instance that were defined with the "updateable" parameter set to true will be updated with the information currently held in the memory variable.

**Parameters***pWhereClause*

Pointer to a valid SQL WHERE clause. Do not include the keyword 'WHERE'.

**Remarks**

Care should be used when updating columns that are part of indexes with this function so as not to violate an unique key constraints.

A *wxDb::CommitTrans* (p. 384) or *wxDb::RollbackTrans* (p. 401) must be called after use of this function to commit or rollback the update(s).

**wxDbTable::Where****const wxString & Where()****void Where(const wxString& Where)**

Accessor function for the private class member wxDbTable::where. Can be used as a synonym for *wxDbTable::GetWhereClause* (p. 434) (the first form of this function) to return the current where clause or *wxDbTable::SetWhereClause* (p. 448) (the second form of this function) to set the where clause for this table instance.

**Parameters***Where*

A valid SQL WHERE clause. Do not include the keyword 'WHERE'.

**Return value**

The first form of this function returns the current value of the wxDbTable member variable ::where.

The second form of the function has no return value, as it will always set the where clause successfully.

**See also**

*wxDbTable::GetWhereClause* (p. 434), *wxDbTable::SetWhereClause* (p. 448)

**wxDbTable::operator ++****bool operator ++()**

Synonym for *wxDbTable::GetNext* (p. 432)

**See also**

*wxDbTable::GetNext* (p. 432)

**wxDbTable::operator --****bool operator --()**

Synonym for *wxDbTable::GetPrev* (p. 432)

**See also**

*wxDbTable::GetPrev* (p. 432)

**wxDbTableInf**

```
tableName[0]      = 0;
tableType[0]      = 0;
tableRemarks[0] = 0;
numCols          = 0;
pColInf          = NULL;
```

Currently only used by *wxDb::GetCatalog* (p. 389) internally and *wxDbInf* (p. 415) class, but may be used in future releases for user functions. Contains information describing the table (Name, type, etc). A pointer to a *wxDbColInf* array instance is included so a program can create *awxDbColInf* (p. 408) array instance (using *wxDb::GetColumns* (p. 390)) to maintain all information about the columns of a table in one memory structure.

Eventually, accessor functions will be added for this class

See the *database classes overview* (p. 2152) for an introduction to using the ODBC classes.

**Include files**

<wx/db.h>

**wxDbTableInf::Initialize**

Simply initializes all member variables to a cleared state. Called by the constructor automatically.

**wxDbGridColInfo**

This class is used to define columns to be shown, names of the columns, order and type of data, when using *wxdbGridTableBase* (p. 453) to display a Table or query in a *wxGrid* (p. 735)

See the database grid example in *wxdbGridTableBase* (p. 453) for an introduction to using the *wxDBGrid* classes.

**Include files**

<wx/dbgrid.h>

**wxDBGridCollInfo::wxDBGridCollInfo**

**wxDBGridCollInfo**(int *colNumber*, **wxString** *type*, **wxString** *title*, **wxDBGridCollInfo** \**next*)

Default constructor. See the database grid example in *wxdbGridTableBase* (p. 453) to see two different ways for adding columns.

**Parameters**

*colNumber*

Column number in the *wxDBTable* (p. 415) instance to be used (first column is 0).

*type*

Column type ,**wxString** specifying the grid name for the datatype in this column, or use **wxGRID\_VALUE\_DBAUTO** to determine the type automatically from the *wxDBColDef* (p. 406) definition

*title*

The column label to be used in the grid display

*next*

A pointer to the next *wxDBGridCollInfo* structure if using one-step construction, NULL terminates the list. Use Null also if using two step construction.

See the database grid example in *wxdbGridTableBase* (p. 453) to see two different ways for adding columns.

**wxDBGridCollInfo::~wxDBGridCollInfo**

**~wxDBGridCollInfo**()

Destructor.

**wxDBGridCollInfo::AddCollInfo**

**void AddColInfo(int colNumber,wxString type, wxString title)**

Use this member function for adding columns. See the database grid example in *wxDbGridTableBase* (p. 453) to see two different ways for adding columns.

It is important to note that this class is merely a specifier to the *wxDbGridTableBase* (p. 453) constructor. Changes made to this datatype after the *wxDbGridTableBase* (p. 453) is called will not have any effect.

**Parameters***colNumber*

Column number in the *wxDbTable* (p. 415) instance to be used (first column is 0).

*type*

Column type ,wxString specifying the grid name for the datatype in this column, or use wxGRID\_VALUE\_DBAUTO to determine the type automatically from the *wxDbColDef* (p. 406) definition

*title*

The column label to be used in the grid display

**Remarks**

As wxDbTable must be defined with to have columns which match those to by a wxDbGridColInfo info structure as this is the structure which informs the grid of how you want to display your *wxDbTable* (p. 415). If no datatype conversion or the referenced column number does not exist the the behavior is undefined.

See the example at *wxDbGridColInfo::wxDbGridColInfo* (p. 452).

## wxDbGridTableBase

You can view a database table in a grid using this class.

If you are deriving your own wxDbTable subclass for your table , then you may consider overriding GetCol() and SetCol() to provide calculated fields. This does work but care should be taken when using wxDbGridTableBase in this way.

The constructor and AssignDbTable() call allows you to specify the ownership if the wxDbTable object pointer. If you tell wxGridTableBase to take ownership , it will delete the passed wxDbTable when an new on is assigned or wxGridTableBase's destructor is called. However no checks for aliasing are done so Assign(table,...,true); Assign(table,...,true); is an error. If you need to requery an table object the preferred way is that the client keeps ownership.

**Derived From**

*wxGridTableBase* (p. 793)

**Include files**

<wx/dbgrid.h>

### Example

```
// First step, let's define wxDbTable
int numColumns = 2;
wxDbTable *table = new wxDbTable (db, tblName, numColumns);
int int_var;
wxChar string_name[255];
table->SetColDef (0, "column 0", DB_DATA_TYPE_INTEGER,
&int_var,
                SQL_C_LONG, sizeof(int_var), true);
table->SetColDef (1, "column 1", DB_DATA_TYPE_VARCHAR,
&string_name,
                SQL_C_LONG, sizeof(string_name), false);

// now let's define columns in the grid

// first way to do it
wxDbGridColInfo *columns;
columns = new wxDbGridColInfo(0, wxGRID_VALUE_LONG, "first
column",
                new wxDbGridColInfo(1, wxGRID_VALUE_STRING, "second
column",
                NULL));

// second way to do it
wxDbGridColInfo *columns;
// first column is special
columns = new wxDbGridColInfo(0, wxGRID_VALUE_LONG, "first
column", NULL);
// all the rest
columns->AddColInfo (1, wxGRID_VALUE_STRING, "second column");

// second way may be better when columns are not known at compile
time

// now, let's open the table and make a Query()
table->Open();
// this step is very important
table->SetRowMode (wxDbTable::WX_ROW_MODE_QUERY);
// in the grid we will see only the rows of the result query
m_dbTable->Query();

wxDbGridTableBase *dbgrid = new wxDbGridTableBase(table, columns,
wxUSE_QUERY, true);
delete columns; // not needed anymore
wxGrid *grid = new wxGrid ( ... );
grid->SetTable(dbgrid, true);
grid->Fit();
```

### Include files

<wx/dbgrid.h>

### Helper classes and data structures

**wxDbGridTableBase::wxDbGridTableBase**

**wxDbGridTableBase**(wxDbTable \**tab*, wxDbGridCollInfo \**CollInfo*, int *count* = wxUSE\_QUERY, bool *takeOwnership* = true)

Constructor.

**Parameters**

*tab*

The database table you want to display. Must be opened and queried before display the grid. See the example *above* (p. 453).

*CollInfo*

Columns titles, and other values. See *wxDbGridCollInfo* (p. 451).

*count*

You can use a query result set (wxUSE\_QUERY, to use wxDbTable::Count(wxDbTable::Count() or you can fix the total number of rows (count >= 0) to display, or specify it if you already know the size in avoid calling

*takeOwnership*

If true, this class deletes wxDbTable when it stops referring to it, if false application must take care of deleting it.

**wxDbGridTableBase::ValidateRow**

**void ValidateRow**(int *row*)

It ensures that the row data is fetched from the database, and if the wxDbTable local buffer, the row number passed should be the grid row.

**Parameters**

*row*

Row where validation must be done.

**wxDbGridTableBase::UpdateRow**

**bool UpdateRow**(int *row*)

If row has changed it forces that row to be written back to the database, however support for detecting whether insert/update is required is currently not in wxDbTable, so this function is currently unsupported.

**Parameters**

*row*

Row you want to update.

### **wxDbGridTableBase::AssignDbTable**

**bool AssignDbTable(wxDbTable \*tab, int count = wxUSE\_QUERY, bool takeOwnership = true)**

Resets the grid for using with a new database table, but using the same columns definition. This can be useful when re-querying the database and want to see the changes.

#### **Parameters**

*tab*

Database table you want to assign to the grid.

*count*

Number of rows you want to show or wxUSE\_QUERY for using a query.

*takeOwnership*

If false, user must take care of deleting tab after deleting the wxDbGridTableBase. If true, deletion is made by destructor class.

## **wxDC**

A wxDC is a *device context* onto which graphics and text can be drawn. It is intended to represent a number of output devices in a generic way, so a window can have a device context associated with it, and a printer also has a device context. In this way, the same piece of code may write to a number of different devices, if the device context is used as a parameter.

Notice that wxDC is an abstract base class and can't be created directly, please use *wxPaintDC* (p. 1164), *wxClientDC* (p. 193), *wxWindowDC* (p. 1855), *wxScreenDC* (p. 1407), *wxMemoryDC* (p. 1069) or *wxPrinterDC* (p. 1213).

Please note that in addition to the versions of the methods documented here, there are also versions which accept single *wxPoint* parameter instead of two *wxCoord* ones or *wxPoint* and *wxSize* instead of four of them.

### **Support for Transparency / Alpha Channel**

On Mac OS X when using Core Graphics (wx\_MAC\_USE\_CORE\_GRAPHICS set to 1) colors with alpha are supported, so instances *wxPen* or *wxBrush* that are built from *wxColour* use the color's alpha values when stroking or filling.

#### **Derived from**

*wxObject* (p. 1148)

#### **Include files**



<wx/dc.h>

### See also

Overview (p. 2120)

## wxDC::Blit

**bool Blit**(**wxCoord** *xdest*, **wxCoord** *ydest*, **wxCoord** *width*, **wxCoord** *height*, **wxDC\*** *source*, **wxCoord** *xsrc*, **wxCoord** *ysrc*, **int** *logicalFunc* = *wxCOPY*, **bool** *useMask* = *false*, **wxCoord** *xsrcMask* = -1, **wxCoord** *ysrcMask* = -1)

Copy from a source DC to this DC, specifying the destination coordinates, size of area to copy, source DC, source coordinates, logical function, whether to use a bitmap mask, and mask source position.

### Parameters

*xdest*

Destination device context x position.

*ydest*

Destination device context y position.

*width*

Width of source area to be copied.

*height*

Height of source area to be copied.

*source*

Source device context.

*xsrc*

Source device context x position.

*ysrc*

Source device context y position.

*logicalFunc*

Logical function to use: see *wxDC::SetLogicalFunction* (p. 474).

*useMask*

If true, Blit does a transparent blit using the mask that is associated with the bitmap

selected into the source device context. The Windows implementation does the following if MaskBlt cannot be used:

1. Creates a temporary bitmap and copies the destination area into it.
2. Copies the source area into the temporary bitmap using the specified logical function.
3. Sets the masked area in the temporary bitmap to BLACK by ANDing the mask bitmap with the temp bitmap with the foreground colour set to WHITE and the bg colour set to BLACK.
4. Sets the unmasked area in the destination area to BLACK by ANDing the mask bitmap with the destination area with the foreground colour set to BLACK and the background colour set to WHITE.
5. ORs the temporary bitmap with the destination area.
6. Deletes the temporary bitmap.

This sequence of operations ensures that the source's transparent area need not be black, and logical functions are supported.

**Note:** on Windows, blitting with masks can be speeded up considerably by compiling wxWidgets with the wxUSE\_DC\_CACHE option enabled. You can also influence whether MaskBlt or the explicit mask blitting code above is used, by using *wxSystemOptions* (p. 1590) and setting the **no-maskblt** option to 1.

#### *xsrcMask*

Source x position on the mask. If both *xsrcMask* and *ysrcMask* are -1, *xsrc* and *ysrc* will be assumed for the mask source position. Currently only implemented on Windows.

#### *ysrcMask*

Source y position on the mask. If both *xsrcMask* and *ysrcMask* are -1, *xsrc* and *ysrc* will be assumed for the mask source position. Currently only implemented on Windows.

#### **Remarks**

There is partial support for Blit in wxPostScriptDC, under X.

See *wxMemoryDC* (p. 1069) for typical usage.

#### **See also**

*wxMemoryDC* (p. 1069), *wxBitmap* (p. 123), *wxMask* (p. 1036)

#### **wxDC::CalcBoundingBox**

**void CalcBoundingBox(wxCoord x, wxCoord y)**

Adds the specified point to the bounding box which can be retrieved with *MinX* (p. 471), *MaxX* (p. 471) and *MinY* (p. 471), *MaxY* (p. 471) functions.

**See also**

*ResetBoundingBox* (p. 472)

**wxDC::Clear****void Clear()**

Clears the device context using the current background brush.

**wxDC::ComputeScaleAndOrigin****virtual void ComputeScaleAndOrigin()**

Performs all necessary computations for given platform and context type after each change of scale and origin parameters. Usually called automatically internally after such changes.

**wxDC::CrossHair****void CrossHair(wxCoord x, wxCoord y)**

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

**wxDC::DestroyClippingRegion****void DestroyClippingRegion()**

Destroys the current clipping region so that none of the DC is clipped. See also *wxDC::SetClippingRegion* (p. 473).

**wxDC::DeviceToLogicalX****wxCoord DeviceToLogicalX(wxCoord x)**

Convert device X coordinate to logical coordinate, using the current mapping mode.

**wxDC::DeviceToLogicalXRel****wxCoord DeviceToLogicalXRel(wxCoord x)**

Convert device X coordinate to relative logical coordinate, using the current mapping mode but ignoring the x axis orientation. Use this function for converting a width, for example.

**wxDC::DeviceToLogicalY****wxCoord DeviceToLogicalY(wxCoord y)**

Converts device Y coordinate to logical coordinate, using the current mapping mode.

**wxDC::DeviceToLogicalYRel****wxCoord DeviceToLogicalYRel(wxCoord y)**

Convert device Y coordinate to relative logical coordinate, using the current mapping mode but ignoring the y axis orientation. Use this function for converting a height, for example.

**wxDC::DrawArc****void DrawArc(wxCoord x1, wxCoord y1, wxCoord x2, wxCoord y2, wxCoord xc, wxCoord yc)**

Draws an arc of a circle, centred on (xc, yc), with starting point (x1, y1) and ending at (x2, y2). The current pen is used for the outline and the current brush for filling the shape.

The arc is drawn in an anticlockwise direction from the start point to the end point.

**wxDC::DrawBitmap****void DrawBitmap(const wxBitmap& bitmap, wxCoord x, wxCoord y, bool transparent)**

Draw a bitmap on the device context at the specified point. If *transparent* is true and the bitmap has a transparency mask, the bitmap will be drawn transparently.

When drawing a mono-bitmap, the current text foreground colour will be used to draw the foreground of the bitmap (all bits set to 1), and the current text background colour to draw the background (all bits set to 0). See also *SetTextForeground* (p. 475), *SetTextBackground* (p. 475) and *wxMemoryDC* (p. 1069).

**wxDC::DrawCheckMark****void DrawCheckMark(wxCoord x, wxCoord y, wxCoord width, wxCoord height)****void DrawCheckMark(const wxRect &rect)**

Draws a check mark inside the given rectangle.

**wxDC::DrawCircle****void DrawCircle(wxCoord x, wxCoord y, wxCoord radius)****void DrawCircle(const wxPoint& pt, wxCoord radius)**

Draws a circle with the given centre and radius.

**See also**

*DrawEllipse* (p. 461)

**wxDC::DrawEllipse**

**void DrawEllipse(wxCoord x, wxCoord y, wxCoord width, wxCoord height)**

**void DrawEllipse(const wxPoint& pt, const wxSize& size)**

**void DrawEllipse(const wxRect& rect)**

Draws an ellipse contained in the rectangle specified either with the given top left corner and the given size or directly. The current pen is used for the outline and the current brush for filling the shape.

**See also**

*DrawCircle* (p. 460)

**wxDC::DrawEllipticArc**

**void DrawEllipticArc(wxCoord x, wxCoord y, wxCoord width, wxCoord height,  
double start, double end)**

Draws an arc of an ellipse. The current pen is used for drawing the arc and the current brush is used for drawing the pie.

*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(const wxIcon& icon, wxCoord x, wxCoord y)**

Draw an icon on the display (does nothing if the device context is PostScript). This can be the simplest way of drawing bitmaps on a window.

**wxDC::DrawLabel**

**virtual void DrawLabel(const wxString& text, const wxBitmap& image,  
const wxRect& rect, int alignment = wxALIGN\_LEFT | wxALIGN\_TOP,  
int indexAccel = -1, wxRect \*rectBounding = NULL)**

**void DrawLabel(const wxString& text, const wxRect& rect, int alignment**

`= wxALIGN_LEFT | wxALIGN_TOP, int indexAccel = -1)`

Draw optional bitmap and the text into the given rectangle and aligns it as specified by alignment parameter; it also will emphasize the character with the given index if it is != -1 and return the bounding rectangle if required.

### **wxDC::DrawLine**

**void DrawLine(wxCoord x1, wxCoord y1, wxCoord x2, wxCoord y2)**

Draws a line from the first point to the second. The current pen is used for drawing the line. Note that the point (x2, y2) is *not* part of the line and is not drawn by this function (this is consistent with the behaviour of many other toolkits).

### **wxDC::DrawLines**

**void DrawLines(int n, wxPoint points[], wxCoord xoffset = 0, wxCoord yoffset = 0)**

**void DrawLines(wxList \*points, wxCoord xoffset = 0, wxCoord yoffset = 0)**

Draws lines using an array of *points* of size *n*, or list of pointers to points, adding the optional offset coordinate. The current pen is used for drawing the lines. The programmer is responsible for deleting the list of points.

**wxPython note:** The wxPython version of this method accepts a Python list of wxPoint objects.

**wxPerl note:** The wxPerl version of this method accepts as its first parameter a reference to an array of wxPoint objects.

### **wxDC::DrawPolygon**

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

**void DrawPolygon(wxList \*points, wxCoord xoffset = 0, wxCoord 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 wxWidgets automatically closes the first and last points.

**wxPython note:** The wxPython version of this method accepts a Python list of wxPoint objects.

**wxPerl note:** The wxPerl version of this method accepts `as` as its first parameter a reference to an array of wxPoint objects.

### **wxDC::DrawPolyPolygon**

```
void DrawPolyPolygon(int n, int count[], wxPoint points[], wxCoord xoffset = 0,  
wxCoord yoffset = 0,  
    int fill_style = wxODDEVEN_RULE)
```

Draws two or more filled polygons using an array of *points*, adding the optional offset coordinates.

Notice that for the platforms providing a native implementation of this function (Windows and PostScript-based wxDC currently), this is more efficient than using *DrawPolygon* (p. 462) in a loop.

*n* specifies the number of polygons to draw, the array *count* of size *n* specifies the number of points in each of the polygons in the *points* array.

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 polygons maybe disjoint or overlapping. Each polygon specified in a call to **DrawPolyPolygon** must be closed. Unlike polygons created by the *DrawPolygon* (p. 462) member function, the polygons created by **DrawPolyPolygon** are not closed automatically.

**wxPython note:** Not implemented yet

**wxPerl note:** Not implemented yet

### **wxDC::DrawPoint**

```
void DrawPoint(wxCoord x, wxCoord y)
```

Draws a point using the color of the current pen. Note that the other properties of the pen are not used, such as width etc..

### **wxDC::DrawRectangle**

```
void DrawRectangle(wxCoord x, wxCoord y, wxCoord width, wxCoord 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::DrawRotatedText**

```
void DrawRotatedText(const wxString& text, wxCoord x, wxCoord y, double angle)
```

Draws the text rotated by *angle* degrees.

**NB:** Under Win9x only TrueType fonts can be drawn by this function. In particular, a font different from `wxNORMAL_FONT` should be used as the latter is not a TrueType font.

`wxSWISS_FONT` is an example of a font which is.

**See also**

*DrawText* (p. 464)

**wxDC::DrawRoundedRectangle**

**void DrawRoundedRectangle(wxCoord x, wxCoord y, wxCoord width, wxCoord height, double radius)**

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(int n, wxPoint points[])**

Draws a spline between all given control points, using the current pen.

**void DrawSpline(wxList \*points)**

Draws a spline between all given control points, using the current pen. Doesn't delete the `wxList` and contents.

**void DrawSpline(wxCoord x1, wxCoord y1, wxCoord x2, wxCoord y2, wxCoord x3, wxCoord y3)**

Draws a three-point spline using the current pen.

**wxPython note:** The wxPython version of this method accepts a Python list of `wxPoint` objects.

**wxPerl note:** The wxPerl version of this method accepts a reference to an array of `wxPoint` objects.

**wxDC::DrawText**

**void DrawText(const wxString& text, wxCoord x, wxCoord y)**

Draws a text string at the specified point, using the current text font, and the current text



foreground and background colours.

The coordinates refer to the top-left corner of the rectangle bounding the string. See `wxDC::GetTextExtent` (p. 469) for how to get the dimensions of a text string, which can be used to position the text more precisely.

**NB:** under wxGTK the current *logical function* (p. 467) is used by this function but it is ignored by wxMSW. Thus, you should avoid using logical functions with this function in portable programs.

### **wxDC::EndDoc**

**void EndDoc()**

Ends a document (only relevant when outputting to a printer).

### **wxDC::EndPage**

**void EndPage()**

Ends a document page (only relevant when outputting to a printer).

### **wxDC::FloodFill**

**bool FloodFill(wxCoord x, wxCoord y, const wxColour& colour, int style=wxFLOOD\_SURFACE)**

Flood fills the device context starting from the given point, using the *current brush 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.

Returns false if the operation failed.

*Note:* The present implementation for non-Windows platforms may fail to find colour borders if the pixels do not match the colour exactly. However the function will still return true.

### **wxDC::GetBackground**

**const wxBrush& GetBackground() const**

Gets the brush used for painting the background (see `wxDC::SetBackground` (p. 472)).

### **wxDC::GetBackgroundMode**

**int GetBackgroundMode() const**

Returns the current background mode: `wxSOLID` or `wxTRANSPARENT`.

**See also**

*SetBackgroundMode* (p. 472)

**wxDC::GetBrush**

**const wxBrush& GetBrush() const**

Gets the current brush (see *wxDC::SetBrush* (p. 472)).

**wxDC::GetCharHeight**

**wxCoord GetCharHeight()**

Gets the character height of the currently set font.

**wxDC::GetCharWidth**

**wxCoord GetCharWidth()**

Gets the average character width of the currently set font.

**wxDC::GetClippingBox**

**void GetClippingBox(wxCoord \*x, wxCoord \*y, wxCoord \*width, wxCoord \*height)**

Gets the rectangle surrounding the current clipping region.

**wxPython note:** No arguments are required and the four values defining the rectangle are returned as a tuple.

**wxPerl note:** This method takes no arguments and returns a four element list ( `x`, `y`, `width`, `height` )

**wxDC::GetFont**

**const wxFont& GetFont() const**

Gets the current font. Notice that even although each device context object has some default font after creation, this method would return a `wxNullFont` initially and only after calling *wxDC::SetFont* (p. 473) a valid font is returned.

**wxDC::GetLayoutDirection**

**wxLayoutDirection GetLayoutDirection() const**

Gets the current layout direction of the device context. On platforms where RTL layout is supported, the return value will either be `wxLayout_LeftToRight` or `wxLayout_RightToLeft`. If RTL layout is not supported, the return value will be

`wxLayout_Default`.

**See also**

*SetLayoutDirection* (p. 473)

**wxDC::GetLogicalFunction**

**int GetLogicalFunction()**

Gets the current logical function (see *wxDC::SetLogicalFunction* (p. 474)).

**wxDC::GetMapMode**

**int GetMapMode()**

Gets the *mapping mode* for the device context (see *wxDC::SetMapMode* (p. 474)).

**wxDC::GetMultiLineTextExtent**

**void GetMultiLineTextExtent(const wxString& string, wxCoord \*w,  
wxCoord \*h, wxCoord \*heightLine = NULL, wxFont \*font = NULL) const**

**wxSize GetMultiLineTextExtent(const wxString& string) const**

Gets the dimensions of the string using the currently selected font. *string* is the text string to measure, *heightLine*, if non NULL, is where to store the height of a single line.

The text extent is returned in *w* and *h* pointers (first form) or as a *wxSize* (p. 1440) object (second form).

If the optional parameter *font* is specified and valid, then it is used for the text extent calculation. Otherwise the currently selected font is.

Note that this function works both with single-line and multi-line strings.

**See also**

*wxFont* (p. 655), *wxDC::SetFont* (p. 473), *wxDC::GetPartialTextExtents* (p. 467), *wxDC::GetTextExtent* (p. 469)

**wxDC::GetPartialTextExtents**

**bool GetPartialTextExtents(const wxString& text, wxArrayInt& widths) const**

Fills the *widths* array with the widths from the beginning of *text* to the corresponding character of *text*. The generic version simply builds a running total of the widths of each character using *GetTextExtent* (p. 469), however if the various platforms have a native API function that is faster or more accurate than the generic implementation then it should be used instead.

**See also**

*wxDC::GetMultiLineTextExtent* (p. 467), *wxDC::GetTextExtent* (p. 469)

**wxPython note:** This method only takes the *text* parameter and returns a Python list of integers.

### **wxDC::GetPen**

**const wxPen& GetPen() const**

Gets the current pen (see *wxDC::SetPen* (p. 475)).

### **wxDC::GetPixel**

**bool GetPixel(wxCoord x, wxCoord y, wxColour \*colour)**

Gets in *colour* the colour at the specified location. Not available for *wxPostScriptDC* or *wxMetafileDC*.

Note that setting a pixel can be done using *DrawPoint* (p. 463).

**wxPython note:** For wxPython the *wxColour* value is returned and is not required as a parameter.

**wxPerl note:** This method only takes the parameters *x* and *y* and returns a *Wx::Colour* value

### **wxDC::GetPPI**

**wxSize GetPPI() const**

Returns the resolution of the device in pixels per inch.

### **wxDC::GetSize**

**void GetSize(wxCoord \*width, wxCoord \*height) const**

**wxSize GetSize() const**

This gets the horizontal and vertical resolution in device units. It can be used to scale graphics to fit the page. 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:

```
wxCoord w, h;  
dc.GetSize(&w, &h);  
double scaleX=(double)(maxX/w);  
double scaleY=(double)(maxY/h);  
dc.SetUserScale(min(scaleX,scaleY),min(scaleX,scaleY));
```

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

<b>GetSize()</b>	Returns a <code>wxSize</code>
<b>GetSizeTuple()</b>	Returns a 2-tuple (width, height)

**wxPerl note:** In place of a single overloaded method, wxPerl uses:

<b>GetSize()</b>	Returns a <code>Wx::Size</code>
<b>GetSizeWH()</b>	Returns a 2-element list ( <code>width</code> , <code>height</code> )

### **wxDC::GetSizeMM**

**void GetSizeMM(wxCoord \*width, wxCoord \*height) const**

**wxSize GetSizeMM() const**

Returns the horizontal and vertical resolution in millimetres.

### **wxDC::GetTextBackground**

**const wxColour& GetTextBackground() const**

Gets the current text background colour (see `wxDC::SetTextBackground` (p. 475)).

### **wxDC::GetTextExtent**

**void GetTextExtent(const wxString& string, wxCoord \*w, wxCoord \*h,  
wxCoord \*descent = NULL, wxCoord \*externalLeading = NULL, wxFont \*font = NULL)  
const**

**wxSize GetTextExtent(const wxString& string) const**

Gets the dimensions of the string using the currently selected font. *string* is the text string to measure, *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 text extent is returned in *w* and *h* pointers (first form) or as a `wxSize` (p. 1440) object (second form).

If the optional parameter *font* is specified and valid, then it is used for the text extent calculation. Otherwise the currently selected font is.

Note that this function only works with single-line strings.

### **See also**

`wxFont` (p. 655), `wxDC::SetFont` (p. 473), `wxDC::GetPartialTextExtents` (p. 467),  
`wxDC::GetMultiLineTextExtent` (p. 467)

**wxPython note:** The following methods are implemented in wxPython:

**GetTextExtent(string)** Returns a 2-tuple, (width, height)

**GetFullTextExtent(string, font=NULL)** Returns a 4-tuple, (width, height, descent, externalLeading)

**wxPerl note:** In wxPerl this method is implemented as **GetTextExtent( string, font = undef )** returning a four element array ( width, height, descent, externalLeading )

### **wxDC::GetTextForeground**

**const wxColour& GetTextForeground() const**

Gets the current text foreground colour (see *wxDC::SetTextForeground* (p. 475)).

### **wxDC::GetUserScale**

**void GetUserScale(double \*x, double \*y)**

Gets the current user scale factor (set by *SetUserScale* (p. 476)).

**wxPerl note:** In wxPerl this method takes no arguments and return a two element array ( x, y )

### **wxDC::GradientFillConcentric**

**void GradientFillConcentric(const wxRect& rect, const wxColour& initialColour, const wxColour& destColour)**

**void GradientFillConcentric(const wxRect& rect, const wxColour& initialColour, const wxColour& destColour, const wxPoint& circleCenter)**

Fill the area specified by *rect* with a radial gradient, starting from *initialColour* at the centre of the circle and fading to *destColour* on the circle outside.

*circleCenter* are the relative coordinates of centre of the circle in the specified *rect*. If not specified, the circle is placed at the centre of *rect*.

**Note:** Currently this function is very slow, don't use it for real-time drawing.

### **wxDC::GradientFillLinear**

**void GradientFillLinear(const wxRect& rect, const wxColour& initialColour, const wxColour& destColour, wxDirection nDirection = wxEAST)**

Fill the area specified by *rect* with a linear gradient, starting from *initialColour* and eventually fading to *destColour*. The *nDirection* specifies the direction of the colour change, default is to use *initialColour* on the left part of the rectangle and *destColour* on the right one.

**wxDC::LogicalToDeviceX****wxCoord LogicalToDeviceX(wxCoord x)**

Converts logical X coordinate to device coordinate, using the current mapping mode.

**wxDC::LogicalToDeviceXRel****wxCoord LogicalToDeviceXRel(wxCoord x)**

Converts logical X coordinate to relative device coordinate, using the current mapping mode but ignoring the x axis orientation. Use this for converting a width, for example.

**wxDC::LogicalToDeviceY****wxCoord LogicalToDeviceY(wxCoord y)**

Converts logical Y coordinate to device coordinate, using the current mapping mode.

**wxDC::LogicalToDeviceYRel****wxCoord LogicalToDeviceYRel(wxCoord y)**

Converts logical Y coordinate to relative device coordinate, using the current mapping mode but ignoring the y axis orientation. Use this for converting a height, for example.

**wxDC::MaxX****wxCoord MaxX()**

Gets the maximum horizontal extent used in drawing commands so far.

**wxDC::MaxY****wxCoord MaxY()**

Gets the maximum vertical extent used in drawing commands so far.

**wxDC::MinX****wxCoord MinX()**

Gets the minimum horizontal extent used in drawing commands so far.

**wxDC::MinY****wxCoord MinY()**

Gets the minimum vertical extent used in drawing commands so far.

**wxDC::IsOk****bool Ok()**

Returns true if the DC is ok to use.

**wxDC::ResetBoundingBox****void ResetBoundingBox()**

Resets the bounding box: after a call to this function, the bounding box doesn't contain anything.

**See also**

*CalcBoundingBox* (p. 458)

**wxDC::SetAxisOrientation****void SetAxisOrientation(bool xLeftRight, bool yBottomUp)**

Sets the x and y axis orientation (i.e., the direction from lowest to highest values on the axis). The default orientation is x axis from left to right and y axis from top down.

**Parameters***xLeftRight*

True to set the x axis orientation to the natural left to right orientation, false to invert it.

*yBottomUp*

True to set the y axis orientation to the natural bottom up orientation, false to invert it.

**wxDC::SetBackground****void SetBackground(const wxBrush& brush)**

Sets the current background brush for the DC.

**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::SetBrush****void SetBrush(const wxBrush& brush)**



Sets the current brush for the DC.

If the argument is `wxNullBrush`, the current brush is selected out of the device context, and the original brush restored, allowing the current brush to be destroyed safely.

See also *wxBrush* (p. 150).

See also *wxMemoryDC* (p. 1069) for the interpretation of colours when drawing into a monochrome bitmap.

### **wxDC::SetClippingRegion**

**void SetClippingRegion(wxCoord x, wxCoord y, wxCoord width, wxCoord height)**

**void SetClippingRegion(const wxPoint& pt, const wxSize& sz)**

**void SetClippingRegion(const wxRect& rect)**

**void SetClippingRegion(const wxRegion& region)**

Sets the clipping region for this device context to the intersection of the given region described by the parameters of this method and the previously set clipping region. You should call *DestroyClippingRegion* (p. 459) if you want to set the clipping region exactly to the region specified.

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

#### **See also**

*wxDC::DestroyClippingRegion* (p. 459), *wxRegion* (p. 1264)

### **wxDC::SetDeviceOrigin**

**void SetDeviceOrigin(wxCoord x, wxCoord 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::SetFont**

**void SetFont(const wxFont& font)**

Sets the current font for the DC. It must be a valid font, in particular you should not pass `wxNullFont` to this method.

See also *wxFont* (p. 655).

### **wxDC::SetLayoutDirection**

**void SetLayoutDirection(wxLayoutDirection dir)**

Sets the current layout direction for the device context. *dir* may be either `wxLayout_Default`, `wxLayout_LeftToRight` or `wxLayout_RightToLeft`.

**See also**

*GetLayoutDirection* (p. 466)

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

Sets the current logical function for the device context. This determines how a source pixel (from a pen or brush colour, or source device context if using *wxDC::Blit* (p. 457)) 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. `wxINVERT` 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* (p. 476)) scales the text appropriately. In Windows, scalable TrueType fonts are always used; in X, results depend on availability of fonts, but usually a reasonable match is found.

The coordinate origin is always at the top left of the screen/printer.

Drawing to a Windows printer device context uses the current mapping mode, but mapping mode is currently ignored for PostScript output.

The mapping mode can be one of the following:

<code>wxMM_TWIPS</code>	Each logical unit is 1/20 of a point, or 1/1440 of an inch.
<code>wxMM_POINTS</code>	Each logical unit is a point, or 1/72 of an inch.
<code>wxMM_METRIC</code>	Each logical unit is 1 mm.
<code>wxMM_LOMETRIC</code>	Each logical unit is 1/10 of a mm.
<code>wxMM_TEXT</code>	Each logical unit is 1 pixel.

### **wxDC::SetPalette**

**void SetPalette(const wxPalette& palette)**

If this is a window DC or memory DC, assigns the given palette to the window or bitmap associated with the DC. If the argument is `wxNullPalette`, the current palette is selected out of the device context, and the original palette restored.

See *wxPalette* (p. 1166) for further details.

### **wxDC::SetPen**

**void SetPen(const wxPen& pen)**

Sets the current pen for the DC.

If the argument is `wxNullPen`, the current pen is selected out of the device context, and the original pen restored.

See also *wxMemoryDC* (p. 1069) for the interpretation of colours when drawing into a monochrome bitmap.

### **wxDC::SetTextBackground**

**void SetTextBackground(const wxColour& colour)**

Sets the current text background colour for the DC.

### **wxDC::SetTextForeground**

**void SetTextForeground(const wxColour& colour)**

Sets the current text foreground colour for the DC.

See also *wxMemoryDC* (p. 1069) for the interpretation of colours when drawing into a

monochrome bitmap.

### **wxDC::SetUserScale**

**void SetUserScale(double xScale, double yScale)**

Sets the user scaling factor, useful for applications which require 'zooming'.

### **wxDC::StartDoc**

**bool StartDoc(const wxString& message)**

Starts a document (only relevant when outputting to a printer). Message is a message to show while printing.

### **wxDC::StartPage**

**bool StartPage()**

Starts a document page (only relevant when outputting to a printer).

## **wxDCClipper**

wxDCClipper is a small helper class for setting a clipping region on a wxDC (p. 456) and unsetting it automatically. An object of wxDCClipper class is typically created on the stack so that it is automatically destroyed when the object goes out of scope. A typical usage example:

```
void MyFunction(wxDC& dc)
{
    wxDCClipper clip(rect);
    ... drawing functions here are affected by clipping rect ...
}

void OtherFunction()
{
    wxDC dc;
    MyFunction(dc);
    ... drawing functions here are not affected by clipping
rect ...
}
```

### **Derived from**

No base class

### **Include files**

<wx/dc.h>

### **See also**

*wxDC::SetClippingRegion* (p. 473)

### **wxDCClipper::wxDCClipper**

**wxDCClipper(wxDC& dc, const wxRegion& r)**

**wxDCClipper(wxDC& dc, const wxRect& rect)**

**wxDCClipper(wxDC& dc, int x, int y, int w, int h)**

Sets the clipping region to the specified region *r* or rectangle specified by either a single *rect* parameter or its position (*x* and *y*) and size (*w* and *h*).

The clipping region is automatically unset when this object is destroyed.

## **wxDDEClient**

A wxDDEClient object represents the client part of a client-server DDE (Dynamic Data Exchange) conversation.

To create a client which can communicate with a suitable server, you need to derive a class from wxDDEConnection and another from wxDDEClient. The custom wxDDEConnection class will intercept communications in a 'conversation' with a server, and the custom wxDDEServer is required so that a user-overridden *wxDDEClient::OnMakeConnection* (p. 478) member can return a wxDDEConnection of the required class, when a connection is made.

This DDE-based implementation is available on Windows only, but a platform-independent, socket-based version of this API is available using *wxTCPClient* (p. 1609).

### **Derived from**

wxClientBase  
*wxObject* (p. 1148)

### **Include files**

<wx/dde.h>

### **See also**

*wxDDEServer* (p. 482), *wxDDEConnection* (p. 478), *Interprocess communications overview* (p. 2175)

### **wxDDEClient::wxDDEClient**

**wxDDEClient()**

Constructs a client object.

### **wxDDEClient::MakeConnection**

**wxConnectionBase \* MakeConnection**(const wxString& *host*, const wxString& *service*, const wxString& *topic*)

Tries to make a connection with a server specified by the *host* (machine name under UNIX, ignored under Windows), *service* name (must contain an integer port number under UNIX), and *topic* string. If the server allows a connection, a *wxDDEConnection* object will be returned. The type of *wxDDEConnection* returned can be altered by overriding the *wxDDEClient::OnMakeConnection* (p. 478) member to return your own derived connection object.

### **wxDDEClient::OnMakeConnection**

**wxConnectionBase \* OnMakeConnection**()

The type of *wxDDEConnection* (p. 478) returned from a *wxDDEClient::MakeConnection* (p. 478) call can be altered by deriving the **OnMakeConnection** member to return your own derived connection object. By default, a *wxDDEConnection* object is returned.

The advantage of deriving your own connection class is that it will enable you to intercept messages initiated by the server, such as *wxDDEConnection::OnAdvise* (p. 278). You may also want to store application-specific data in instances of the new class.

### **wxDDEClient::ValidHost**

**bool ValidHost**(const wxString& *host*)

Returns `true` if this is a valid host name, `false` otherwise. This always returns `true` under MS Windows.

## **wxDDEConnection**

A *wxDDEConnection* object represents the connection between a client and a server. It can be created by making a connection using a *wxDDEClient* (p. 477) object, or by the acceptance of a connection by a *wxDDEServer* (p. 482) object. The bulk of a DDE (Dynamic Data Exchange) conversation is controlled by calling members in a **wxDDEConnection** object or by overriding its members.

An application should normally derive a new connection class from *wxDDEConnection*, in order to override the communication event handlers to do something interesting.

This DDE-based implementation is available on Windows only, but a platform-independent, socket-based version of this API is available using *wxTCPConnection* (p. 1610).

### **Derived from**

`wxConnectionBase`  
*wxObject* (p. 1148)

### Include files

`<wx/dde.h>`

### Types

`wxIPCFormat` is defined as follows:

```
enum wxIPCFormat
{
    wxIPC_INVALID = 0,
    wxIPC_TEXT = 1, /* CF_TEXT */
    wxIPC_BITMAP = 2, /* CF_BITMAP */
    wxIPC_METAFILE = 3, /* CF_METAFILEPICT */
    wxIPC_SYLK = 4,
    wxIPC_DIF = 5,
    wxIPC_TIFF = 6,
    wxIPC_OEMTEXT = 7, /* CF_OEMTEXT */
    wxIPC_DIB = 8, /* CF_DIB */
    wxIPC_PALETTE = 9,
    wxIPC_PENDATA = 10,
    wxIPC_RIFF = 11,
    wxIPC_WAVE = 12,
    wxIPC_UNICODETEXT = 13,
    wxIPC_ENHMETAFILE = 14,
    wxIPC_FILENAME = 15, /* CF_HDROP */
    wxIPC_LOCALE = 16,
    wxIPC_PRIVATE = 20
};
```

### See also

*wxDDEClient* (p. 477), *wxDDEServer* (p. 482), *Interprocess communications overview* (p. 2175)

## **wxDDEConnection::wxDDEConnection**

### **wxDDEConnection()**

### **wxDDEConnection(char\* buffer, int size)**

Constructs a connection object. If no user-defined connection object is to be derived from `wxDDEConnection`, then the constructor should not be called directly, since the default connection object will be provided on requesting (or accepting) a connection. However, if the user defines his or her own derived connection object, the *wxDDEServer::OnAcceptConnection* (p. 483) and/or *wxDDEClient::OnMakeConnection* (p. 478) members should be replaced by functions which construct the new connection object. If the arguments of the `wxDDEConnection` constructor are void, then a default buffer is associated with the connection. Otherwise, the programmer must provide a

buffer and size of the buffer for the connection object to use in transactions.

### **wxDDEConnection::Advise**

**bool Advise**(const wxString& *item*, char\* *data*, int *size* = -1, wxIPCFormat *format* = wxCF\_TEXT)

Called by the server application to advise the client of a change in the data associated with the given item. Causes the client connection's *wxDDEConnection::OnAdvise* (p. 480) member to be called. Returns true if successful.

### **wxDDEConnection::Execute**

**bool Execute**(char\* *data*, int *size* = -1, wxIPCFormat *format* = wxCF\_TEXT)

Called by the client application to execute a command on the server. Can also be used to transfer arbitrary data to the server (similar to *wxDDEConnection::Poke* (p. 481) in that respect). Causes the server connection's *wxDDEConnection::OnExecute* (p. 480) member to be called. Returns true if successful.

### **wxDDEConnection::Disconnect**

**bool Disconnect**()

Called by the client or server application to disconnect from the other program; it causes the *wxDDEConnection::OnDisconnect* (p. 480) 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.

### **wxDDEConnection::OnAdvise**

**virtual bool OnAdvise**(const wxString& *topic*, const wxString& *item*, char\* *data*, int *size*, wxIPCFormat *format*)

Message sent to the client application when the server notifies it of a change in the data associated with the given item.

### **wxDDEConnection::OnDisconnect**

**virtual bool OnDisconnect**()

Message sent to the client or server application when the other application notifies it to delete the connection. Default behaviour is to delete the connection object.

### **wxDDEConnection::OnExecute**

**virtual bool OnExecute**(const wxString& *topic*, char\* *data*, int *size*, wxIPCFormat *format*)



Message sent to the server application when the client notifies it to execute the given data. Note that there is no item associated with this message.

#### **wxDDEConnection::OnPoke**

**virtual bool OnPoke(const wxString& topic, const wxString& item, char\* data, int size, wxIPCFormat format)**

Message sent to the server application when the client notifies it to accept the given data.

#### **wxDDEConnection::OnRequest**

**virtual char\* OnRequest(const wxString& topic, const wxString& item, int \*size, wxIPCFormat format)**

Message sent to the server application when the client calls *wxDDEConnection::Request* (p. 481). The server should respond by returning a character string from **OnRequest**, or NULL to indicate no data.

#### **wxDDEConnection::OnStartAdvise**

**virtual bool OnStartAdvise(const wxString& topic, const wxString& item)**

Message sent to the server application by the client, when the client wishes to start an 'advise loop' for the given topic and item. The server can refuse to participate by returning false.

#### **wxDDEConnection::OnStopAdvise**

**virtual bool OnStopAdvise(const wxString& topic, const wxString& item)**

Message sent to the server application by the client, when the client wishes to stop an 'advise loop' for the given topic and item. The server can refuse to stop the advise loop by returning false, although this doesn't have much meaning in practice.

#### **wxDDEConnection::Poke**

**bool Poke(const wxString& item, char\* data, int size = -1, wxIPCFormat format = wxCF\_TEXT)**

Called by the client application to poke data into the server. Can be used to transfer arbitrary data to the server. Causes the server connection's *wxDDEConnection::OnPoke* (p. 481) member to be called. Returns true if successful.

#### **wxDDEConnection::Request**

**char\* Request(const wxString& item, int \*size, wxIPCFormat format = wxIPC\_TEXT)**

Called by the client application to request data from the server. Causes the server connection's *wxDDEConnection::OnRequest* (p. 481) member to be called. Returns a

character string (actually a pointer to the connection's buffer) if successful, NULL otherwise.

### **wxDDEConnection::StartAdvise**

**bool StartAdvise(const wxString& item)**

Called by the client application to ask if an advise loop can be started with the server. Causes the server connection's *wxDDEConnection::OnStartAdvise* (p. 481) member to be called. Returns true if the server okays it, false otherwise.

### **wxDDEConnection::StopAdvise**

**bool StopAdvise(const wxString& item)**

Called by the client application to ask if an advise loop can be stopped. Causes the server connection's *wxDDEConnection::OnStopAdvise* (p. 481) member to be called. Returns true if the server okays it, false otherwise.

## **wxDDEServer**

A wxDDEServer object represents the server part of a client-server DDE (Dynamic Data Exchange) conversation.

This DDE-based implementation is available on Windows only, but a platform-independent, socket-based version of this API is available using *wxTCPServer* (p. 1614).

### **Derived from**

wxServerBase

### **Include files**

<wx/dde.h>

### **See also**

*wxDDEClient* (p. 477), *wxDDEConnection* (p. 478), *IPC overview* (p. 2175)

### **wxDDEServer::wxDDEServer**

**wxDDEServer()**

Constructs a server object.

### **wxDDEServer::Create**

**bool Create(const wxString& service)**

Registers the server using the given service name. Under UNIX, the string must contain an integer id which is used as an Internet port number. false is returned if the call failed (for example, the port number is already in use).

### **wxDDEServer::OnAcceptConnection**

**virtual wxConnectionBase \* OnAcceptConnection(const wxString& topic)**

When a client calls **MakeConnection**, the server receives the message and this member is called. The application should derive a member to intercept this message and return a connection object of either the standard wxDDEConnection type, or of a user-derived type. If the topic is "STDIO", the application may wish to refuse the connection. Under UNIX, when a server is created the OnAcceptConnection message is always sent for standard input and output, but in the context of DDE messages it doesn't make a lot of sense.

## **wxDebugContext**

A class for performing various debugging and memory tracing operations. Full functionality (such as printing out objects currently allocated) is only present in a debugging build of wxWidgets, i.e. if the `__WXDEBUG__` symbol is defined. wxDebugContext and related functions and macros can be compiled out by setting `wxUSE_DEBUG_CONTEXT` to 0 in setup.h

### **Derived from**

No parent class.

### **Include files**

<wx/memory.h>

### **See also**

*Overview* (p. 2074)

### **wxDebugContext::Check**

**int Check()**

Checks the memory blocks for errors, starting from the currently set checkpoint.

### **Return value**

Returns the number of errors, so a value of zero represents success. Returns -1 if an error was detected that prevents further checking.

### **wxDebugContext::Dump**

**bool Dump()**

Performs a memory dump from the currently set checkpoint, writing to the current debug stream. Calls the **Dump** member function for each wxObject derived instance.

**Return value**

true if the function succeeded, false otherwise.

**wxDebugContext::GetCheckPrevious****bool GetCheckPrevious()**

Returns true if the memory allocator checks all previous memory blocks for errors. By default, this is false since it slows down execution considerably.

**See also**

*wxDebugContext::SetCheckPrevious* (p. 486)

**wxDebugContext::GetDebugMode****bool GetDebugMode()**

Returns true if debug mode is on. If debug mode is on, the wxObject new and delete operators store or use information about memory allocation. Otherwise, a straight malloc and free will be performed by these operators.

**See also**

*wxDebugContext::SetDebugMode* (p. 486)

**wxDebugContext::GetLevel****int GetLevel()**

Gets the debug level (default 1). The debug level is used by the wxTraceLevel function and the WXTRACELEVEL macro to specify how detailed the trace information is; setting a different level will only have an effect if trace statements in the application specify a value other than one.

This is obsolete, replaced by *wxLog* (p. 1018) functionality.

**See also**

*wxDebugContext::SetLevel* (p. 487)

**wxDebugContext::GetStream****ostream& GetStream()**

Returns the output stream associated with the debug context.

This is obsolete, replaced by *wxLog* (p. 1018) functionality.

**See also**

*wxDebugContext::SetStream* (p. 487)

**wxDebugContext::GetStreamBuf**

**streambuf\*** **GetStreamBuf()**

Returns a pointer to the output stream buffer associated with the debug context. There may not necessarily be a stream buffer if the stream has been set by the user.

This is obsolete, replaced by *wxLog* (p. 1018) functionality.

**wxDebugContext::HasStream**

**bool** **HasStream()**

Returns true if there is a stream currently associated with the debug context.

This is obsolete, replaced by *wxLog* (p. 1018) functionality.

**See also**

*wxDebugContext::SetStream* (p. 487), *wxDebugContext::GetStream* (p. 484)

**wxDebugContext::PrintClasses**

**bool** **PrintClasses()**

Prints a list of the classes declared in this application, giving derivation and whether instances of this class can be dynamically created.

**See also**

*wxDebugContext::PrintStatistics* (p. 485)

**wxDebugContext::PrintStatistics**

**bool** **PrintStatistics**(**bool** *detailed* = *true*)

Performs a statistics analysis from the currently set checkpoint, writing to the current debug stream. The number of object and non-object allocations is printed, together with the total size.

**Parameters**

*detailed*

If true, the function will also print how many objects of each class have been allocated, and the space taken by these class instances.

**See also**

*wxDebugContext::PrintStatistics* (p. 485)

### **wxDebugContext::SetCheckpoint**

**void SetCheckpoint(bool all = false)**

Sets the current checkpoint: Dump and PrintStatistics operations will be performed from this point on. This allows you to ignore allocations that have been performed up to this point.

#### **Parameters**

*all*

If true, the checkpoint is reset to include all memory allocations since the program started.

### **wxDebugContext::SetCheckPrevious**

**void SetCheckPrevious(bool check)**

Tells the memory allocator to check all previous memory blocks for errors. By default, this is false since it slows down execution considerably.

#### **See also**

*wxDebugContext::GetCheckPrevious* (p. 484)

### **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 `__WXDEBUG__` is defined. If the application uses this function, it should make sure that all object memory allocated is deallocated with the same value of debug mode. Otherwise, the delete operator might try to look for memory information that does not exist.

#### **See also**

*wxDebugContext::GetDebugMode* (p. 484)

### **wxDebugContext::SetFile**

**bool SetFile(const wxString& filename)**

Sets the current debug file and creates a stream. This will delete any existing stream and stream buffer. By default, the debug context stream outputs to the debugger (Windows) or standard error (other platforms).

**wxDebugContext::SetLevel****void SetLevel(int level)**

Sets the debug level (default 1). The debug level is used by the `wxTraceLevel` function and the `WXTRACELEVEL` macro to specify how detailed the trace information is; setting a different level will only have an effect if trace statements in the application specify a value other than one.

This is obsolete, replaced by `wxLog` (p. 1018) functionality.

**See also**

`wxDebugContext::GetLevel` (p. 484)

**wxDebugContext::SetStandardError****bool SetStandardError()**

Sets the debugging stream to be the debugger (Windows) or standard error (other platforms). This is the default setting. The existing stream will be flushed and deleted.

This is obsolete, replaced by `wxLog` (p. 1018) functionality.

**wxDebugContext::SetStream****void SetStream(ostream\* stream, streambuf\* streamBuf = NULL)**

Sets the stream and optionally, stream buffer associated with the debug context. This operation flushes and deletes the existing stream (and stream buffer if any).

This is obsolete, replaced by `wxLog` (p. 1018) functionality.

**Parameters***stream*

Stream to associate with the debug context. Do not set this to NULL.

*streamBuf*

Stream buffer to associate with the debug context.

**See also**

`wxDebugContext::GetStream` (p. 484), `wxDebugContext::HasStream` (p. 485)

**wxDebugStreamBuf**

This class allows you to treat debugging output in a similar (stream-based) fashion on different platforms. Under Windows, an `ostream` constructed with this buffer outputs to the debugger, or other program that intercepts debugging output. On other platforms, the

output goes to standard error (cerr).

This is soon to be obsolete, replaced by *wxLog* (p. 1018) functionality.

### Derived from

streambuf

### Include files

<wx/memory.h>

### Example

```
wxDebugStreamBuf streamBuf;  
ostream stream(&streamBuf);  
  
stream << "Hello world!" << endl;
```

### See also

*Overview* (p. 2074)

## wxDebugReport

wxDebugReport is used to generate a debug report, containing information about the program current state. It is usually used from *wxApp::OnFatalException()* (p. 51) as shown in the *sample* (p. 2033).

A wxDebugReport object contains one or more files. A few of them can be created by the class itself but more can be created from the outside and then added to the report. Also note that several virtual functions may be overridden to further customize the class behaviour.

Once a report is fully assembled, it can simply be left in the temporary directory so that the user can email it to the developers (in which case you should still use *wxDebugReportCompress* (p. 492) to compress it in a single file) or uploaded to a Web server using *wxDebugReportUpload* (p. 494) (setting up the Web server to accept uploads is your responsibility, of course). Other handlers, for example for automatically emailing the report, can be defined as well but are not currently included in wxWidgets.

### Example of use

```
wxDebugReport report;  
wxDebugReportPreviewStd preview;  
  
report.AddCurrentContext(); // could also use AddAll()  
report.AddCurrentDump();   // to do both at once  
  
if ( preview.Show(report) )  
    report.Process();
```

### Derived from



No base class

### Include files

<wx/debugrpt.h>

### Data structures

This enum is used for functions that report either the current state or the state during the last (fatal) exception:

```
enum wxDebugReport::Context
{
    Context_Current,
    Context_Exception
};
```

### **wxDebugReport::wxDebugReport**

#### **wxDebugReport()**

The constructor creates a temporary directory where the files that will be included in the report are created. Use *IsOk()* (p. 492) to check for errors.

### **wxDebugReport::~~wxDebugReport**

#### **~wxDebugReport()**

The destructor normally destroys the temporary directory created in the constructor with all the files it contains. Call *Reset()* (p. 492) to prevent this from happening.

### **wxDebugReport::AddAll**

**void AddAll(Context context = Context\_Exception)**

Adds all available information to the report. Currently this includes a text (XML) file describing the process context and, under Win32, a minidump file.

### **wxDebugReport::AddContext**

**bool AddContext(Context ctx)**

Add an XML file containing the current or exception context and the stack trace.

### **wxDebugReport::AddCurrentContext**

**bool AddCurrentContext()**

The same as *AddContext(Context\_Current)* (p. 489).

**wxDebugReport::AddCurrentDump****bool AddCurrentDump()**

The same as *AddDump(Context\_Current)* (p. 490).

**wxDebugReport::AddDump****bool AddDump(Context ctx)**

Adds the minidump file to the debug report.

Minidumps are only available under recent Win32 versions (*dbghlp32.dll* can be installed under older systems to make minidumps available).

**wxDebugReport::AddExceptionContext****bool AddExceptionContext()**

The same as *AddContext(Context\_Exception)* (p. 489).

**wxDebugReport::AddExceptionDump****bool AddExceptionDump()**

The same as *AddDump(Context\_Exception)* (p. 490).

**wxDebugReport::AddFile****void AddFile(const wxString& filename, const wxString& description)**

Add another file to the report. If *filename* is an absolute path, it is copied to a file in the debug report directory with the same name. Otherwise the file should already exist in this directory

*description* only exists to be displayed to the user in the report summary shown by *wxDebugReportPreview* (p. 493).

**See also**

*GetDirectory()* (p. 491),  
*AddText()* (p. 490)

**wxDebugReport::AddText****bool AddText(const wxString& filename, const wxString& text, const wxString& description)**

This is a convenient wrapper around *AddFile* (p. 490). It creates the file with the given *name* and writes *text* to it, then adds the file to the report. The *filename* shouldn't contain the path.

Returns `true` if file could be added successfully, `false` if an IO error occurred.

### **wxDebugReport::DoAddCustomContext**

**void DoAddCustomContext(wxXmlNode\* nodeRoot)**

This function may be overridden to add arbitrary custom context to the XML context file created by *AddContext* (p. 489). By default, it does nothing.

### **wxDebugReport::DoAddExceptionInfo**

**bool DoAddExceptionInfo(wxXmlNode\* nodeContext)**

This function may be overridden to modify the contents of the exception tag in the XML context file.

### **wxDebugReport::DoAddLoadedModules**

**bool DoAddLoadedModules(wxXmlNode\* nodeModules)**

This function may be overridden to modify the contents of the modules tag in the XML context file.

### **wxDebugReport::DoAddSystemInfo**

**bool DoAddSystemInfo(wxXmlNode\* nodeSystemInfo)**

This function may be overridden to modify the contents of the system tag in the XML context file.

### **wxDebugReport::GetDirectory**

**const wxString& GetDirectory() const**

Returns the name of the temporary directory used for the files in this report.

This method should be used to construct the full name of the files which you wish to add to the report using *AddFile* (p. 490).

### **wxDebugReport::GetFile**

**bool GetFile(size\_t n, wxString\* name, wxString\* desc) const**

Retrieves the name (relative to *GetDirectory()* (p. 491)) and the description of the file with the given index. If *n* is greater than or equal to the number of files, `false` is returned.

### **wxDebugReport::GetFilesCount**

**size\_t GetFilesCount() const**

Gets the current number files in this report.

### **wxDebugReport::GetReportName**

**wxString GetReportName() const**

Gets the name used as a base name for various files, by default *wxApp::GetAppName()* (p. 47) is used.

### **wxDebugReport::IsOk**

**bool IsOk() const**

Returns `true` if the object was successfully initialized. If this method returns `false` the report can't be used.

### **wxDebugReport::Process**

**bool Process()**

Processes this report: the base class simply notifies the user that the report has been generated. This is usually not enough -- instead you should override this method to do something more useful to you.

### **wxDebugReport::RemoveFile**

**void RemoveFile(const wxString& name)**

Removes the file from report: this is used by *wxDebugReportPreview* (p. 493) to allow the user to remove files potentially containing private information from the report.

### **wxDebugReport::Reset**

**void Reset()**

Resets the directory name we use. The object can't be used any more after this as it becomes uninitialized and invalid.

## **wxDebugReportCompress**

*wxDebugReportCompress* is a *wxDebugReport* (p. 488) which compresses all the files in this debug report into a single .ZIP file in its *Process()* function.

### **Derived from**

*wxDebugReport* (p. 488)

### **Include files**

<wx/debugrpt.h>

**wxDebugReportCompress::wxDebugReportCompress****wxDebugReportCompress()**

Default constructor does nothing special.

**wxDebugReportCompress::GetCompressedFileName****const wxString& GetCompressedFileName() const**

Returns the full path of the compressed file (empty if creation failed).

**wxDebugReportPreview**

This class presents the debug report to the user and allows him to veto report entirely or remove some parts of it. Although not mandatory, using this class is strongly recommended as data included in the debug report might contain sensitive private information and the user should be notified about it as well as having a possibility to examine the data which had been gathered to check whether this is effectively the case and discard the debug report if it is.

wxDebugReportPreview is an abstract base class, currently the only concrete class deriving from it is *wxDebugReportPreviewStd* (p. 494).

**Derived from**

No base class

**Include files**

<wx/debugrpt.h>

**wxDebugReportPreview::wxDebugReportPreview****wxDebugReportPreview()**

Trivial default constructor.

**wxDebugReportPreview::~~wxDebugReportPreview****~wxDebugReportPreview()**

dtor is trivial as well but should be virtual for a base class

**wxDebugReportPreview::Show**

**bool Show(wxDebugReport& dbg rpt) const**

Present the report to the user and allow him to modify it by removing some or all of the files and, potentially, adding some notes. Return `true` if the report should be processed or `false` if the user chose to cancel report generation or removed all files from it.

**wxDebugReportPreviewStd**

`wxDebugReportPreviewStd` is a standard debug report preview window. It displays a `GUIdialog` allowing the user to examine the contents of a debug report, remove files from and add notes to it.

**Derived from**

*wxDebugReportPreview* (p. 493)

**Include files**

<wx/debugrpt.h>

**wxDebugReportPreviewStd::wxDebugReportPreviewStd****wxDebugReportPreviewStd()**

Trivial default constructor.

**wxDebugReportPreviewStd::Show****bool Show(wxDebugReport& dbg rpt) const**

Show the dialog, see *wxDebugReportPreview::Show()* (p. 493) for more information.

**wxDebugReportUpload**

This class is used to upload a compressed file using HTTP POST request. As this class derives from `wxDebugReportCompress`, before upload the report is compressed in a single .ZIP file.

**Derived from**

*wxDebugReportCompress* (p. 492)

**Include files**

<wx/debugrpt.h>

### **wxDebugReportUpload::wxDebugReportUpload**

**wxDebugReportUpload(const wxString& url, const wxString& input, const wxString& action, const wxString& curl = \_T("curl"))**

This class will upload the compressed file created by its base class to an HTML multipart/form-data form at the specified address. The *url* is the upload page address, *input* is the name of the "type=file" control on the form used for the file name and *action* is the value of the form action field. The report is uploaded using *curl* program which should be available, the *curl* parameter may be used to specify the full path to it.

### **wxDebugReportUpload::OnServerReply**

**bool OnServerReply(const wxArrayString& WXUNUSED(reply))**

This function may be overridden in a derived class to show the output from curl: this may be an HTML page or anything else that the server returned. Value returned by this function becomes the return value of *wxDebugReport::Process()* (p. 492).

## **wxDelegateRendererNative**

*wxDelegateRendererNative* allows reuse of renderers code by forwarding all the *wxRendererNative* (p. 1276) methods to the given object and thus allowing you to only modify some of its methods -- without having to reimplement all of them.

Note that the "normal", inheritance-based approach, doesn't work with the renderers as it is impossible to derive from a class unknown at compile-time and the renderer is only chosen at run-time. So suppose that you want to only add something to the drawing of the tree control buttons but leave all the other methods unchanged -- the only way to do it, considering that the renderer class which you want to customize might not even be written yet when you write your code (it could be written later and loaded from a DLL during run-time), is by using this class.

Except for the constructor, it has exactly the same methods as *wxRendererNative* (p. 1276) and their implementation is trivial: they are simply forwarded to the real renderer. Note that the "real" renderer may, in turn, be a *wxDelegateRendererNative* as well and that there may be arbitrarily many levels like this -- but at the end of the chain there must be a real renderer which does the drawing.

### **Derived from**

*wxRendererNative* (p. 1276)

### **Include files**

<wx/renderer.h>

### **wxDelegateRendererNative::wxDelegateRendererNative**

**wxDelegateRendererNative()****wxDelegateRendererNative(wxRendererNative& *rendererNative*)**

The default constructor does the same thing as the other one except that it uses the *generic renderer* (p. 1279) instead of the user-specified *rendererNative*.

In any case, this sets up the delegate renderer object to follow all calls to the specified real renderer.

Note that this object does *not* take ownership of (i.e. won't delete) *rendererNative*.

**wxDelegateRendererNative::DrawXXX****DrawXXX(...)**

This class also provides all the virtual methods of *wxRendererNative* (p. 1276), please refer to that class documentation for the details.

## wxDialog

A dialog box is a window with a title bar and sometimes a system menu, which can be moved around the screen. It can contain controls and other windows and is often used to allow the user to make some choice or to answer a question.

**Dialog Buttons**

The dialog usually contains either a single button allowing to close the dialog or two buttons, one accepting the changes and the other one discarding them (such button, if present, is automatically activated if the user presses the "ESC" key). By default, buttons with the standard `wxID_OK` and `wxID_CANCEL` identifiers behave as expected. Starting with *wxWidgets* 2.7 it is also possible to use a button with a different identifier instead, see *SetAffirmativeId* (p. 503) and *SetEscapeId* (p. 503).

Also notice that the *CreateButtonSizer()* (p. 499) should be used to create the buttons appropriate for the current platform and positioned correctly (including their order which is platform-dependent).

**Derived from**

*wxTopLevelWindow* (p. 1713)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

**Include files**

<wx/dialog.h>

**Modal and modeless dialogs**

There are two kinds of dialog -- *modal* and *modeless*. A modal dialog blocks program flow



and user input on other windows until it is dismissed, whereas a modeless dialog behaves more like a frame in that program flow continues, and input in other windows is still possible. To show a modal dialog you should use the *ShowModal* (p. 505) method while to show a dialog modelessly you simply use *Show* (p. 504), just as with frames.

Note that the modal dialog is one of the very few examples of *wxWindow*-derived objects which may be created on the stack and not on the heap. In other words, although this code snippet:

```
void AskUser()
{
    MyAskDialog *dlg = new MyAskDialog(...);
    if ( dlg->ShowModal() == wxID_OK )
        ...
    //else: dialog was cancelled or some another button pressed

    dlg->Destroy();
}
```

works, you can also achieve the same result by using a simpler code fragment below:

```
void AskUser()
{
    MyAskDialog dlg(...);
    if ( dlg.ShowModal() == wxID_OK )
        ...

    // no need to call Destroy() here
}
```

An application can define a *wxCloseEvent* (p. 200) handler for the dialog to respond to system close events.

### Window styles

<b>wxCAPTION</b>	Puts a caption on the dialog box.
<b>wxDEFAULT_DIALOG_STYLE</b>	Equivalent to a combination of <b>wxCAPTION</b> , <b>wxCLOSE_BOX</b> and <b>wxSYSTEM_MENU</b> (the last one is not used under Unix)
<b>wxRESIZE_BORDER</b>	Display a resizable frame around the window.
<b>wxSYSTEM_MENU</b>	Display a system menu.
<b>wxCLOSE_BOX</b>	Displays a close box on the frame.
<b>wxMAXIMIZE_BOX</b>	Displays a maximize box on the dialog.
<b>wxMINIMIZE_BOX</b>	Displays a minimize box on the dialog.
<b>wxTHICK_FRAME</b>	Display a thick frame around the window.
<b>wxSTAY_ON_TOP</b>	The dialog stays on top of all other windows.

- wxNO\_3D** Under Windows, specifies that the child controls should not have 3D borders unless specified in the control.
- wxDIALOG\_NO\_PARENT** By default, a dialog created with a `NULL` parent window will be given the *application's top level window* (p. 48) as parent. Use this style to prevent this from happening and create an orphan dialog. This is not recommended for modal dialogs.
- wxDIALOG\_EX\_CONTEXTHELP** Under Windows, puts a query button on the caption. When pressed, Windows will go into a context-sensitive help mode and `wxWidgets` will send a `wxEVT_HELP` event if the user clicked on an application window. *Note* that this is an extended style and must be set by calling `SetExtraStyle` (p. 1839) before `Create` is called (two-step construction).
- wxDIALOG\_EX\_METAL** On Mac OS X, frames with this style will be shown with a metallic look. This is an *extra* style.

Under Unix or Linux, MWM (the Motif Window Manager) or other window managers recognizing the MWM hints should be running for any of these styles to have an effect.

See also *Generic window styles* (p. 2089).

#### See also

*wxDialog overview* (p. 2092), *wxFrame* (p. 682), *Validator overview* (p. 2092)

## wxDialog::wxDialog

### wxDialog()

Default constructor.

**wxDialog(wxWindow\* parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT\_DIALOG\_STYLE, const wxString& name = "dialogBox")**

Constructor.

### Parameters

*parent*

Can be `NULL`, a frame or another dialog box.

*id*

An identifier for the dialog. A value of -1 is taken to mean a default.

*title*

The title of the dialog.

*pos*

The dialog position. A value of (-1, -1) indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform.

*size*

The dialog size. A value of (-1, -1) indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform.

*style*

The window style. See *wxDialog* (p. 496).

*name*

Used to associate a name with the window, allowing the application user to set Motif resource values for individual dialog boxes.

### See also

*wxDialog::Create* (p. 499)

### **wxDialog::~wxDialog**

**~wxDialog()**

Destructor. Deletes any child windows before deleting the physical window.

### **wxDialog::Centre**

**void Centre(int direction = wxBOTH)**

Centres the dialog box on the display.

### Parameters

*direction*

May be wxHORIZONTAL, wxVERTICAL or wxBOTH.

### **wxDialog::Create**

**bool Create(wxWindow\* parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT\_DIALOG\_STYLE, const wxString& name = "dialogBox")**

Used for two-step dialog box construction. See *wxDialog::wxDialog* (p. 498) for details.

### **wxDialog::CreateButtonSizer**

**wxSizer\* CreateButtonSizer(long flags)**

Creates a sizer with standard buttons. *flags* is a bit list of the following flags: `wxOK`, `wxCANCEL`, `wxYES`, `wxNO`, `wxHELP`, `wxNO_DEFAULT`.

The sizer lays out the buttons in a manner appropriate to the platform.

This function uses *CreateStdDialogButtonSizer* (p. 500) internally for most platforms but doesn't create the sizer at all for the platforms with hardware buttons (such as smartphones) for which it sets up the hardware buttons appropriately and returns `NULL`, so don't forget to test that the return value is valid before using it.

### **`wxDialog::CreateSeparatedButtonSizer`**

**`wxSizer* CreateSeparatedButtonSizer(long flags)`**

Creates a sizer with standard buttons using *CreateButtonSizer* (p. 499) separated from the rest of the dialog contents by a horizontal *wxStaticLine* (p. 1532).

Please notice that just like *CreateButtonSizer()* this function may return `NULL` if no buttons were created.

### **`wxDialog::CreateStdDialogButtonSizer`**

**`wxStdDialogButtonSizer* CreateStdDialogButtonSizer(long flags)`**

Creates a *wxStdDialogButtonSizer* (p. 1541) with standard buttons. *flags* is a bit list of the following flags: `wxOK`, `wxCANCEL`, `wxYES`, `wxNO`, `wxHELP`, `wxNO_DEFAULT`.

The sizer lays out the buttons in a manner appropriate to the platform.

### **`wxDialog::DoOK`**

**`virtual bool DoOK()`**

This function is called when the titlebar OK button is pressed (PocketPC only). A command event for the identifier returned by *GetAffirmativeId* is sent by default. You can override this function. If the function returns false, *wxWidgets* will call *Close()* for the dialog.

### **`wxDialog::EndModal`**

**`void EndModal(int retCode)`**

Ends a modal dialog, passing a value to be returned from the *wxDialog::ShowModal* (p. 505) invocation.

#### **Parameters**

*retCode*

The value that should be returned by **ShowModal**.

#### **See also**

*wxDialog::ShowModal* (p. 505), *wxDialog::GetReturnCode* (p. 501),  
*wxDialog::SetReturnCode* (p. 504)

### **wxDialog::GetAffirmativeId**

**int GetAffirmativeId() const**

Gets the identifier of the button which works like standard OK button in this dialog.

#### **See also**

*wxDialog::SetAffirmativeId* (p. 503)

### **wxDialog::GetEscapId**

**int GetEscapId() const**

Gets the identifier of the button to map presses of `ESC` button to.

#### **See also**

*wxDialog::SetEscapId* (p. 503)

### **wxDialog::GetReturnCode**

**int GetReturnCode()**

Gets the return code for this window.

#### **Remarks**

A return code is normally associated with a modal dialog, where *wxDialog::ShowModal* (p. 505) returns a code to the application.

#### **See also**

*wxDialog::SetReturnCode* (p. 504), *wxDialog::ShowModal* (p. 505), *wxDialog::EndModal* (p. 500)

### **wxDialog::GetToolBar**

**wxToolBar\* GetToolBar() const**

On PocketPC, a dialog is automatically provided with an empty toolbar. `GetToolBar` allows you to access the toolbar and add tools to it. Removing tools and adding arbitrary controls are not currently supported.

This function is not available on any other platform.

### **wxDialog::Iconize**

**void Iconize(const bool iconize)**

Iconizes or restores the dialog. Windows only.

### Parameters

*iconize*

If true, iconizes the dialog box; if false, shows and restores it.

### Remarks

Note that in Windows, iconization has no effect since dialog boxes cannot be iconized. However, applications may need to explicitly restore dialog boxes under Motif which have user-iconizable frames, and under Windows calling `Iconize(false)` will bring the window to the front, as does `Show(true)`.

### **wxDialog::IsIconized**

**bool IsIconized() const**

Returns true if the dialog box is iconized. Windows only.

### Remarks

Always returns false under Windows since dialogs cannot be iconized.

### **wxDialog::IsModal**

**bool IsModal() const**

Returns true if the dialog box is modal, false otherwise.

### **wxDialog::OnSysColourChanged**

**void OnSysColourChanged(wxSysColourChangedEvent& event)**

The default handler for `wxEVT_SYS_COLOUR_CHANGED`.

### Parameters

*event*

The colour change event.

### Remarks

Changes the dialog's colour to conform to the current settings (Windows only). Add an event table entry for your dialog class if you wish the behaviour to be different (such as keeping a user-defined background colour). If you do override this function, call `wxEvent::Skip` to propagate the notification to child windows and controls.

### See also

*wxSysColourChangedEvent* (p. 1590)

**wxDialog::SetAffirmativeId****void SetAffirmativeId(int id)**

Sets the identifier to be used as OK button. When the button with this identifier is pressed, the dialog calls *Validate* (p. 1853) and *wxWindow::TransferDataFromWindow* (p. 1851) and, if they both return `true`, closes the dialog with `wxID_OK` return code.

Also, when the user presses a hardware OK button on the devices having one or the special OK button in the PocketPC title bar, an event with this id is generated.

By default, the affirmative id is `wxID_OK`.

**See also**

*wxDialog::GetAffirmativeId* (p. 501), *wxDialog::SetEscapeId* (p. 503)

**wxDialog::SetEscapeId****void SetEscapeId(int id)**

Sets the identifier of the button which should work like the standard `CANCEL` button in this dialog. When the button with this id is clicked, the dialog is closed. Also, when the user presses `ESC` key in the dialog or closes the dialog using the close button in the title bar, this is mapped to the click of the button with the specified id.

By default, the escape id is the special value `wxID_ANY` meaning that `wxID_CANCEL` button is used if it's present in the dialog and otherwise the button with *GetAffirmativeId()* (p. 501) is used. Another special value for *id* is `wxID_NONE` meaning that `ESC` presses should be ignored. If any other value is given, it is interpreted as the id of the button to map the escape key to.

**wxDialog::SetIcon****void SetIcon(const wxIcon& icon)**

Sets the icon for this dialog.

**Parameters***icon*

The icon to associate with this dialog.

See also *wxIcon* (p. 894).

**wxDialog::SetIcons****void SetIcons(const wxIconBundle& icons)**

Sets the icons for this dialog.

**Parameters**

*icons*

The icons to associate with this dialog.

See also *wxIconBundle* (p. 901).

**wxDialog::SetModal**

**void SetModal(const bool *flag*)**

**NB:** This function is deprecated and doesn't work for all ports, just use *ShowModal* (p. 505) to show a modal dialog instead.

Allows the programmer to specify whether the dialog box is modal (*wxDialog::Show* blocks control until the dialog is hidden) or modeless (control returns immediately).

**Parameters**

*flag*

If true, the dialog will be modal, otherwise it will be modeless.

**wxDialog::SetReturnCode**

**void SetReturnCode(int *retCode*)**

Sets the return code for this window.

**Parameters**

*retCode*

The integer return code, usually a control identifier.

**Remarks**

A return code is normally associated with a modal dialog, where *wxDialog::ShowModal* (p. 505) returns a code to the application. The function *wxDialog::EndModal* (p. 500) calls **SetReturnCode**.

**See also**

*wxDialog::GetReturnCode* (p. 501), *wxDialog::ShowModal* (p. 505), *wxDialog::EndModal* (p. 500)

**wxDialog::Show**

**bool Show(const bool *show*)**

Hides or shows the dialog.

**Parameters**



*show*

If true, the dialog box is shown and brought to the front; otherwise the box is hidden.  
If false and the dialog is modal, control is returned to the calling program.

#### Remarks

The preferred way of dismissing a modal dialog is to use *wxDialog::EndModal* (p. 500).

### **wxDialog::ShowModal**

#### **int ShowModal()**

Shows a modal dialog. Program flow does not return until the dialog has been dismissed with *wxDialog::EndModal* (p. 500).

#### Return value

The return value is the value set with *wxDialog::SetReturnCode* (p. 504).

#### See also

*wxDialog::EndModal* (p. 500), *wxDialog::GetReturnCode* (p. 501),  
*wxDialog::SetReturnCode* (p. 504)

## **wxDialUpEvent**

This is the event class for the dialup events sent by *wxDialUpManager* (p. 506).

#### Derived from

*wxEvent* (p. 572)  
*wxObject* (p. 1148)

#### Include files

<wx/dialup.h>

### **wxDialUpEvent::wxDialUpEvent**

**wxDialUpEvent**(bool *isConnected*, bool *isOwnEvent*)

Constructor is only used by *wxDialUpManager* (p. 506).

### **wxDialUpEvent::IsConnectedEvent**

#### **bool IsConnectedEvent() const**

Is this a `CONNECTED` or `DISCONNECTED` event? In other words, does it notify about transition from offline to online state or vice versa?

**wxDialUpEvent::IsOwnEvent****bool IsOwnEvent() const**

Does this event come from wxDialUpManager::Dial() or from some external process (i.e. does it result from our own attempt to establish the connection)?

**wxDialUpManager**

This class encapsulates functions dealing with verifying the connection status of the workstation (connected to the Internet via a direct connection, connected through a modem or not connected at all) and to establish this connection if possible/required (i.e. in the case of the modem).

The program may also wish to be notified about the change in the connection status (for example, to perform some action when the user connects to the network the next time or, on the contrary, to stop receiving data from the net when the user hangs up the modem). For this, you need to use one of the event macros described below.

This class is different from other wxWidgets classes in that there is at most one instance of this class in the program accessed via *wxDialUpManager::Create()* (p. 506) and you can't create the objects of this class directly.

**Derived from**

No base class

**Include files**

<wx/dialup.h>

**Event table macros**

To be notified about the change in the network connection status, use these event handler macros to direct input to member functions that take a *wxDialUpEvent* (p. 505) argument.

**EVT\_DIALUP\_CONNECTED(func)**      A connection with the network was established.

**EVT\_DIALUP\_DISCONNECTED(func)**      The connection with the network was lost.

**See also**

*dialup sample* (p. 2033)

*wxDialUpEvent* (p. 505)

**wxDialUpManager::Create****wxDialUpManager\* Create()**

This function should create and return the object of the platform-specific class derived from wxDialUpManager. You should delete the pointer when you are done with it.

**wxDialUpManager::IsOk****bool IsOk() const**

Returns `true` if the dialup manager was initialized correctly. If this function returns `false`, no other functions will work neither, so it is a good idea to call this function and check its result before calling any other `wxDialUpManager` methods

**wxDialUpManager::~~wxDialUpManager****~wxDialUpManager()**

Destructor.

**wxDialUpManager::GetISPNames****size\_t GetISPNames(wxArrayString& names) const**

This function is only implemented under Windows.

Fills the array with the names of all possible values for the first parameter to *Dial()* (p. 507) on this machine and returns their number (may be 0).

**wxDialUpManager::Dial**

**bool Dial(const wxString& nameOfISP = wxEmptyString, const wxString& username = wxEmptyString, const wxString& password = wxEmptyString, bool async = true)**

Dial the given ISP, use *username* and *password* to authenticate.

The parameters are only used under Windows currently, for Unix you should use *SetConnectCommand* (p. 509) to customize this functions behaviour.

If no *nameOfISP* is given, the function will select the default one (proposing the user to choose among all connections defined on this machine) and if no *username* and/or *password* are given, the function will try to do without them, but will ask the user if really needed.

If *async* parameter is `false`, the function waits until the end of dialing and returns `true` upon successful completion.

If *async* is `true`, the function only initiates the connection and returns immediately - the result is reported via events (an event is sent anyhow, but if dialing failed it will be a DISCONNECTED one).

**wxDialUpManager::IsDialing****bool IsDialing() const**

Returns `true` if (async) dialing is in progress.

**See also**

*Dial* (p. 507)

### **wxDialUpManager::CancelDialing**

#### **bool CancelDialing()**

Cancel dialing the number initiated with *Dial* (p. 507) with `async` parameter equal to `true`.

Note that this won't result in `DISCONNECTED` event being sent.

#### **See also**

*IsDialing* (p. 507)

### **wxDialUpManager::HangUp**

#### **bool HangUp()**

Hang up the currently active dial up connection.

### **wxDialUpManager::IsAlwaysOnline**

#### **bool IsAlwaysOnline() const**

Returns `true` if the computer has a permanent network connection (i.e. is on a LAN) and so there is no need to use *Dial*() function to go online.

**NB:** this functions tries to guess the result and it is not always guaranteed to be correct, so it is better to ask user for confirmation or give him a possibility to override it.

### **wxDialUpManager::IsOnline**

#### **bool IsOnline() const**

Returns `true` if the computer is connected to the network: under Windows, this just means that a RAS connection exists, under Unix we check that the "well-known host" (as specified by *SetWellKnownHost* (p. 509)) is reachable.

### **wxDialUpManager::SetOnlineStatus**

#### **void SetOnlineStatus(bool isOnline = true)**

Sometimes the built-in logic for determining the online status may fail, so, in general, the user should be allowed to override it. This function allows to forcefully set the online status - whatever our internal algorithm may think about it.

#### **See also**

*IsOnline* (p. 508)

### **wxDialUpManager::EnableAutoCheckOnlineStatus**

**bool EnableAutoCheckOnlineStatus(size\_t nSeconds = 60)**

Enable automatic checks for the connection status and sending of `wxEVT_DIALUP_CONNECTED`/`wxEVT_DIALUP_DISCONNECTED` events. The interval parameter is only for Unix where we do the check manually and specifies how often should we repeat the check (each minute by default). Under Windows, the notification about the change of connection status is sent by the system and so we don't do any polling and this parameter is ignored.

Returns `false` if couldn't set up automatic check for online status.

**wxDialUpManager::DisableAutoCheckOnlineStatus**

**void DisableAutoCheckOnlineStatus()**

Disable automatic check for connection status change - notice that the `wxEVT_DIALUP_XXX` events won't be sent any more neither.

**wxDialUpManager::SetWellKnownHost**

**void SetWellKnownHost(const wxString& hostname, int portno = 80)**

This method is for Unix only.

Under Unix, the value of well-known host is used to check whether we're connected to the internet. It is unused under Windows, but this function is always safe to call. The default value is `www.yahoo.com:80`.

**wxDialUpManager::SetConnectCommand**

**void SetConnectCommand(const wxString& commandDial = wxT("/usr/bin/pon"),  
const wxString& commandHangup = wxT("/usr/bin/poff"))**

This method is for Unix only.

Sets the commands to start up the network and to hang up again.

**See also**

*Dial* (p. 507)

**wxDir**

`wxDir` is a portable equivalent of Unix `open/read/closedir` functions which allow enumerating of the files in a directory. `wxDir` allows to enumerate files as well as directories.

`wxDir` also provides a flexible way to enumerate files recursively using *Traverse* (p. 513) or a simpler *GetAllFiles* (p. 511) function.

Example of use:

```
wxDir dir(wxGetCwd());

if ( !dir.IsOpened() )
{
    // deal with the error here - wxDir would already log an error
message // explaining the exact reason of the failure
    return;
}

puts("Enumerating object files in current directory:");

wxString filename;

bool cont = dir.GetFirst(&filename, filespec, flags);
while ( cont )
{
    printf("%s\n", filename.c_str());

    cont = dir.GetNext(&filename);
}
```

### Derived from

No base class

### Constants

These flags define what kind of filename is included in the list of files enumerated by GetFirst/GetNext.

```
enum
{
    wxDIR_FILES      = 0x0001,      // include files
    wxDIR_DIRS       = 0x0002,      // include directories
    wxDIR_HIDDEN     = 0x0004,      // include hidden files
    wxDIR_DOTDOT     = 0x0008,      // include '.' and '..'

    // by default, enumerate everything except '.' and '..'
    wxDIR_DEFAULT    = wxDIR_FILES | wxDIR_DIRS | wxDIR_HIDDEN
}
```

### Include files

<wx/dir.h>

### wxDir::wxDir

**wxDir()**

Default constructor, use *Open()* (p. 513) afterwards.

**wxDir(const wxString& dir)**

Opens the directory for enumeration, use *IsOpened()* (p. 512) to test for errors.

**wxDir::~~wxDir****~wxDir()**

Destructor cleans up the associated resources. It is not virtual and so this class is not meant to be used polymorphically.

**wxDir::Exists****static bool Exists(const wxString& dir)**

Test for existence of a directory with the given name

**wxDir::GetAllFiles****static size\_t GetAllFiles(const wxString& dirname, wxArrayString \*files, const wxString& filespec = wxEmptyString, int flags = wxDIR\_DEFAULT)**

The function appends the names of all the files under directory *dirname* to the array *files* (note that its old content is preserved). Only files matching the *filespec* are taken, with empty spec matching all the files.

The *flags* parameter should always include `wxDIR_FILES` or the array would be unchanged and should include `wxDIR_DIRS` flag to recurse into subdirectories (both flags are included in the value by default).

See also: *Traverse* (p. 513)

**wxDir::FindFirst****static wxString FindFirst(const wxString& dirname, const wxString& filespec, int flags = wxDIR\_DEFAULT)**

The function returns the path of the first file matching the given *filespec* or an empty string if there are no files matching it.

The *flags* parameter may or may not include `wxDIR_FILES`, the function always behaves as if it were specified. By default, *flags* includes `wxDIR_DIRS` and so the function recurses into the subdirectories but if this flag is not specified, the function restricts the search only to the directory *dirname* itself.

See also: *Traverse* (p. 513)

**wxDir::GetFirst****bool GetFirst(wxString\* filename, const wxString& filespec = wxEmptyString, int flags = wxDIR\_DEFAULT) const**

Start enumerating all files matching *filespec* (or all files if it is empty) and *flags*, return `true` on success.

### **wxDir::GetName**

**wxString GetName() const**

Returns the name of the directory itself. The returned string does not have the trailing path separator (slash or backslash).

### **wxDir::GetNext**

**bool GetNext(wxString\* filename) const**

Continue enumerating files which satisfy the criteria specified by the last call to *GetFirst* (p. 511).

### **wxDir::GetTotalSize**

**static wxULongLong GetTotalSize(const wxString& dir, wxArrayString\* filesSkipped = NULL)**

Returns the size (in bytes) of all files recursively found in *dir* or `wxInvalidSize` in case of error.

In case it happens that while traversing folders a file's size can not be read, that file is added to the *filesSkipped* array, if not `NULL`, and then skipped. This usually happens with some special folders which are locked by the operating system or by another process. Remember that when *filesSkipped*→*GetCount*() is not zero, then the returned value is not 100% accurate and, if the skipped files were big, it could be far from real size of the directory.

See also: *wxFileName::GetHumanReadableSize* (p. 619), *wxGetDiskSpace* (p. 1921)

### **wxDir::HasFiles**

**bool HasFiles(const wxString& filespec = wxEmptyString)**

Returns `true` if the directory contains any files matching the given *filespec*. If *filespec* is empty, look for any files at all. In any case, even hidden files are taken into account.

### **wxDir::HasSubDirs**

**bool HasSubDirs(const wxString& dirs spec = wxEmptyString)**

Returns `true` if the directory contains any subdirectories (if a non empty *filespec* is given, only check for directories matching it). The hidden subdirectories are taken into account as well.

### **wxDir::IsOpened**



**bool IsOpened() const**

Returns true if the directory was successfully opened by a previous call to *Open* (p. 513).

**wxDir::Open****bool Open(const wxString& dir)**

Open the directory for enumerating, returns `true` on success or `false` if an error occurred.

**wxDir::Traverse**

**size\_t Traverse(wxDirTraverser& sink, const wxString& filespec = wxEmptyString, int flags = wxDIR\_DEFAULT)**

Enumerate all files and directories under the given directory recursively calling the element of the provided *wxDirTraverser* (p. 518) object for each of them.

More precisely, the function will really recurse into subdirectories if *flags* contains `wxDIR_DIRS` flag. It will ignore the files (but still possibly recurse into subdirectories) if `wxDIR_FILES` flag is given.

For each found directory, *sink.OnDir()* (p. 519) is called and *sink.OnFile()* (p. 519) is called for every file. Depending on the return value, the enumeration may continue or stop.

The function returns the total number of files found or `(size_t)-1` on error.

See also: *GetAllFiles* (p. 511)

**wxDirDialog**

This class represents the directory chooser dialog.

**Derived from**

*wxDialog* (p. 496)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/dirdlg.h>

**Window styles****wxDD\_DEFAULT\_STYLE**

Equivalent to a combination of `wxDEFAULT_DIALOG_STYLE` and `wxRESIZE_BORDER` (the last one is not used under `wxWinCE`).

**wxDD\_DIR\_MUST\_EXIST**

The dialog will allow the user to choose only an existing folder. When this style is not given, a "Create new directory" button is added to the dialog (on Windows) or some other way is provided to the user to type the name of a new folder.

**wxDD\_CHANGE\_DIR**

Change the current working directory to the directory chosen by the user.

**NB:** on Windows the new directory button is only available with recent versions of the common dialogs.

See also *Generic window styles* (p. 2089).

**See also**

*wxDirDialog* overview (p. 2131), *wxFileDialog* (p. 600)

**wxDirDialog::wxDirDialog**

**wxDirDialog(wxWindow\* parent, const wxString& message = "Choose a directory", const wxString& defaultPath = "", long style = wxDD\_DEFAULT\_STYLE, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, const wxString& name = "wxDirCtrl")**

Constructor. Use *wxDirDialog::ShowModal* (p. 515) to show the dialog.

**Parameters**

*parent*

Parent window.

*message*

Message to show on the dialog.

*defaultPath*

The default path, or the empty string.

*style*

The dialog style. See *wxDirDialog* (p. 513)

*pos*

Dialog position. Ignored under Windows.

*size*

Dialog size. Ignored under Windows.

*name*

The dialog name, not used.

### **wxDirDialog::~wxDirDialog**

**~wxDirDialog()**

Destructor.

### **wxDirDialog::GetPath**

**wxString GetPath() const**

Returns the default or user-selected path.

### **wxDirDialog::GetMessage**

**wxString GetMessage() const**

Returns the message that will be displayed on the dialog.

### **wxDirDialog::SetMessage**

**void SetMessage(const wxString& message)**

Sets the message that will be displayed on the dialog.

### **wxDirDialog::SetPath**

**void SetPath(const wxString& path)**

Sets the default path.

### **wxDirDialog::ShowModal**

**int ShowModal()**

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

## **wxDirPickerCtrl**

This control allows the user to select a directory. The generic implementation is a button which brings up a *wxDirDialog* (p. 513) when clicked. Native implementation may differ but this is usually a (small) widget which give access to the dir-chooser dialog. It is only available if `wxUSE_DIRPICKERCTRL` is set to 1 (the default).

**Derived from**

*wxPickerBase* (p. 1183)

*wxControl* (p. 285)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### Include files

<wx/filepicker.h>

### Window styles

**wxDIRP\_DEFAULT\_STYLE** The default style: includes **wxDIRP\_DIR\_MUST\_EXIST** and, under **wxMSW** only, **wxDIRP\_USE\_TEXTCTRL**.

**wxDIRP\_USE\_TEXTCTRL** Creates a text control to the left of the picker button which is completely managed by the *wxDirPickerCtrl* (p. 515) and which can be used by the user to specify a path (see *SetPath* (p. 518)). The text control is automatically synchronized with button's value. Use functions defined in *wxPickerBase* (p. 1183) to modify the text control.

**wxDIRP\_DIR\_MUST\_EXIST** Creates a picker which allows to select only existing directories. **wxGTK** control always adds this flag internally as it does not support its absence.

**wxDIRP\_CHANGE\_DIR** Change current working directory on each user directory selection change.

### Event handling

To process a directory picker event, use these event handler macros to direct input to member functions that take a *wxFileDirPickerEvent* (p. 632) argument.

**EVT\_DIRPICKER\_CHANGED(id, func)** The user changed the directory selected in the control either using the button or using text control (see **wxDIRP\_USE\_TEXTCTRL**; note that in this case the event is fired only if the user's input is valid, e.g. an existing directory path).

### See also

*wxDirDialog* (p. 513),

*wxFileDirPickerEvent* (p. 632)

### **wxDirPickerCtrl::wxDirPickerCtrl**

**wxDirPickerCtrl**(*wxWindow \*parent*, *wxWindowID id*, *const wxString& path* = *wxEmptyString*, *const wxString& message* = "Select a folder", *const wxPoint& pos* = *wxDefaultPosition*, *const wxSize& size* = *wxDefaultSize*, *long style* =

```
wxDIRP_DEFAULT_STYLE, const wxValidator& validator = wxDefaultValidator, const wxString& name = "dirpickerctrl")
```

Initializes the object and calls *Create* (p. 517) with all the parameters.

### **wxDirPickerCtrl::Create**

```
bool Create(wxWindow *parent, wxWindowID id, const wxString& path =  
wxEmptyString, const wxString& message = "Select a folder", const wxPoint& pos =  
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxDIRP_DEFAULT_STYLE, const wxValidator& validator = wxDefaultValidator, const  
wxString& name = "dirpickerctrl")
```

#### **Parameters**

*parent*

Parent window, must not be non-NULL.

*id*

The identifier for the control.

*path*

The initial directory shown in the control. Must be a valid path to a directory or the empty string.

*message*

The message shown to the user in the *wxDirDialog* (p. 513) shown by the control.

*pos*

Initial position.

*size*

Initial size.

*style*

The window style, see *wxDIRP\_\** flags.

*validator*

Validator which can be used for additional date checks.

*name*

Control name.

#### **Return value**

*true* if the control was successfully created or *false* if creation failed.

**wxDirPickerCtrl::GetPath****wxString GetPath() const**

Returns the absolute path of the currently selected directory.

**wxDirPickerCtrl::SetPath****void SetPath(const wxString &dirname)**

Sets the absolute path of the currently selected directory. This must be a valid directory if `wxDIRP_DIR_MUST_EXIST` style was given.

**wxDirTraverser**

`wxDirTraverser` is an abstract interface which must be implemented by objects passed to *Traverse* (p. 513) function.

Example of use (this works almost like *GetAllFiles* (p. 511)):

```
class wxDirTraverserSimple : public wxDirTraverser
{
public:
    wxDirTraverserSimple(wxArrayString& files) : m_files(files)
    { }

    virtual wxDirTraverseResult OnFile(const wxString& filename)
    {
        m_files.Add(filename);
        return wxDIR_CONTINUE;
    }

    virtual wxDirTraverseResult OnDir(const wxString&
WXUNUSED(dirname))
    {
        return wxDIR_CONTINUE;
    }

private:
    wxArrayString& m_files;
};

// get the names of all files in the array
wxArrayString files;
wxDirTraverserSimple traverser(files);

wxDir dir(dirname);
dir.Traverse(traverser);
```

**Derived from**

No base class

## Constants

The elements of `wxDirTraverseResult` are the possible return values of the callback functions:

```
enum wxDirTraverseResult
{
    wxDIR_IGNORE = -1,      // ignore this directory but continue with
    others
    wxDIR_STOP,             // stop traversing
    wxDIR_CONTINUE          // continue into this directory
};
```

## Include files

`<wx/dir.h>`

## `wxDirTraverser::OnDir`

**virtual `wxDirTraverseResult OnDir(const wxString& dirname)`**

This function is called for each directory. It may return `wxSIR_STOP` to abort traversing completely, `wxDIR_IGNORE` to skip this directory but continue with others or `wxDIR_CONTINUE` to enumerate all files and subdirectories in this directory.

This is a pure virtual function and must be implemented in the derived class.

## `wxDirTraverser::OnFile`

**virtual `wxDirTraverseResult OnFile(const wxString& filename)`**

This function is called for each file. It may return `wxDIR_STOP` to abort traversing (for example, if the file being searched is found) or `wxDIR_CONTINUE` to proceed.

This is a pure virtual function and must be implemented in the derived class.

## `wxDirTraverser::OnOpenError`

**virtual `wxDirTraverseResult OnOpenError(const wxString& openerrorname)`**

This function is called for each directory which we failed to open for enumerating. It may return `wxSIR_STOP` to abort traversing completely, `wxDIR_IGNORE` to skip this directory but continue with others or `wxDIR_CONTINUE` to retry opening this directory once again.

The base class version always returns `wxDIR_IGNORE`.

## `wxDisplay`

Determines the sizes and locations of displays connected to the system.

**Derived from**

None

**Include files**

<wx/display.h>

**See also**

*wxCliEntDisplayRect* (p. 1945), *wxDisplaySize* (p. 1945), *wxDisplaySizeMM* (p. 1945)

**wxDisplay::wxDisplay**

**wxDisplay(unsigned index = 0)**

Constructor, setting up a wxDisplay instance with the specified display.

**Parameters**

*index*

The index of the display to use. This must be non-negative and lower than the value returned by *GetCount()* (p. 521).

**wxDisplay::~~wxDisplay**

**void ~wxDisplay()**

Destructor.

**wxDisplay::ChangeMode**

**bool ChangeMode(const wxVideoMode& mode = wxDefaultVideoMode)**

Changes the video mode of this display to the mode specified in the mode parameter.

If wxDefaultVideoMode is passed in as the mode parameter, the defined behaviour is that wxDisplay will reset the video mode to the default mode used by the display. On Windows, the behavior is normal. However, there are differences on other platforms. On Unix variations using X11 extensions it should behave as defined, but some irregularities may occur.

On wxMac passing in wxDefaultVideoMode as the mode parameter does nothing. This happens because carbon no longer has access to DMUseScreenPrefs, an undocumented function that changed the video mode to the system default by using the system's 'scrn' resource.

**wxDisplay::GetClientArea**



**wxRect GetClientArea() const**

Returns the client area of the display. The client area is the part of the display available for the normal (non full screen) windows, usually it is the same as *GetGeometry* (p. 522) but it could be less if there is a taskbar (or equivalent) on this display.

**See also:**

*wxClientDisplayRect* (p. 1945)

**wxDisplay::GetCount****static unsigned GetCount()**

Returns the number of connected displays.

**wxDisplay::GetCurrentMode****wxVideoMode GetCurrentMode() const**

Returns the current video mode that this display is in.

**wxDisplay::GetDepth****int GetDepth() const**

Returns the bit depth of the display whose index was passed to the constructor.

**wxDisplay::GetFromPoint****static int GetFromPoint(const wxPoint& pt)**

Returns the index of the display on which the given point lies. Returns `wxNOT_FOUND` if the point is not on any connected display.

**Parameters**

*pt*

The point to locate.

**wxDisplay::GetFromWindow****static int GetFromWindow(wxWindow\* win)**

Returns the index of the display on which the given window lies.

If the window is on more than one display it gets the display that overlaps the window the most.

Returns `wxNOT_FOUND` if the window is not on any connected display.

**Parameters**

*win*

The window to locate.

**wxDisplay::GetGeometry**

**wxRect GetGeometry() const**

Returns the bounding rectangle of the display whose index was passed to the constructor.

**See also:**

*GetClientArea* (p. 520), *wxDisplaySize* (p. 1945)

**wxDisplay::GetModes**

**wxArrayVideoModes GetModes(const wxVideoMode& mode = wxDefaultVideoMode) const**

Fills and returns an array with all the video modes that are supported by this display, or video modes that are supported by this display and match the mode parameter (if mode is not wxDefaultVideoMode).

**wxDisplay::GetName**

**wxString GetName() const**

Returns the display's name. A name is not available on all platforms.

**wxDisplay::IsPrimary**

**bool IsPrimary()**

Returns true if the display is the primary display. The primary display is the one whose index is 0.

**wxDllLoader**

**Deprecation note:** This class is deprecated since version 2.4 and is not compiled in by default in version 2.6 and will be removed in 2.8. Please use *wxDynamicLibrary* (p. 564) instead.

wxDllLoader is a class providing an interface similar to Unix's `dlopen()`. It is used by the wxLibrary framework and manages the actual loading of shared libraries and the resolving of symbols in them. There are no instances of this class, it simply serves as a namespace for its static member functions.

Please note that class *wxDynamicLibrary* (p. 564) provides alternative, friendlier interface to wxDllLoader.

The terms *DLL* and *shared library/object* will both be used in the documentation to refer to the same thing: a `.dll` file under Windows or `.so` or `.sl` one under Unix.

Example of using this class to dynamically load the `strlen()` function:

```
#if defined(__WXMSW__)
    static const wxChar *LIB_NAME = _T("kernel32");
    static const wxChar *FUNC_NAME = _T("lstrlenA");
#elif defined(__UNIX__)
    static const wxChar *LIB_NAME = _T("/lib/libc-2.0.7.so");
    static const wxChar *FUNC_NAME = _T("strlen");
#endif

wxDllType dllHandle = wxDllLoader::LoadLibrary(LIB_NAME);
if ( !dllHandle )
{
    ... error ...
}
else
{
    typedef int (*strlenType)(char *);
    strlenType pfnStrlen =
    (strlenType)wxDllLoader::GetSymbol(dllHandle, FUNC_NAME);
    if ( !pfnStrlen )
    {
        ... error ...
    }
    else
    {
        {
            if ( pfnStrlen("foo") != 3 )
            {
                ... error ...
            }
            else
            {
                ... ok! ...
            }
        }
    }

    wxDllLoader::UnloadLibrary(dllHandle);
}
```

#### **Derived from**

No base class

#### **Include files**

<wx/dynlib.h>

#### **Data structures**

This header defines a platform-dependent `wxDllType` typedef which stores a handle to a loaded DLLs on the given platform.

**wxDllLoader::GetDllExt****static wxString GetDllExt()**

Returns the string containing the usual extension for shared libraries for the given systems (including the leading dot if not empty).

For example, this function will return ".dll" under Windows or (usually) ".so" under Unix.

**wxDllLoader::GetProgramHandle****wxDllType GetProgramHandle()**

This function returns a valid handle for the main program itself. Notice that the `NULL` return value is valid for some systems (i.e. doesn't mean that the function failed).

**NB:** This function is Unix specific. It will always fail under Windows or OS/2.

**wxDllLoader::GetSymbol****void \* GetSymbol(wxDllType dllHandle, const wxString& name)**

This function resolves a symbol in a loaded DLL, such as a variable or function name.

Returned value will be `NULL` if the symbol was not found in the DLL or if an error occurred.

**Parameters**

*dllHandle*

Valid handle previously returned by *LoadLibrary* (p. 524)

*name*

Name of the symbol.

**wxDllLoader::LoadLibrary****wxDllType LoadLibrary(const wxString & libname, bool\* success = NULL)**

This function loads a shared library into memory, with *libname* being the name of the library: it may be either the full name including path and (platform-dependent) extension, just the basename (no path and no extension) or a basename with extension. In the last two cases, the library will be searched in all standard locations.

Returns a handle to the loaded DLL. Use *success* parameter to test if it is valid. If the handle is valid, the library must be unloaded later with *UnloadLibrary* (p. 525).

**Parameters**

*libname*

Name of the shared object to load.

*success*

May point to a bool variable which will be set to true or false; may also be `NULL`.

### **wxDllLoader::UnloadLibrary**

**void UnloadLibrary(wxDllType *dllhandle*)**

This function unloads the shared library. The handle *dllhandle* must have been returned by *LoadLibrary* (p. 524) previously.

## **wxDocChildFrame**

The `wxDocChildFrame` class provides a default frame for displaying documents on separate windows. This class can only be used for SDI (not MDI) child frames.

The class is part of the document/view framework supported by `wxWidgets`, and cooperates with the *wxView* (p. 1780), *wxDocument* (p. 545), *wxDocManager* (p. 527) and *wxDocTemplate* (p. 540) classes.

See the example application in `samples/docview`.

### **Derived from**

*wxFrame* (p. 682)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### **Include files**

`<wx/docview.h>`

### **See also**

*Document/view overview* (p. 2131), *wxFrame* (p. 682)

### **wxDocChildFrame::m\_childDocument**

**wxDocument\* m\_childDocument**

The document associated with the frame.

### **wxDocChildFrame::m\_childView**

**wxView\* m\_childView**

The view associated with the frame.

**wxDocChildFrame::wxDocChildFrame**

```
wxDocChildFrame(wxDocument* doc, wxView* view, wxFrame* parent,  
wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const  
wxSize& size = wxDefaultSize, long style = wxDEFAULT_FRAME_STYLE, const  
wxString& name = "frame")
```

Constructor.

**wxDocChildFrame::~~wxDocChildFrame**

```
~wxDocChildFrame()
```

Destructor.

**wxDocChildFrame::GetDocument**

```
wxDocument* GetDocument() const
```

Returns the document associated with this frame.

**wxDocChildFrame::GetView**

```
wxView* GetView() const
```

Returns the view associated with this frame.

**wxDocChildFrame::OnActivate**

```
void OnActivate(wxActivateEvent event)
```

Sets the currently active view to be the frame's view. You may need to override (but still call) this function in order to set the keyboard focus for your subwindow.

**wxDocChildFrame::OnCloseWindow**

```
void OnCloseWindow(wxCloseEvent& event)
```

Closes and deletes the current view and document.

**wxDocChildFrame::SetDocument**

```
void SetDocument(wxDocument *doc)
```

Sets the document for this frame.

**wxDocChildFrame::SetView**

```
void SetView(wxView *view)
```

Sets the view for this frame.

## **wxDocManager**

The `wxDocManager` class is part of the document/view framework supported by `wxWidgets`, and cooperates with the `wxView` (p. 1780), `wxDocument` (p. 545) and `wxDocTemplate` (p. 540) classes.

### **Derived from**

`wxEvtHandler` (p. 576)  
`wxObject` (p. 1148)

### **Include files**

<wx/docview.h>

### **See also**

`wxDocManager` overview (p. 2135), `wxDocument` (p. 545), `wxView` (p. 1780),  
`wxDocTemplate` (p. 540), `wxFileHistory` (p. 605)

### **wxDocManager::m\_currentView**

**wxView\* m\_currentView**

The currently active view.

### **wxDocManager::m\_defaultDocumentNameCounter**

**int m\_defaultDocumentNameCounter**

Stores the integer to be used for the next default document name.

### **wxDocManager::m\_fileHistory**

**wxFileHistory\* m\_fileHistory**

A pointer to an instance of `wxFileHistory` (p. 605), which manages the history of recently-visited files on the File menu.

### **wxDocManager::m\_maxDocsOpen**

**int m\_maxDocsOpen**

Stores the maximum number of documents that can be opened before existing documents are closed. By default, this is 10,000.

**wxDocManager::m\_docs****wxList m\_docs**

A list of all documents.

**wxDocManager::m\_flags****long m\_flags**

Stores the flags passed to the constructor.

**wxDocManager::m\_lastDirectory**

The directory last selected by the user when opening a file.

**wxFileHistory\* m\_fileHistory****wxDocManager::m\_templates****wxList m\_templates**

A list of all document templates.

**wxDocManager::wxDocManager**

**wxDocManager**(*long flags* = *wxDEFAULT\_DOCMAN\_FLAGS*, *bool initialize* = *true*)

Constructor. Create a document manager instance dynamically near the start of your application before doing any document or view operations.

*flags* is currently unused.

If *initialize* is true, the *Initialize* (p. 532) 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()**

Destructor.

**wxDocManager::ActivateView**

**void ActivateView**(*wxView\* doc*, *bool activate* = *true*)

Sets the current view.



**wxDocManager::AddDocument****void AddDocument(wxDocument \*doc)**

Adds the document to the list of documents.

**wxDocManager::AddFileToHistory****void AddFileToHistory(const wxString& filename)**

Adds a file to the file history list, if we have a pointer to an appropriate file menu.

**wxDocManager::AssociateTemplate****void AssociateTemplate(wxDocTemplate \*temp)**

Adds the template to the document manager's template list.

**wxDocManager::CloseDocuments****bool CloseDocuments(bool force = true)**

Closes all currently opened documents.

**wxDocManager::CreateDocument****wxDocument\* CreateDocument(const wxString& path, long flags)**

Creates a new document in a manner determined by the *flags* parameter, which can be:

- wxDOC\_NEW Creates a fresh document.
- wxDOC\_SILENT Silently loads the given document file.

If wxDOC\_NEW is present, a new document will be created and returned, possibly after asking the user for a template to use if there is more than one document template. If wxDOC\_SILENT is present, a new document will be created and the given file loaded into it. If neither of these flags is present, the user will be presented with a file selector for the file to load, and the template to use will be determined by the extension (Windows) or by popping up a template choice list (other platforms).

If the maximum number of documents has been reached, this function will delete the oldest currently loaded document before creating a new one.

**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::FileHistoryAddFilesToMenu****void FileHistoryAddFilesToMenu()**

Appends the files in the history list, to all menus managed by the file history object.

**void FileHistoryAddFilesToMenu(wxMenu\* menu)**

Appends the files in the history list, to the given menu only.

**wxDocManager::FileHistoryLoad****void FileHistoryLoad(wxConfigBase& config)**

Loads the file history from a config object.

**See also**

*wxConfig* (p. 262)

**wxDocManager::FileHistoryRemoveMenu****void FileHistoryRemoveMenu(wxMenu\* menu)**

Removes the given menu from the list of menus managed by the file history object.

**wxDocManager::FileHistorySave****void FileHistorySave(wxConfigBase& resourceFile)**

Saves the file history into a config object. This must be called explicitly by the application.

**See also**

*wxConfig* (p. 262)

**wxDocManager::FileHistoryUseMenu****void FileHistoryUseMenu(wxMenu\* menu)**

Use this menu for appending recently-visited document filenames, for convenient access. Calling this function with a valid menu pointer enables the history list functionality.

Note that you can add multiple menus using this function, to be managed by the file history object.

**wxDocManager::FindTemplateForPath****wxDocTemplate \* FindTemplateForPath(const wxString& path)**

Given a path, try to find template that matches the extension. This is only an approximate method of finding a template for creating a document.

**wxDocManager::GetCurrentDocument****wxDocument \* GetCurrentDocument()**

Returns the document associated with the currently active view (if any).

**wxDocManager::GetCurrentView****wxView \* GetCurrentView()**

Returns the currently active view

**wxDocManager::GetDocuments****wxList& GetDocuments()**

Returns a reference to the list of documents.

**wxDocManager::GetFileHistory****wxFileHistory \* GetFileHistory()**

Returns a pointer to file history.

**wxDocManager::GetLastDirectory****wxString GetLastDirectory() const**

Returns the directory last selected by the user when opening a file. Initially empty.

**wxDocManager::GetMaxDocsOpen****int GetMaxDocsOpen()**

Returns the number of documents that can be open simultaneously.

**wxDocManager::GetHistoryFilesCount****size\_t GetHistoryFilesCount()**

Returns the number of files currently stored in the file history.

**wxDocManager::GetTemplates**

**wxList& GetTemplates()**

Returns a reference to the list of associated templates.

**wxDocManager::Initialize****bool Initialize()**

Initializes data; currently just calls `OnCreateFileHistory`. Some data cannot always be initialized in the constructor because the programmer must be given the opportunity to override functionality. If `OnCreateFileHistory` was called from the constructor, an overridden virtual `OnCreateFileHistory` would not be called due to C++'s 'interesting' constructor semantics. In fact `Initialize` is called from the `wxDocManager` constructor, but this can be vetoed by passing false to the second argument, allowing the derived class's constructor to call `Initialize`, possibly calling a different `OnCreateFileHistory` from the default.

The bottom line: if you're not deriving from `Initialize`, forget it and construct `wxDocManager` with no arguments.

**wxDocManager::MakeDefaultName****bool MakeDefaultName(const wxString& buf)**

This method is preserved for backwards compatibility in `wxWidgets` 2.8 but is renamed to *MakeNewDocumentName* (p. 532) in `wxWidgets` 3.0, please see its description for the details.

This function simply copies the value returned by `MakeNewDocumentName()` into the provided *buf* and returns `true`.

**wxPerl note:** In `wxPerl` this function must return the modified name rather than just modifying the argument.

**wxDocManager::MakeNewDocumentName****wxString MakeNewDocumentName()**

Returns the name to be used for a new document. The default implementation appends an integer counter to the string **unnamed** and increments the counter. To customize this behaviour, you need to override the virtual *MakeNewDocumentName* (p. 532) method but if you only need to call this method, and not to override it, please use this method which is forward-compatible with `wxWidgets` 3.0.

This function is new since `wxWidgets` version 2.8.8

**wxDocManager::OnCreateFileHistory****wxFileHistory \* OnCreateFileHistory()**

A hook to allow a derived class to create a different type of file history. Called from *Initialize*

(p. 532).

**wxDocManager::OnFileClose****void OnFileClose(wxCommandEvent& event)**

Closes and deletes the currently active document.

**wxDocManager::OnFileCloseAll****void OnFileCloseAll(wxCommandEvent& event)**

Closes and deletes all the currently opened documents.

**wxDocManager::OnFileNew****void OnFileNew(wxCommandEvent& event)**

Creates a document from a list of templates (if more than one template).

**wxDocManager::OnFileOpen****void OnFileOpen(wxCommandEvent& event)**

Creates a new document and reads in the selected file.

**wxDocManager::OnFileRevert****void OnFileRevert(wxCommandEvent& event)**

Reverts the current document by calling wxDocument::Revert for the current document.

**wxDocManager::OnFileSave****void OnFileSave(wxCommandEvent& event)**

Saves the current document by calling wxDocument::Save for the current document.

**wxDocManager::OnFileSaveAs****void OnFileSaveAs(wxCommandEvent& event)**

Calls wxDocument::SaveAs for the current document.

**wxDocManager::RemoveDocument****void RemoveDocument(wxDocument \*doc)**

Removes the document from the list of documents.

**wxDocManager::SelectDocumentPath**

**wxDocTemplate \* SelectDocumentPath(wxDocTemplate \*\*templates, int noTemplates, wxString& path, 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.

**wxPerl note:** In wxPerl `templates` is a reference to a list of templates. If you override this method in your document manager it must return two values, eg:

```
(doctemplate, path) = My::DocManager->SelectDocumentPath( ... );
```

**wxDocManager::SelectDocumentType**

**wxDocTemplate \* SelectDocumentType(wxDocTemplate \*\*templates, int noTemplates, bool sort=false)**

Returns a document template by asking the user (if there is more than one template). This function is used in wxDocManager::CreateDocument.

**Parameters**

*templates*

Pointer to an array of templates from which to choose a desired template.

*noTemplates*

Number of templates being pointed to by the *templates* pointer.

*sort*

If more than one template is passed in in *templates*, then this parameter indicates whether the list of templates that the user will have to choose from is sorted or not when shown the choice box dialog. Default is false.

**wxPerl note:** In wxPerl `templates` is a reference to a list of templates.

**wxDocManager::SelectViewType**

**wxDocTemplate \* SelectViewType(wxDocTemplate \*\*templates, int noTemplates, bool sort=false)**

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 will not appear because the array of templates only contains those relevant to the document in question, and often there will only be one such.

**Parameters***templates*

Pointer to an array of templates from which to choose a desired template.

*noTemplates*

Number of templates being pointed to by the *templates* pointer.

*sort*

If more than one template is passed in in *templates*, then this parameter indicates whether the list of templates that the user will have to choose from is sorted or not when shown the choice box dialog. Default is false.

**wxPerl note:** In wxPerl *templates* is a reference to a list of templates.

**wxDocManager::SetLastDirectory**

**void SetLastDirectory(const wxString& dir)**

Sets the directory to be displayed to the user when opening a file. Initially this is empty.

**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.

**wxDocMDIChildFrame**

The wxDocMDIChildFrame class provides a default frame for displaying documents on separate windows. This class can only be used for MDI child frames.

The class is part of the document/view framework supported by wxWidgets, and cooperates with the *wxView* (p. 1780), *wxDocument* (p. 545), *wxDocManager* (p. 527) and *wxDocTemplate* (p. 540) classes.

See the example application in `samples/docview`.

**Derived from**

*wxMDIChildFrame* (p. 1047)

*wxFrame* (p. 682)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

**Include files**

<wx/docmdi.h>

**See also**

*Document/view overview* (p. 2131), *wxMDIChildFrame* (p. 1047)

**wxDocMDIChildFrame::m\_childDocument**

**wxDocument\*** m\_childDocument

The document associated with the frame.

**wxDocMDIChildFrame::m\_childView**

**wxView\*** m\_childView

The view associated with the frame.

**wxDocMDIChildFrame::wxDocMDIChildFrame**

**wxDocMDIChildFrame**(wxDocument\* doc, wxView\* view, wxFrame\* parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT\_FRAME\_STYLE, const wxString& name = "frame")

Constructor.

**wxDocMDIChildFrame::~~wxDocMDIChildFrame**

**~wxDocMDIChildFrame**()

Destructor.

**wxDocMDIChildFrame::GetDocument**

**wxDocument\*** GetDocument() const

Returns the document associated with this frame.

**wxDocMDIChildFrame::GetView**

**wxView\*** GetView() const

Returns the view associated with this frame.

**wxDocMDIChildFrame::OnActivate**

**void** OnActivate(wxActivateEvent event)



Sets the currently active view to be the frame's view. You may need to override (but still call) this function in order to set the keyboard focus for your subwindow.

#### **wxDocMDIChildFrame::OnCloseWindow**

**void OnCloseWindow(wxCloseEvent& event)**

Closes and deletes the current view and document.

#### **wxDocMDIChildFrame::SetDocument**

**void SetDocument(wxDocument \*doc)**

Sets the document for this frame.

#### **wxDocMDIChildFrame::SetView**

**void SetView(wxView \*view)**

Sets the view for this frame.

### **wxDocMDIParentFrame**

The `wxDocMDIParentFrame` class provides a default top-level frame for applications using the document/view framework. This class can only be used for MDI parent frames.

It cooperates with the `wxView` (p. 1780), `wxDocument` (p. 545), `wxDocManager` (p. 527) and `wxDocTemplates` (p. 540) classes.

See the example application in `samples/docview`.

#### **Derived from**

`wxMDIParentFrame` (p. 1051)  
`wxFrame` (p. 682)  
`wxWindow` (p. 1795)  
`wxEvtHandler` (p. 576)  
`wxObject` (p. 1148)

#### **Include files**

`<wx/docmdi.h>`

#### **See also**

*Document/view overview* (p. 2131), `wxMDIParentFrame` (p. 1051)

#### **wxDocMDIParentFrame::wxDocMDIParentFrame**

**wxDocMDIParentFrame()**

```
wxDocMDIParentFrame(wxDocManager* manager, wxFrame *parent, wxWindowID
id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize, long style = wxDEFAULT_FRAME_STYLE, const wxString& name =
"frame")
```

Constructor.

**wxDocMDIParentFrame::~~wxDocMDIParentFrame****~wxDocMDIParentFrame()**

Destructor.

**wxDocMDIParentFrame::Create**

```
bool Create(wxDocManager* manager, wxFrame *parent, wxWindowID id, const
wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize, long style = wxDEFAULT_FRAME_STYLE, const wxString& name =
"frame")
```

Creates the window.

**wxDocMDIParentFrame::OnCloseWindow****void OnCloseWindow(wxCloseEvent& event)**

Deletes all views and documents. If no user input cancelled the operation, the frame will be destroyed and the application will exit.

Since understanding how document/view clean-up takes place can be difficult, the implementation of this function is shown below.

```
void wxDocParentFrame::OnCloseWindow(wxCloseEvent& event)
{
    if (m_docManager->Clear(!event.CanVeto()))
    {
        this->Destroy();
    }
    else
        event.Veto();
}
```

**wxDocParentFrame**

The `wxDocParentFrame` class provides a default top-level frame for applications using the document/view framework. This class can only be used for SDI (not MDI) parent frames.

It cooperates with the `wxView` (p. 1780), `wxDocument` (p. 545), `wxDocManager` (p. 527) and `wxDocTemplates` (p. 540) classes.

See the example application in `samples/docview`.

### Derived from

`wxFrame` (p. 682)  
`wxWindow` (p. 1795)  
`wxEvtHandler` (p. 576)  
`wxObject` (p. 1148)

### Include files

`<wx/docview.h>`

### See also

*Document/view overview* (p. 2131), `wxFrame` (p. 682)

## **wxDocParentFrame::wxDocParentFrame**

### **wxDocParentFrame()**

Default constructor.

**wxDocParentFrame(wxDocManager\* manager, wxFrame \*parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT\_FRAME\_STYLE, const wxString& name = "frame")**

Constructor.

## **wxDocParentFrame::~~wxDocParentFrame**

### **~wxDocParentFrame()**

Destructor.

## **wxDocParentFrame::Create**

**bool Create(wxDocManager\* manager, wxFrame \*parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT\_FRAME\_STYLE, const wxString& name = "frame")**

Used in two-step construction.

## **wxDocParentFrame::GetDocumentManager**

### **wxDocManager \* GetDocumentManager() const**

Returns the associated *document manager object* (p. 527).

**wxDocParentFrame::OnCloseWindow****void OnCloseWindow(wxCloseEvent& event)**

Deletes all views and documents. If no user input cancelled the operation, the frame will be destroyed and the application will exit.

Since understanding how document/view clean-up takes place can be difficult, the implementation of this function is shown below.

```
void wxDocParentFrame::OnCloseWindow(wxCloseEvent& event)
{
    if (m_docManager->Clear(!event.CanVeto()))
    {
        this->Destroy();
    }
    else
        event.Veto();
}
```

**wxDocTemplate**

The wxDocTemplate class is used to model the relationship between a document class and a view class.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/docview.h>

**See also**

*wxDocTemplate overview* (p. 2134), *wxDocument* (p. 545), *wxView* (p. 1780)

**wxDocTemplate::m\_defaultExt****wxString m\_defaultExt**

The default extension for files of this type.

**wxDocTemplate::m\_description****wxString m\_description**

A short description of this template.

**wxDocTemplate::m\_directory**

**wxString m\_directory**

The default directory for files of this type.

**wxDocTemplate::m\_docClassInfo****wxClassInfo\* m\_docClassInfo**

Run-time class information that allows document instances to be constructed dynamically.

**wxDocTemplate::m\_docTypeName****wxString m\_docTypeName**

The named type of the document associated with this template.

**wxDocTemplate::m\_documentManager****wxDocTemplate\* m\_documentManager**

A pointer to the document manager for which this template was created.

**wxDocTemplate::m\_fileFilter****wxString m\_fileFilter**

The file filter (such as \*.txt) to be used in file selector dialogs.

**wxDocTemplate::m\_flags****long m\_flags**

The flags passed to the constructor.

**wxDocTemplate::m\_viewClassInfo****wxClassInfo\* m\_viewClassInfo**

Run-time class information that allows view instances to be constructed dynamically.

**wxDocTemplate::m\_viewTypeName****wxString m\_viewTypeName**

The named type of the view associated with this template.

**wxDocTemplate::wxDocTemplate**

**wxDocTemplate(wxDocManager\* manager, const wxString& descr, const wxString& filter, const wxString& dir, const wxString& ext, const wxString&**

*docTypeName*, **const wxString&** *viewTypeName*, **wxClassInfo\*** *docClassInfo* = NULL,  
**wxClassInfo\*** *viewClassInfo* = NULL, **long** *flags* = wxDEFAULT\_TEMPLATE\_FLAGS)

Constructor. Create instances dynamically near the start of your application after creating a wxDocManager instance, and before doing any document or view operations.

*manager* is the document manager object which manages this template.

*descr* is a short description of what the template is for. This string will be displayed in the file filter list of Windows file selectors.

*filter* is an appropriate file filter such as \*.txt.

*dir* is the default directory to use for file selectors.

*ext* is the default file extension (such as txt).

*docTypeName* is a name that should be unique for a given type of document, used for gathering a list of views relevant to a particular document.

*viewTypeName* is a name that should be unique for a given view.

*docClassInfo* is a pointer to the run-time document class information as returned by the CLASSINFO macro, e.g. CLASSINFO(MyDocumentClass). If this is not supplied, you will need to derive a new wxDocTemplate class and override the CreateDocument member to return a new document instance on demand.

*viewClassInfo* is a pointer to the run-time view class information as returned by the CLASSINFO macro, e.g. CLASSINFO(MyViewClass). If this is not supplied, you will need to derive a new wxDocTemplate class and override the CreateView member to return a new view instance on demand.

*flags* is a bit list of the following:

- wxTEMPLATE\_VISIBLE The template may be displayed to the user in dialogs.
- wxTEMPLATE\_INVISIBLE The template may not be displayed to the user in dialogs.
- wxDEFAULT\_TEMPLATE\_FLAGS Defined as wxTEMPLATE\_VISIBLE.

**wxPerl note:** In wxPerl *docClassInfo* and *viewClassInfo* can be either `Wx::ClassInfo` objects or strings which contain the name of the perl packages which are to be used as `Wx::Document` and `Wx::View` classes (they must have a constructor named `new`):

```
Wx::DocTemplate->new( docmgr, descr, filter, dir, ext, docTypeName,  
                    viewTypeName, docClassInfo,  
                    viewClassInfo, flags )    will  
                                              construct document and view  
                                              objects from the class information
```

```
Wx::DocTemplate->new( docmgr, descr, filter, dir, ext, docTypeName,  
                    viewTypeName, docClassName,
```

**viewClassName, flags )** will  
construct document and view  
objects from perl packages

**Wx::DocTemplate->new( docmgr, descr, filter, dir, ext, docTypeName,  
viewTypeName )**  
Wx::DocTemplate::Crea  
teDocument()  
andWx::DocTemplate::Create  
View() must be overridden

### **wxDocTemplate::~~wxDocTemplate**

**void ~wxDocTemplate()**

Destructor.

### **wxDocTemplate::CreateDocument**

**wxDocument \* CreateDocument(const wxString& path, long flags = 0)**

Creates a new instance of the associated document class. If you have not supplied a `wxClassInfo` parameter to the template constructor, you will need to override this function to return an appropriate document instance.

This function calls `wxDocTemplate::InitDocument` which in turns calls `wxDocument::OnCreate`.

### **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**

**wxString GetDefaultExtension()**

Returns the default file extension for the document data, as passed to the document template constructor.

### **wxDocTemplate::GetDescription**

**wxString GetDescription()**

Returns the text description of this template, as passed to the document template constructor.

**wxDocTemplate::GetDirectory****wxString GetDirectory()**

Returns the default directory, as passed to the document template constructor.

**wxDocTemplate::GetDocumentManager****wxDocManager \* GetDocumentManager()**

Returns a pointer to the document manager instance for which this template was created.

**wxDocTemplate::GetDocumentName****wxString GetDocumentName()**

Returns the document type name, as passed to the document template constructor.

**wxDocTemplate::GetFileFilter****wxString GetFileFilter()**

Returns the file filter, as passed to the document template constructor.

**wxDocTemplate::GetFlags****long GetFlags()**

Returns the flags, as passed to the document template constructor.

**wxDocTemplate::GetViewName****wxString GetViewName()**

Returns the view type name, as passed to the document template constructor.

**wxDocTemplate::InitDocument****bool InitDocument(wxDocument\* doc, const wxString& path, long flags = 0)**

Initialises the document, calling wxDocument::OnCreate. This is called from wxDocTemplate::CreateDocument.

**wxDocTemplate::IsVisible****bool IsVisible()**

Returns true if the document template can be shown in user dialogs, false otherwise.

**wxDocTemplate::SetDefaultExtension**



**void SetDefaultExtension(const wxString& ext)**

Sets the default file extension.

**wxDocTemplate::SetDescription**

**void SetDescription(const wxString& descr)**

Sets the template description.

**wxDocTemplate::SetDirectory**

**void SetDirectory(const wxString& 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(const wxString& filter)**

Sets the file filter.

**wxDocTemplate::SetFlags**

**void SetFlags(long flags)**

Sets the internal document template flags (see the constructor description for more details).

## wxDocument

The document class can be used to model an application's file-based data. It is part of the document/view framework supported by wxWidgets, and cooperates with the *wxView* (p. 1780), *wxDocTemplate* (p. 540) and *wxDocManager* (p. 527) classes.

### Derived from

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### Include files

<wx/docview.h>

**See also**

*wxDocument overview* (p. 2133), *wxView* (p. 1780), *wxDocTemplate* (p. 540), *wxDocManager* (p. 527)

**wxDocument::m\_commandProcessor****wxCommandProcessor\* m\_commandProcessor**

A pointer to the command processor associated with this document.

**wxDocument::m\_documentFile****wxString m\_documentFile**

Filename associated with this document ("" if none).

**wxDocument::m\_documentModified****bool m\_documentModified**

true if the document has been modified, false otherwise.

**wxDocument::m\_documentTemplate****wxDocTemplate \* m\_documentTemplate**

A pointer to the template from which this document was created.

**wxDocument::m\_documentTitle****wxString m\_documentTitle**

Document title. The document title is used for an associated frame (if any), and is usually constructed by the framework from the filename.

**wxDocument::m\_documentTypeName****wxString m\_documentTypeName**

The document type name given to the *wxDocTemplate* constructor, copied to this variable when the document is created. If several document templates are created that use the same document type, this variable is used in *wxDocManager::CreateView* to collate a list of alternative view types that can be used on this kind of document. Do not change the value of this variable.

**wxDocument::m\_documentViews**

**wxList m\_documentViews**

List of wxView instances associated with this document.

**wxDocument::wxDocument****wxDocument()**

Constructor. Define your own default constructor to initialize application-specific data.

**wxDocument::~~wxDocument****~wxDocument()**

Destructor. Removes itself from the document manager.

**wxDocument::AddView****virtual bool AddView(wxView \*view)**

If the view is not already in the list of views, adds the view and calls OnChangedViewList.

**wxDocument::Close****virtual bool Close()**

Closes the document, by calling OnSaveModified and then (if this returned true) OnCloseDocument. This does not normally delete the document object: use DeleteAllViews to do this implicitly.

**wxDocument::DeleteAllViews****virtual bool DeleteAllViews()**

Calls wxView::Close and deletes each view. Deleting the final view will implicitly delete the document itself, because the wxView destructor calls RemoveView. This in turns calls wxDocument::OnChangedViewList, whose default implementation is to save and delete the document if no views exist.

**wxDocument::GetCommandProcessor****wxCommandProcessor\* GetCommandProcessor() const**

Returns a pointer to the command processor associated with this document.

See *wxCommandProcessor* (p. 255).

**wxDocument::GetDocumentTemplate****wxDocTemplate\* GetDocumentTemplate() const**

Gets a pointer to the template that created the document.

### **wxDocument::GetDocumentManager**

**wxDocManager\* GetDocumentManager() const**

Gets a pointer to the associated document manager.

### **wxDocument::GetDocumentName**

**wxString GetDocumentName() const**

Gets the document type name for this document. See the comment for *documentTypeName* (p. 546).

### **wxDocument::GetDocumentWindow**

**wxWindow\* GetDocumentWindow() const**

Intended to return a suitable window for using as a parent for document-related dialog boxes. By default, uses the frame associated with the first view.

### **wxDocument::GetFilename**

**wxString GetFilename() const**

Gets the filename associated with this document, or "" if none is associated.

### **wxDocument::GetFirstView**

**wxView \* GetFirstView() const**

A convenience function to get the first view for a document, because in many cases a document will only have a single view.

See also: *GetViews* (p. 549)

### **wxDocument::GetPrintableName**

**virtual void GetPrintableName(wxString& name) const**

Copies a suitable document name into the supplied *name* buffer. The default function uses the title, or if there is no title, uses the filename; or if no filename, the string **unnamed**.

**wxPerl note:** In wxPerl this function must return the modified name rather than just modifying the argument.

### **wxDocument::GetTitle**

**wxString GetTitle() const**

Gets the title for this document. The document title is used for an associated frame (if any), and is usually constructed by the framework from the filename.

**wxDocument::GetViews****wxList & GetViews() const**

Returns the list whose elements are the views on the document.

See also: *GetFirstView* (p. 548)

**wxDocument::IsModified****virtual bool IsModified() const**

Returns true if the document has been modified since the last save, false otherwise. You may need to override this if your document view maintains its own record of being modified (for example if using *wxTextWindow* to view and edit the document).

See also *Modify* (p. 549).

**wxDocument::LoadObject****virtual istream& LoadObject(istream& stream)****virtual wxInputStream& LoadObject(wxInputStream& 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.

Note that only one of these forms exists, depending on how *wxWidgets* was configured.

**wxDocument::Modify****virtual void Modify(bool modify)**

Call with true to mark the document as modified since the last save, false otherwise. You may need to override this if your document view maintains its own record of being modified (for example if using *wxTextWindow* to view and edit the document).

See also *IsModified* (p. 549).

**wxDocument::OnChangedViewList****virtual void OnChangedViewList()**

Called when a view is added to or deleted from this document. The default implementation saves and deletes the document if no views exist (the last one has just been removed).

**wxDocument::OnCloseDocument****virtual bool OnCloseDocument()**

The default implementation calls `DeleteContents` (an empty implementation) sets the modified flag to false. Override this to supply additional behaviour when the document is closed with `Close`.

**wxDocument::OnCreate****virtual bool OnCreate(const wxString& path, long flags)**

Called just after the document object is created to give it a chance to initialize itself. The default implementation uses the template associated with the document to create an initial view. If this function returns false, the document is deleted.

**wxDocument::OnCreateCommandProcessor****virtual wxCommandProcessor\* OnCreateCommandProcessor()**

Override this function if you want a different (or no) command processor to be created when the document is created. By default, it returns an instance of `wxCommandProcessor`.

See *wxCommandProcessor* (p. 255).

**wxDocument::OnNewDocument****virtual bool OnNewDocument()**

The default implementation calls `OnSaveModified` and `DeleteContents`, makes a default title for the document, and notifies the views that the filename (in fact, the title) has changed.

**wxDocument::OnOpenDocument****virtual bool OnOpenDocument(const wxString& filename)**

Constructs an input file stream for the given filename (which must not be empty), and calls `LoadObject`. If `LoadObject` returns true, the document is set to unmodified; otherwise, an error message box is displayed. The document's views are notified that the filename has changed, to give windows an opportunity to update their titles. All of the document's views are then updated.

**wxDocument::OnSaveDocument****virtual bool OnSaveDocument(const wxString& filename)**

Constructs an output file stream for the given filename (which must not be empty), and calls `SaveObject`. If `SaveObject` returns true, the document is set to unmodified; otherwise, an error message box is displayed.

**wxDocument::OnSaveModified****virtual bool OnSaveModified()**

If the document has been modified, prompts the user to ask if the changes should be changed. If the user replies Yes, the Save function is called. If No, the document is marked as unmodified and the function succeeds. If Cancel, the function fails.

**wxDocument::RemoveView****virtual bool RemoveView(wxView\* view)**

Removes the view from the document's list of views, and calls OnChangedViewList.

**wxDocument::Save****virtual bool Save()**

Saves the document by calling OnSaveDocument if there is an associated filename, or SaveAs if there is no filename.

**wxDocument::SaveAs****virtual bool SaveAs()**

Prompts the user for a file to save to, and then calls OnSaveDocument.

**wxDocument::SaveObject****virtual ostream& SaveObject(ostream& stream)****virtual wxOutputStream& SaveObject(wxOutputStream& 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.

Note that only one of these forms exists, depending on how wxWidgets was configured.

**wxDocument::SetCommandProcessor****virtual void SetCommandProcessor(wxCommandProcessor \*processor)**

Sets the command processor to be used for this document. The document will then be responsible for its deletion. Normally you should not call this; override OnCreateCommandProcessor instead.

See *wxCommandProcessor* (p. 255).

**wxDocument::SetDocumentName**

**void SetDocumentName(const wxString& name)**

Sets the document type name for this document. See the comment for *documentTypeName* (p. 546).

**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(const wxString& filename, bool notifyViews = false)**

Sets the filename for this document. Usually called by the framework.

If *notifyViews* is true, *wxView::OnChangeFilename* is called for all views.

**wxDocument::SetTitle**

**void SetTitle(const wxString& title)**

Sets the title for this document. The document title is used for an associated frame (if any), and is usually constructed by the framework from the filename.

**wxDocument::UpdateAllViews**

**void UpdateAllViews(wxView\* sender = NULL, wxObject\* hint = NULL)**

Updates all views. If *sender* is non-NULL, does not update this view.

*hint* represents optional information to allow a view to optimize its update.

## wxDragImage

This class is used when you wish to drag an object on the screen, and a simple cursor is not enough.

On Windows, the WIN32 API is used to do achieve smooth dragging. On other platforms, *wxGenericDragImage* is used. Applications may also prefer to use *wxGenericDragImage* on Windows, too.

**wxPython note:** wxPython uses *wxGenericDragImage* on all platforms, but uses the *wxDragImage* name.

To use this class, when you wish to start dragging an image, create a *wxDragImage* object and store it somewhere you can access it as the drag progresses. Call *BeginDrag* to start, and *EndDrag* to stop the drag. To move the image, initially call *Show* and then *Move*. If you wish to update the screen contents during the drag (for example, highlight an item as in the



dragimag sample), first call `Hide`, update the screen, call `Move`, and then call `Show`.

You can drag within one window, or you can use full-screen dragging either across the whole screen, or just restricted to one area of the screen to save resources. If you want the user to drag between two windows, then you will need to use full-screen dragging.

If you wish to draw the image yourself, use `wxGenericDragImage` and override `wxDragImage::DoDrawImage` (p. 555) and `wxDragImage::GetImageRect` (p. 556).

Please see `samples/dragimag` for an example.

### Derived from

`wxObject` (p. 1148)

### Include files

`<wx/dragimag.h>`  
`<wx/generic/dragimgg.h>`

## **wxDragImage::wxDragImage**

### **wxDragImage()**

Default constructor.

**wxDragImage(const wxBitmap& image, const wxCursor& cursor = wxNullCursor, const wxPoint& cursorHotspot = wxPoint(0, 0))**

Constructs a drag image from a bitmap and optional cursor.

**wxDragImage(const wxIcon& image, const wxCursor& cursor = wxNullCursor, const wxPoint& cursorHotspot = wxPoint(0, 0))**

Constructs a drag image from an icon and optional cursor.

**wxPython note:** This constructor is called `wxDragIcon` in wxPython.

**wxDragImage(const wxString& text, const wxCursor& cursor = wxNullCursor, const wxPoint& cursorHotspot = wxPoint(0, 0))**

Constructs a drag image from a text string and optional cursor.

**wxPython note:** This constructor is called `wxDragString` in wxPython.

**wxDragImage(const wxTreeCtrl& treeCtrl, wxTreeItemId& id)**

Constructs a drag image from the text in the given tree control item, and optional cursor.

**wxPython note:** This constructor is called `wxDragTreeItem` in wxPython.

**wxDragImage(const wxListCtrl& treeCtrl, long id)**

Constructs a drag image from the text in the given tree control item, and optional cursor.

**wxPython note:** This constructor is called `wxDragListItem` in wxPython.

**wxDragImage(const wxCursor& cursor = wxNullCursor, const wxPoint& cursorHotspot = wxPoint(0, 0))**

Constructs a drag image an optional cursor. This constructor is only available for `wxGenericDragImage`, and can be used when the application supplies `wxDragImage::DoDrawImage` (p. 555) and `wxDragImage::GetImageRect` (p. 556).

### Parameters

*image*

Icon or bitmap to be used as the drag image. The bitmap can have a mask.

*text*

Text used to construct a drag image.

*cursor*

Optional cursor to combine with the image.

*hotspot*

This parameter is deprecated.

*treeCtrl*

Tree control for constructing a tree drag image.

*listCtrl*

List control for constructing a list drag image.

*id*

Tree or list control item id.

### **wxDragImage::BeginDrag**

**bool BeginDrag(const wxPoint& hotspot, wxWindow\* window, bool fullScreen = false, wxRect\* rect = NULL)**

Start dragging the image, in a window or full screen.

**bool BeginDrag(const wxPoint& hotspot, wxWindow\* window, wxWindow\* boundingWindow)**

Start dragging the image, using the first window to capture the mouse and the second to specify the bounding area. This form is equivalent to using the first form, but more convenient than working out the bounding rectangle explicitly.

You need to then call `wxDragImage::Show` (p. 556) and `wxDragImage::Move` (p. 556) to show the image on the screen.

Call `wxDragImage::EndDrag` (p. 555) when the drag has finished.

Note that this call automatically calls `CaptureMouse`.

### Parameters

#### *hotspot*

The location of the drag position relative to the upper-left corner of the image.

#### *window*

The window that captures the mouse, and within which the dragging is limited unless *fullScreen* is true.

#### *boundingWindow*

In the second form of the function, specifies the area within which the drag occurs.

#### *fullScreen*

If true, specifies that the drag will be visible over the full screen, or over as much of the screen as is specified by *rect*. Note that the mouse will still be captured in *window*.

#### *rect*

If non-NULL, specifies the rectangle (in screen coordinates) that bounds the dragging operation. Specifying this can make the operation more efficient by cutting down on the area under consideration, and it can also make a visual difference since the drag is clipped to this area.

### **wxDragImage::DoDrawImage**

**virtual bool DoDrawImage(wxDC& dc, const wxPoint& pos)**

Draws the image on the device context with top-left corner at the given position.

This function is only available with `wxGenericDragImage`, to allow applications to draw their own image instead of using an actual bitmap. If you override this function, you must also override `wxDragImage::GetImageRect` (p. 556).

### **wxDragImage::EndDrag**

**bool EndDrag()**

Call this when the drag has finished.

Note that this call automatically calls `ReleaseMouse`.

**wxDragImage::GetImageRect****virtual wxRect GetImageRect(const wxPoint& pos) const**

Returns the rectangle enclosing the image, assuming that the image is drawn with its top-left corner at the given point.

This function is available in `wxGenericDragImage` only, and may be overridden (together with `wxDragImage::DoDrawImage` (p. 555)) to provide a virtual drawing capability.

**wxDragImage::Hide****bool Hide()**

Hides the image. You may wish to call this before updating the window contents (perhaps highlighting an item). Then call `wxDragImage::Move` (p. 556) and `wxDragImage::Show` (p. 556).

**wxDragImage::Move****bool Move(const wxPoint& pt)**

Call this to move the image to a new position. The image will only be shown if `wxDragImage::Show` (p. 556) has been called previously (for example at the start of the drag).

*pt* is the position in client coordinates (relative to the window specified in `BeginDrag`).

You can move the image either when the image is hidden or shown, but in general dragging will be smoother if you move the image when it is shown.

**wxDragImage::Show****bool Show()**

Shows the image. Call this at least once when dragging.

**wxDragImage::UpdateBackingFromWindow****bool UpdateBackingFromWindow(wxDC& windowDC, wxMemoryDC& destDC, const wxRect& sourceRect, const wxRect& destRect) const**

Override this if you wish to draw the window contents to the backing bitmap yourself. This can be desirable if you wish to avoid flicker by not having to redraw the updated window itself just before dragging, which can cause a flicker just as the drag starts. Instead, paint the drag image's backing bitmap to show the appropriate graphic *minus the objects to be dragged*, and leave the window itself to be updated by the drag image. This can provide eerily smooth, flicker-free drag behaviour.

The default implementation copies the window contents to the backing bitmap. A new implementation will normally copy information from another source, such as from its own

backing bitmap if it has one, or directly from internal data structures.

This function is available in `wxGenericDragImage` only.

## **wxDropFilesEvent**

This class is used for drop files events, that is, when files have been dropped onto the window. This functionality is currently only available under Windows. The window must have previously been enabled for dropping by calling `wxWindow::DragAcceptFiles` (p. 1806).

Important note: this is a separate implementation to the more general drag and drop implementation documented *here* (p. 2150). It uses the older, Windows message-based approach of dropping files.

### **Derived from**

`wxEvent` (p. 572)

`wxObject` (p. 1148)

### **Include files**

`<wx/event.h>`

### **Event table macros**

To process a drop files event, use these event handler macros to direct input to a member function that takes a `wxDropFilesEvent` argument.

**EVT\_DROP\_FILES(func)**                      Process a `wxEVT_DROP_FILES` event.

### **See also**

*Event handling overview* (p. 2077)

## **wxDropFilesEvent::wxDropFilesEvent**

`wxDropFilesEvent(WXTYPE id = 0, int noFiles = 0, wxString* files = NULL)`

Constructor.

## **wxDropFilesEvent::m\_files**

`wxString* m_files`

An array of filenames.

## **wxDropFilesEvent::m\_noFiles**

`int m_noFiles`

The number of files dropped.

**wxDropFilesEvent::m\_pos**

**wxPoint m\_pos**

The point at which the drop took place.

**wxDropFilesEvent::GetFiles**

**wxString\* GetFiles() const**

Returns an array of filenames.

**wxDropFilesEvent::GetNumberOfFiles**

**int GetNumberOfFiles() const**

Returns the number of files dropped.

**wxDropFilesEvent::GetPosition**

**wxPoint GetPosition() const**

Returns the position at which the files were dropped.

Returns an array of filenames.

## **wxDropSource**

This class represents a source for a drag and drop operation.

See *Drag and drop overview* (p. 2150) and *wxDataObject overview* (p. 2151) for more information.

**Derived from**

None

**Include files**

<wx/dnd.h>

**Types**

wxDragResult is defined as follows:

```
enum wxDragResult
{
    wxDragError,      // error prevented the d&d operation from
completing
    wxDragNone,       // drag target didn't accept the data
}
```

```
        wxDragCopy,        // the data was successfully copied
        wxDragMove,        // the data was successfully moved (MSW only)
        wxDragLink,        // operation is a drag-link
        wxDragCancel       // the operation was cancelled by user (not an
    error)
};
```

### See also

*wxDropTarget* (p. 561), *wxTextDropTarget* (p. 1654), *wxFileDropTarget* (p. 604)

## wxDropSource::wxDropSource

**wxDropSource(wxWindow\* win = NULL, const wxIconOrCursor& iconCopy = wxNullIconOrCursor, const wxIconOrCursor& iconMove = wxNullIconOrCursor, const wxIconOrCursor& iconNone = wxNullIconOrCursor)**

**wxDropSource(wxDataObject& data, wxWindow\* win = NULL, const wxIconOrCursor& iconCopy = wxNullIconOrCursor, const wxIconOrCursor& iconMove = wxNullIconOrCursor, const wxIconOrCursor& iconNone = wxNullIconOrCursor)**

The constructors for wxDataObject.

If you use the constructor without *data* parameter you must call *SetData* (p. 560) later.

Note that the exact type of *iconCopy* and subsequent parameters differs between wxMSW and wxGTK: these are cursors under Windows but icons for GTK. You should use the macro *wxDROP\_ICON* (p. 1945) in portable programs instead of directly using either of these types.

### Parameters

*win*

The window which initiates the drag and drop operation.

*iconCopy*

The icon or cursor used for feedback for copy operation.

*iconMove*

The icon or cursor used for feedback for move operation.

*iconNone*

The icon or cursor used for feedback when operation can't be done.

*win* is the window which initiates the drag and drop operation.

## wxDropSource::~wxDropSource

**virtual ~wxDropSource()**

**wxDropSource::SetData**

**void SetData(wxDataObject& data)**

Sets the data *wxDataObject* (p. 311) associated with the drop source. This will not delete any previously associated data.

**wxDropSource::DoDragDrop**

**virtual wxDragResult DoDragDrop(int flags = wxDrag\_CopyOnly)**

Do it (call this in response to a mouse button press, for example). This starts the drag-and-drop operation which will terminate when the user releases the mouse.

**Parameters**

*flags*

If *wxDrag\_AllowMove* is included in the flags, data may be moved and not only copied (default). If *wxDrag\_DefaultMove* is specified (which includes the previous flag), this is even the default operation

.

**Return value**

Returns the operation requested by the user, may be *wxDragCopy*, *wxDragMove*, *wxDragLink*, *wxDragCancel* or *wxDragNone* if an error occurred.

**wxDropSource::GetDataObject**

**wxDataObject \* GetDataObject()**

Returns the *wxDataObject* object that has been assigned previously.

**wxDropSource::GiveFeedback**

**virtual bool GiveFeedback(wxDragResult effect)**

Overridable: you may give some custom UI feedback during the drag and drop operation in this function. It is called on each mouse move, so your implementation must not be too slow.

**Parameters**

*effect*

The effect to implement. One of *wxDragCopy*, *wxDragMove*, *wxDragLink* and *wxDragNone*.



*scrolling*

true if the window is scrolling. MSW only.

### Return value

Return false if you want default feedback, or true if you implement your own feedback. The return values is ignored under GTK.

## **wxDropSource::SetCursor**

**void SetCursor(wxDragResult res, const wxCursor& cursor)**

Set the icon to use for a certain drag result.

### Parameters

*res*

The drag result to set the icon for.

*cursor*

The ion to show when this drag result occurs.

## **wxDropTarget**

This class represents a target for a drag and drop operation. A *wxDataObject* (p. 311) can be associated with it and by default, this object will be filled with the data from the drag source, if the data formats supported by the data object match the drag source data format.

There are various virtual handler functions defined in this class which may be overridden to give visual feedback or react in a more fine-tuned way, e.g. by not accepting data on the whole window area, but only a small portion of it. The normal sequence of calls is *OnEnter* (p. 563), possibly many times *OnDragOver* (p. 563), *OnDrop* (p. 562) and finally *OnData* (p. 562).

See *Drag and drop overview* (p. 2150) and *wxDataObject overview* (p. 2151) for more information.

### Derived from

None

### Include files

<wx/dnd.h>

### Types

wxDragResult is defined as follows:

```
enum wxDragResult
{
    wxDragError,      // error prevented the d&d operation from
completing
    wxDragNone,       // drag target didn't accept the data
    wxDragCopy,       // the data was successfully copied
    wxDragMove,       // the data was successfully moved (MSW only)
    wxDragLink,       // operation is a drag-link
    wxDragCancel      // the operation was cancelled by user (not an
error)
};
```

**See also**

*wxDropSource* (p. 558), *wxTextDropTarget* (p. 1654), *wxFileDropTarget* (p. 604), *wxDataFormat* (p. 304), *wxDataObject* (p. 311)

**wxDropTarget::wxDropTarget**

**wxDropTarget(wxDataObject\* data = NULL)**

Constructor. *data* is the data to be associated with the drop target.

**wxDropTarget::~~wxDropTarget**

**~wxDropTarget()**

Destructor. Deletes the associated data object, if any.

**wxDropTarget::GetData**

**virtual void GetData()**

This method may only be called from within *OnData* (p. 562). By default, this method copies the data from the drop source to the *wxDataObject* (p. 311) associated with this drop target, calling its *wxDataObject::SetData* (p. 314) method.

**wxDropTarget::OnData**

**virtual wxDragResult OnData(wxCoord x, wxCoord y, wxDragResult def)**

Called after *OnDrop* (p. 562) returns true. By default this will usually *GetData* (p. 562) and will return the suggested default value *def*.

**wxDropTarget::OnDrop**

**virtual bool OnDrop(wxCoord x, wxCoord y)**

Called when the user drops a data object on the target. Return false to veto the operation.

**Parameters***x*

The x coordinate of the mouse.

*y*

The y coordinate of the mouse.

**Return value**

Return true to accept the data, false to veto the operation.

**`wxDropTarget::OnEnter`**

**virtual `wxDragResult` `OnEnter`(`wxCoord` *x*, `wxCoord` *y*, `wxDragResult` *def*)**

Called when the mouse enters the drop target. By default, this calls *OnDragOver* (p. 563).

**Parameters***x*

The x coordinate of the mouse.

*y*

The y coordinate of the mouse.

*def*

Suggested default for return value. Determined by SHIFT or CONTROL key states.

**Return value**

Returns the desired operation or `wxDragNone`. This is used for optical feedback from the side of the drop source, typically in form of changing the icon.

**`wxDropTarget::OnDragOver`**

**virtual `wxDragResult` `OnDragOver`(`wxCoord` *x*, `wxCoord` *y*, `wxDragResult` *def*)**

Called when the mouse is being dragged over the drop target. By default, this calls functions return the suggested return value *def*.

**Parameters***x*

The x coordinate of the mouse.

*y*

The y coordinate of the mouse.

*def*

Suggested value for return value. Determined by SHIFT or CONTROL key states.

### **Return value**

Returns the desired operation or `wxDragNone`. This is used for optical feedback from the side of the drop source, typically in form of changing the icon.

### **wxDropTarget::OnLeave**

**virtual void OnLeave()**

Called when the mouse leaves the drop target.

### **wxDropTarget::SetDataObject**

**void SetDataObject(wxDataObject\* data)**

Sets the data *wxDataObject* (p. 311) associated with the drop target and deletes any previously associated data object.

## **wxDynamicLibrary**

`wxDynamicLibrary` is a class representing dynamically loadable library (Windows DLL, shared library under Unix etc.). Just create an object of this class to load a library and don't worry about unloading it -- it will be done in the objects destructor automatically.

### **Derived from**

No base class.

### **Include files**

<wx/dynlib.h>

(only available if `wxUSE_DYNLIB_CLASS` is set to 1)

### **wxDynamicLibrary::wxDynamicLibrary**

**wxDynamicLibrary()**

**wxDynamicLibrary(const wxString& name, int flags = wxDL\_DEFAULT)**

Constructor. Second form calls *Load* (p. 566).

### **wxDynamicLibrary::CanonicalizeName**

**static wxString CanonicalizeName(const wxString& name,**

**wxDynamicLibraryCategory** *cat* = wxDL\_LIBRARY)

Returns the platform-specific full name for the library called *name*. E.g. it adds a ".dll" extension under Windows and "lib" prefix and ".so", ".sl" or maybe ".dylib" extension under Unix.

The possible values for *cat* are:

wxDL_LIBRARY	normal library
wxDL_MODULE	a loadable module or plugin

#### See also

*CanonicalizePluginName* (p. 565)

### wxDynamicLibrary::CanonicalizePluginName

**static wxString CanonicalizePluginName(const wxString& *name*, wxPluginCategory *cat* = wxDL\_PLUGIN\_GUI)**

This function does the same thing as *CanonicalizeName* (p. 564) but for wxWidgets plugins. The only difference is that compiler and version information are added to the name to ensure that the plugin which is going to be loaded will be compatible with the main program.

The possible values for *cat* are:

wxDL_PLUGIN_GUI	plugin which uses GUI classes (default)
wxDL_PLUGIN_BASE	plugin which only uses wxBase

### wxDynamicLibrary::Detach

**wxDIType Detach()**

Detaches this object from its library handle, i.e. the object will not unload the library any longer in its destructor but it is now the callers responsibility to do this using *Unload* (p. 567).

### wxDynamicLibrary::GetSymbol

**void \* GetSymbol(const wxString& *name*) const**

Returns pointer to symbol *name* in the library or NULL if the library contains no such symbol.

#### See also

*wxDYNLIB\_FUNCTION* (p. 1951)

### wxDynamicLibrary::GetSymbolAorW

**void \* GetSymbolAorW(const wxString& name) const**

This function is available only under Windows as it is only useful when dynamically loading symbols from standard Windows DLLs. Such functions have either 'A' (in ANSI build) or 'W' (in Unicode, or wide character build) suffix if they take string parameters. Using this function you can use just the base name of the function and the correct suffix is appended automatically depending on the current build. Otherwise, this method is identical to *GetSymbol* (p. 565).

**wxDynamicLibrary::GetProgramHandle**

**static wxDllType GetProgramHandle()**

Return a valid handle for the main program itself or `NULL` if symbols from the main program can't be loaded on this platform.

**wxDynamicLibrary::HasSymbol**

**bool HasSymbol(const wxString& name) const**

Returns `true` if the symbol with the given *name* is present in the dynamic library, `false` otherwise. Unlike *GetSymbol* (p. 565), this function doesn't log an error message if the symbol is not found.

This function is new since wxWidgets version 2.5.4

**wxDynamicLibrary::IsLoaded**

**bool IsLoaded() const**

Returns `true` if the library was successfully loaded, `false` otherwise.

**wxDynamicLibrary::ListLoaded**

**static wxDynamicLibraryDetailsArray ListLoaded()**

This static method returns an *array* (p. 71) containing the details of all modules loaded into the address space of the current project, the array elements are object of `wxDynamicLibraryDetails` class. The array will be empty if an error occurred.

This method is currently implemented only under Win32 and Linux and is useful mostly for diagnostics purposes.

**wxDynamicLibrary::Load**

**bool Load(const wxString& name, int flags = wxDL\_DEFAULT)**

Loads DLL with the given *name* into memory. The *flags* argument can be a combination of the following bits:

`wxDL_LAZY`

equivalent of `RTLD_LAZY` under Unix, ignored

	elsewhere
<code>wxDL_NOW</code>	equivalent of <code>RTLD_NOW</code> under Unix, ignored elsewhere
<code>wxDL_GLOBAL</code>	equivalent of <code>RTLD_GLOBAL</code> under Unix, ignored elsewhere
<code>wxDL_VERBATIM</code>	don't try to append the appropriate extension to the library name (this is done by default).
<code>wxDL_DEFAULT</code>	default flags, same as <code>wxDL_NOW</code> currently

Returns `true` if the library was successfully loaded, `false` otherwise.

### **`wxDynamicLibrary::Unload`**

**`void Unload()`**

**`static void Unload(wxDLType handle)`**

Unloads the library from memory. `wxDynamicLibrary` object automatically calls this method from its destructor if it had been successfully loaded.

The second version is only used if you need to keep the library in memory during a longer period of time than the scope of the `wxDynamicLibrary` object. In this case you may call *Detach* (p. 565) and store the handle somewhere and call this static method later to unload it.

## **`wxDynamicLibraryDetails`**

This class is used for the objects returned by `wxDynamicLibrary::ListLoaded` (p. 566) method and contains the information about a single module loaded into the address space of the current process. A module in this context may be either a dynamic library or the main program itself.

### **Derived from**

No base class.

### **Include files**

`<wx/dynlib.h>`

(only available if `wxUSE_DYNLIB_CLASS` is set to 1)

### **`wxDynamicLibraryDetails::GetName`**

**`wxString GetName() const`**

Returns the base name of this module, e.g. `kernel32.dll` or `libc-2.3.2.so`.

### **wxDynamicLibraryDetails::GetPath**

#### **wxString GetPath() const**

Returns the full path of this module if available, e.g.

`c:\windows\system32\kernel32.dll` or `/lib/libc-2.3.2.so`.

### **wxDynamicLibraryDetails::GetAddress**

#### **bool GetAddress(void \*\*addr, size\_t \*len) const**

Retrieves the load address and the size of this module.

#### **Parameters**

*addr*

the pointer to the location to return load address in, may be `NULL`

*len*

pointer to the location to return the size of this module in memory in, may be `NULL`

#### **Return value**

`true` if the load address and module size were retrieved, `false` if this information is not available.

### **wxDynamicLibraryDetails::GetVersion**

#### **wxString GetVersion() const**

Returns the version of this module, e.g. `5.2.3790.0` or `2.3.2`. The returned string is empty if the version information is not available.

## **wxEncodingConverter**

This class is capable of converting strings between two 8-bit encodings/charsets. It can also convert from/to Unicode (but only if you compiled `wxWidgets` with `wxUSE_WCHAR_T` set to 1). Only a limited subset of encodings is supported by `wxEncodingConverter`: `wxFONTENCODING_ISO8859_1..15`, `wxFONTENCODING_CP1250..1257` and `wxFONTENCODING_KOI8`.

#### **Note**

Please use `wxMBConv` classes (p. 2059) instead if possible. `wxCSSConv` (p. 296) has much better support for various encodings than `wxEncodingConverter`. `wxEncodingConverter` is useful only if you rely on `wxCONVERT_SUBSTITUTE` mode of operation (see *Init* (p. 569)).



**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/encconv.h>

**See also**

*wxFontMapper* (p. 674), *wxMBConv* (p. 1038), *Writing non-English applications* (p. 2063)

**wxEncodingConverter::wxEncodingConverter****wxEncodingConverter()**

Constructor.

**wxEncodingConverter::Init**

**bool Init(wxFontEncoding input\_enc, wxFontEncoding output\_enc, int method = wxCONVERT\_STRICT)**

Initialize conversion. Both output or input encoding may be wxFONTENCODING\_UNICODE, but only if wxUSE\_ENCODING is set to 1. All subsequent calls to *Convert()* (p. 570) will interpret its argument as a string in *input\_enc* encoding and will output string in *output\_enc* encoding. You must call this method before calling *Convert*. You may call it more than once in order to switch to another conversion. *Method* affects behaviour of *Convert()* in case input character cannot be converted because it does not exist in output encoding:

**wxCONVERT\_STRICT**

follow behaviour of GNU Recode - just copy unconvertible characters to output and don't change them (its integer value will stay the same)

**wxCONVERT\_SUBSTITUTE**

try some (lossy) substitutions - e.g. replace unconvertible latin capitals with acute by ordinary capitals, replace en-dash or em-dash by '-' etc.

Both modes guarantee that output string will have same length as input string.

**Return value**

false if given conversion is impossible, true otherwise (conversion may be impossible either if you try to convert to Unicode with non-Unicode build of wxWidgets or if input or output encoding is not supported.)

**wxEncodingConverter::CanConvert**

**static bool CanConvert(wxFontEncoding *encIn*, wxFontEncoding *encOut*)**

Return true if (any text in) multibyte encoding *encIn* can be converted to another one (*encOut*) losslessly.

Do not call this method with `wxFONTENCODING_UNICODE` as either parameter, it doesn't make sense (always works in one sense and always depends on the text to convert in the other).

### **wxEncodingConverter::Convert**

**bool Convert(const char\* *input*, char\* *output*) const**

**bool Convert(const wchar\_t\* *input*, wchar\_t\* *output*) const**

**bool Convert(const char\* *input*, wchar\_t\* *output*) const**

**bool Convert(const wchar\_t\* *input*, char\* *output*) const**

Convert input string according to settings passed to *Init* (p. 569) and writes the result to *output*.

**bool Convert(char\* *str*) const**

**bool Convert(wchar\_t\* *str*) const**

Convert input string according to settings passed to *Init* (p. 569) in-place, i.e. write the result to the same memory area.

All of the versions above return `true` if the conversion was lossless and `false` if at least one of the characters couldn't be converted and was replaced with ' ? ' in the output. Note that if `wxCONVERT_SUBSTITUTE` was passed to *Init* (p. 569), substitution is considered lossless operation.

**wxString Convert(const wxString& *input*) const**

Convert `wxString` and return new `wxString` object.

### **Notes**

You must call *Init* (p. 569) before using this method!

`wchar_t` versions of the method are not available if `wxWidgets` was compiled with `wxUSE_WCHAR_T` set to 0.

### **wxEncodingConverter::GetPlatformEquivalents**

**static wxFontEncodingArray GetPlatformEquivalents(wxFontEncoding *enc*, int *platform* = `wxPLATFORM_CURRENT`)**

Return equivalents for given font that are used under given platform. Supported platforms:

- `wxPLATFORM_UNIX`

- wxPLATFORM\_WINDOWS
- wxPLATFORM\_OS2
- wxPLATFORM\_MAC
- wxPLATFORM\_CURRENT

wxPLATFORM\_CURRENT means the platform this binary was compiled for.

Examples:

current platform	enc	returned value
-----	-----	-----
unix	CP1250	{ ISO8859_2 }
unix	ISO8859_2	{ ISO8859_2 }
windows	ISO8859_2	{ CP1250 }
unix	CP1252	{ ISO8859_1, ISO8859_15 }

Equivalence is defined in terms of convertibility: two encodings are equivalent if you can convert text between them without losing information (it may - and will - happen that you lose special chars like quotation marks or em-dashes but you shouldn't lose any diacritics and language-specific characters when converting between equivalent encodings).

Remember that this function does **NOT** check for presence of fonts in system. It only tells you what are most suitable encodings. (It usually returns only one encoding.)

#### Notes

- Note that argument *enc* itself may be present in the returned array, so that you can, as a side-effect, detect whether the encoding is native for this platform or not.
- *Convert* (p. 570) is not limited to converting between equivalent encodings, it can convert between two arbitrary encodings.
- If *enc* is present in the returned array, then it is **always** the first item of it.
- Please note that the returned array may contain no items at all.

#### wxEncodingConverter::GetAllEquivalents

**static wxFontEncodingArray GetAllEquivalents(wxFontEncoding enc)**

Similar to *GetPlatformEquivalents* (p. 570), but this one will return ALL equivalent encodings, regardless of the platform, and including itself.

This platform's encodings are before others in the array. And again, if *enc* is in the array, it is the very first item in it.

#### wxEraseEvent

An erase event is sent when a window's background needs to be repainted.

On some platforms, such as GTK+, this event is simulated (simply generated just before the paint event) and may cause flicker. It is therefore recommended that you set the text background colour explicitly in order to prevent flicker. The default background colour under GTK+ is grey.

To intercept this event, use the `EVT_ERASE_BACKGROUND` macro in an event table definition.

You must call `wxEraseEvent::GetDC` and use the returned device context if it is non-NULL. If it is NULL, create your own temporary `wxClientDC` object.

### Derived from

*wxEvent* (p. 572)

*wxObject* (p. 1148)

### Include files

<wx/event.h>

### Event table macros

To process an erase event, use this event handler macro to direct input to a member function that takes a `wxEraseEvent` argument.

**EVT\_ERASE\_BACKGROUND(func)**    Process a `wxEVT_ERASE_BACKGROUND` event.

### Remarks

Use the device context returned by *GetDC* (p. 572) to draw on, don't create a `wxPaintDC` in the event handler.

### See also

*Event handling overview* (p. 2077)

## **wxEraseEvent::wxEraseEvent**

**wxEraseEvent(int id = 0, wxDC\* dc = NULL)**

Constructor.

## **wxEraseEvent::GetDC**

**wxDC\* GetDC() const**

Returns the device context associated with the erase event to draw on.

## **wxEvent**

An event is a structure holding information about an event passed to a callback or member function. **wxEvt** used to be a multipurpose event object, and is an abstract base class for other event classes (see below).

For more information about events, see the *Event handling overview* (p. 2077).

**wxPerl note:** In wxPerl custom event classes should be derived from `Wx::PlEvent` and `Wx::PlCommandEvent`.

### Derived from

*wxObject* (p. 1148)

### Include files

<wx/event.h>

### See also

*wxCommandEvent* (p. 250), *wxMouseEvent* (p. 1119)

## wxEvt::wxEvt

**wxEvt(int id = 0, wxEventType eventType = wxEVT\_NULL)**

Constructor. Should not need to be used directly by an application.

## wxEvt::m\_propagationLevel

### int m\_propagationLevel

Indicates how many levels the event can propagate. This member is protected and should typically only be set in the constructors of the derived classes. It may be temporarily changed by *StopPropagation* (p. 576) and *ResumePropagation* (p. 575) and tested with *ShouldPropagate* (p. 575).

The initial value is set to either `wxEVT_PROPAGATE_NONE` (by default) meaning that the event shouldn't be propagated at all or to `wxEVT_PROPAGATE_MAX` (for command events) meaning that it should be propagated as much as necessary.

Any positive number means that the event should be propagated but no more than the given number of times. E.g. the propagation level may be set to 1 to propagate the event to its parent only, but not to its grandparent.

## wxEvt::Clone

**virtual wxEvt\* Clone() const**

Returns a copy of the event.

Any event that is posted to the wxWidgets event system for later action

(via `wxEvtHandler::AddPendingEvent` (p. 576) or `wxPostEvent` (p. 1960)) must implement this method. All `wxWidgets` events fully implement this method, but any derived events implemented by the user should also implement this method just in case they (or some event derived from them) are ever posted.

All `wxWidgets` events implement a copy constructor, so the easiest way of implementing the `Clone` function is to implement a copy constructor for a new event (call it `MyEvent`) and then define the `Clone` function like this:

```
wxEvent *Clone(void) const { return new MyEvent(*this); }
```

### **wxEvent::GetEventObject**

**wxObject\* GetEventObject()**

Returns the object (usually a window) associated with the event, if any.

### **wxEvent::GetEventType**

**wxEventType GetEventType()**

Returns the identifier of the given event type, such as `wxEVT_COMMAND_BUTTON_CLICKED`.

### **wxEvent::GetId**

**int GetId() const**

Returns the identifier associated with this event, such as a button command id.

### **wxEvent::GetSkipped**

**bool GetSkipped() const**

Returns true if the event handler should be skipped, false otherwise.

### **wxEvent::GetTimestamp**

**long GetTimestamp()**

Gets the timestamp for the event. The timestamp is the time in milliseconds since some fixed moment (*not* necessarily the standard Unix Epoch, so only differences between the timestamps and not their absolute values usually make sense).

### **wxEvent::IsCommandEvent**

**bool IsCommandEvent() const**

Returns true if the event is or is derived from `wxCommandEvent` (p. 250) else it returns false. Note: Exists only for optimization purposes.

**wxEvt::ResumePropagation****void ResumePropagation(int *propagationLevel*)**

Sets the propagation level to the given value (for example returned from an earlier call to *StopPropagation* (p. 576)).

**wxEvt::SetEventObject****void SetEventObject(wxObject\* *object*)**

Sets the originating object.

**wxEvt::SetEventType****void SetEventType(wxEventType *type*)**

Sets the event type.

**wxEvt::SetId****void SetId(int *id*)**

Sets the identifier associated with this event, such as a button command id.

**wxEvt::SetTimestamp****void SetTimestamp(long *timeStamp*)**

Sets the timestamp for the event.

**wxEvt::ShouldPropagate****bool ShouldPropagate() const**

Test if this event should be propagated or not, i.e. if the propagation level is currently greater than 0.

**wxEvt::Skip****void Skip(bool *skip = true*)**

This method can be used inside an event handler to control whether further event handlers bound to this event will be called after the current one returns. Without *Skip()* (or equivalently if *Skip(false)* is used), the event will not be processed any more. If *Skip(true)* is called, the event processing system continues searching for a further handler function for this event, even though it has been processed already in the current handler.

In general, it is recommended to skip all non-command events to allow the default handling to take place. The command events are, however, normally not skipped as usually a single command such as a button click or menu item selection must only be

processed by one handler.

### **wxEvtHandler::StopPropagation**

#### **int StopPropagation()**

Stop the event from propagating to its parent window.

Returns the old propagation level value which may be later passed to *ResumePropagation* (p. 575) to allow propagating the event again.

## **wxEvtHandler**

A class that can handle events from the windowing system. wxWindow (and therefore all window classes) are derived from this class.

When events are received, wxEvtHandler invokes the method listed in the event table using itself as the object. When using multiple inheritance it is imperative that the wxEvtHandler(-derived) class be the first class inherited such that the "this" pointer for the overall object will be identical to the "this" pointer for the wxEvtHandler portion.

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/event.h>

### **See also**

*Event handling overview* (p. 2077)

### **wxEvtHandler::wxEvtHandler**

#### **wxEvtHandler()**

Constructor.

### **wxEvtHandler::~~wxEvtHandler**

#### **~wxEvtHandler()**

Destructor. If the handler is part of a chain, the destructor will unlink itself and restore the previous and next handlers so that they point to each other.

### **wxEvtHandler::AddPendingEvent**

#### **void AddPendingEvent(wxEvent& event)**



This function posts an event to be processed later.

### Parameters

*event*

Event to add to process queue.

### Remarks

The difference between sending an event (using the *ProcessEvent* (p. 580) method) and posting it is that in the first case the event is processed before the function returns, while in the second case, the function returns immediately and the event will be processed sometime later (usually during the next event loop iteration).

A copy of *event* is made by the function, so the original can be deleted as soon as function returns (it is common that the original is created on the stack). This requires that the *wxEvent::Clone* (p. 573) method be implemented by *event* so that it can be duplicated and stored until it gets processed.

This is also the method to call for inter-thread communication---it will post events safely between different threads which means that this method is thread-safe by using critical sections where needed. In a multi-threaded program, you often need to inform the main GUI thread about the status of other working threads and such notification should be done using this method.

This method automatically wakes up idle handling if the underlying window system is currently idle and thus would not send any idle events. (Waking up idle handling is done calling *::wxWakeUpIdle* (p. 1912).)

### **wxEvtHandler::Connect**

**void Connect(int id, int lastId, wxEventType eventType, wxObjectEventFunction function, wxObject\* userData = NULL, wxEvtHandler\* eventSink = NULL)**

**void Connect(int id, wxEventType eventType, wxObjectEventFunction function, wxObject\* userData = NULL, wxEvtHandler\* eventSink = NULL)**

**void Connect(wxEventType eventType, wxObjectEventFunction function, wxObject\* userData = NULL, wxEvtHandler\* eventSink = NULL)**

Connects the given function dynamically with the event handler, id and event type. This is an alternative to the use of static event tables. See the 'event' or the old 'dynamic' sample for usage.

### Parameters

*id*

The identifier (or first of the identifier range) to be associated with the event handler function. For the version not taking this argument, it defaults to *wxID\_ANY*.

*lastId*

The second part of the identifier range to be associated with the event handler function.

*eventType*

The event type to be associated with this event handler.

*function*

The event handler function. Note that this function should be explicitly converted to the correct type which can be done using a macro called `wxFooHandler` for the handler for any `wxFooEvent`.

*userData*

Data to be associated with the event table entry.

*eventSink*

Object whose member function should be called. If this is `NULL`, *this* will be used.

### Example

```
frame->Connect( wxID_EXIT,
               wxEVT_COMMAND_MENU_SELECTED,
               wxCommandEventHandler(MyFrame::OnQuit) );
```

**wxPerl note:** In `wxPerl` this function takes 4 arguments: `id`, `lastid`, `type`, `method`; if `method` is `undef`, the handler is disconnected.

### **wxEvtHandler::Disconnect**

**bool Disconnect(wxEventType eventType = wxEVT\_NULL, wxObjectEventFunction function = NULL, wxObject\* userData = NULL, wxEvtHandler\* eventSink = NULL)**

**bool Disconnect(int id = wxID\_ANY, wxEventType eventType = wxEVT\_NULL, wxObjectEventFunction function = NULL, wxObject\* userData = NULL, wxEvtHandler\* eventSink = NULL)**

**bool Disconnect(int id, int lastId = wxID\_ANY, wxEventType eventType = wxEVT\_NULL, wxObjectEventFunction function = NULL, wxObject\* userData = NULL, wxEvtHandler\* eventSink = NULL)**

Disconnects the given function dynamically from the event handler, using the specified parameters as search criteria and returning true if a matching function has been found and removed. This method can only disconnect functions which have been added using the `wxEvtHandler::Connect` (p. 577) method. There is no way to disconnect functions connected using the (static) event tables.

### Parameters

*id*

The identifier (or first of the identifier range) associated with the event handler

function.

*lastId*

The second part of the identifier range associated with the event handler function.

*eventType*

The event type associated with this event handler.

*function*

The event handler function.

*userData*

Data associated with the event table entry.

*eventSink*

Object whose member function should be called.

**wxPerl note:** In wxPerl this function takes 3 arguments: *id*, *lastid*, *type*.

### **wxEvtHandler::GetClientData**

**void\* GetClientData()**

Gets user-supplied client data.

#### **Remarks**

Normally, any extra data the programmer wishes to associate with the object should be made available by deriving a new class with new data members.

#### **See also**

*wxEvtHandler::SetClientData* (p. 582)

### **wxEvtHandler::GetClientObject**

**wxClientData\* GetClientObject() const**

Get a pointer to the user-supplied client data object.

#### **See also**

*wxEvtHandler::SetClientObject* (p. 582), *wxClientData* (p. 193)

### **wxEvtHandler::GetEventHandlerEnabled**

**bool GetEventHandlerEnabled()**

Returns true if the event handler is enabled, false otherwise.

**See also**

*wxEvtHandler::SetEvtHandlerEnabled* (p. 583)

**wxEvtHandler::GetNextHandler****wxEvtHandler\* GetNextHandler()**

Gets the pointer to the next handler in the chain.

**See also**

*wxEvtHandler::SetNextHandler* (p. 583), *wxEvtHandler::GetPreviousHandler* (p. 580), *wxEvtHandler::SetPreviousHandler* (p. 583), *wxWindow::PushEventHandler* (p. 1829), *wxWindow::PopEventHandler* (p. 1828)

**wxEvtHandler::GetPreviousHandler****wxEvtHandler\* GetPreviousHandler()**

Gets the pointer to the previous handler in the chain.

**See also**

*wxEvtHandler::SetPreviousHandler* (p. 583), *wxEvtHandler::GetNextHandler* (p. 580), *wxEvtHandler::SetNextHandler* (p. 583), *wxWindow::PushEventHandler* (p. 1829), *wxWindow::PopEventHandler* (p. 1828)

**wxEvtHandler::ProcessEvent****virtual bool ProcessEvent(*wxEvent& event*)**

Processes an event, searching event tables and calling zero or more suitable event handler function(s).

**Parameters**

*event*

Event to process.

**Return value**

true if a suitable event handler function was found and executed, and the function did not call *wxEvent::Skip* (p. 575).

**Remarks**

Normally, your application would not call this function: it is called in the *wxWidgets* implementation to dispatch incoming user interface events to the framework (and application).

However, you might need to call it if implementing new functionality (such as a new

control) where you define new event types, as opposed to allowing the user to override virtual functions.

An instance where you might actually override the **ProcessEvent** function is where you want to direct event processing to event handlers not normally noticed by wxWidgets. For example, in the document/view architecture, documents and views are potential event handlers. When an event reaches a frame, **ProcessEvent** will need to be called on the associated document and view in case event handler functions are associated with these objects. The property classes library (wxProperty) also overrides **ProcessEvent** for similar reasons.

The normal order of event table searching is as follows:

1. If the object is disabled (via a call to *wxEvtHandler::SetEvtHandlerEnabled* (p. 583)) the function skips to step (6).
2. If the object is a wxWindow, **ProcessEvent** is recursively called on the window's *wxValidator* (p. 1767). If this returns true, the function exits.
3. **SearchEventTable** is called for this event handler. If this fails, the base class table is tried, and so on until no more tables exist or an appropriate function was found, in which case the function exits.
4. The search is applied down the entire chain of event handlers (usually the chain has a length of one). If this succeeds, the function exits.
5. If the object is a wxWindow and the event is a wxCommandEvent, **ProcessEvent** is recursively applied to the parent window's event handler. If this returns true, the function exits.
6. Finally, **ProcessEvent** is called on the wxApp object.

#### See also

*wxEvtHandler::SearchEventTable* (p. 581)

#### wxEvtHandler::SearchEventTable

**virtual bool SearchEventTable**(wxEventTable& *table*, wxEvent& *event*)

Searches the event table, executing an event handler function if an appropriate one is found.

#### Parameters

*table*

Event table to be searched.

*event*

Event to be matched against an event table entry.

#### Return value

true if a suitable event handler function was found and executed, and the function did not call *wxEvtHandler::Skip* (p. 575).

### Remarks

This function looks through the object's event table and tries to find an entry that will match the event.

An entry will match if:

1. The event type matches, and
2. the identifier or identifier range matches, or the event table entry's identifier is zero.

If a suitable function is called but calls *wxEvtHandler::Skip* (p. 575), this function will fail, and searching will continue.

### See also

*wxEvtHandler::ProcessEvent* (p. 580)

### **wxEvtHandler::SetClientData**

**void SetClientData(void\* data)**

Sets user-supplied client data.

### Parameters

*data*

Data to be associated with the event handler.

### Remarks

Normally, any extra data the programmer wishes to associate with the object should be made available by deriving a new class with new data members. You must not call this method and *SetClientObject* (p. 582) on the same class - only one of them.

### See also

*wxEvtHandler::GetClientData* (p. 579)

### **wxEvtHandler::SetClientObject**

**void SetClientObject(wxClientData\* data)**

Set the client data object. Any previous object will be deleted.

### See also

*wxEvtHandler::GetClientObject* (p. 579), *wxClientData* (p. 193)

**wxEvtHandler::SetEvtHandlerEnabled****void SetEvtHandlerEnabled**(bool *enabled*)

Enables or disables the event handler.

**Parameters**

*enabled*

true if the event handler is to be enabled, false if it is to be disabled.

**Remarks**

You can use this function to avoid having to remove the event handler from the chain, for example when implementing a dialog editor and changing from edit to test mode.

**See also**

*wxEvtHandler::GetEvtHandlerEnabled* (p. 579)

**wxEvtHandler::SetNextHandler****void SetNextHandler**(wxEvtHandler\* *handler*)

Sets the pointer to the next handler.

**Parameters**

*handler*

Event handler to be set as the next handler.

**See also**

*wxEvtHandler::GetNextHandler* (p. 580), *wxEvtHandler::SetPreviousHandler* (p. 583), *wxEvtHandler::GetPreviousHandler* (p. 580), *wxWindow::PushEventHandler* (p. 1829), *wxWindow::PopEventHandler* (p. 1828)

**wxEvtHandler::SetPreviousHandler****void SetPreviousHandler**(wxEvtHandler\* *handler*)

Sets the pointer to the previous handler.

**Parameters**

*handler*

Event handler to be set as the previous handler.

**See also**

*wxEvtHandler::GetPreviousHandler* (p. 580), *wxEvtHandler::SetNextHandler* (p. 583),

*wxEvtHandler::GetNextHandler* (p. 580), *wxWindow::PushEventHandler* (p. 1829),  
*wxWindow::PopEventHandler* (p. 1828)

## wxFile

wxFile implements buffered file I/O. This is a very small class designed to minimize the overhead of using it - in fact, there is hardly any overhead at all, but using it brings you automatic error checking and hides differences between platforms and compilers. It wraps inside it a `FILE *` handle used by standard C IO library (also known as `stdio`).

### Derived from

None.

### Include files

<wx/file.h>

<b>wxFromStart</b>	Count offset from the start of the file
<b>wxFromCurrent</b>	Count offset from the current position of the file pointer
<b>wxFromEnd</b>	Count offset from the end of the file (backwards)

### wxFile::wxFile

**wxFile()**

Default constructor.

**wxFile(const char\* filename, const char\* mode = "r")**

Opens a file with the given mode. As there is no way to return whether the operation was successful or not from the constructor you should test the return value of *IsOpened* (p. 586) to check that it didn't fail.

**wxFile(FILE\* fp)**

Opens a file with the given file pointer, which has already been opened.

### Parameters

*filename*

The filename.

*mode*

The mode in which to open the file using standard C strings. Note that you should use "b" flag if you use binary files under Windows or the results might be



unexpected due to automatic newline conversion done for the text files.

*fp*

An existing file descriptor, such as `stderr`.

### **wxFile::~~wxFile**

**~wxFile()**

Destructor will close the file.

NB: it is not virtual so you should *not* derive from `wxFile`!

### **wxFile::Attach**

**void Attach(FILE\* fp)**

Attaches an existing file pointer to the `wxFile` object.

The descriptor should be already opened and it will be closed by `wxFile` object.

### **wxFile::Close**

**bool Close()**

Closes the file and returns `true` on success.

### **wxFile::Detach**

**void Detach()**

Get back a file pointer from `wxFile` object -- the caller is responsible for closing the file if this descriptor is opened. *IsOpened()* (p. 586) will return `false` after call to `Detach()`.

### **wxFile::fp**

**FILE \* fp() const**

Returns the file pointer associated with the file.

### **wxFile::Eof**

**bool Eof() const**

Returns `true` if the an attempt has been made to read *past* the end of the file.

Note that the behaviour of the file descriptor based class `wxFile` (p. 591) is different as `wxFile::Eof` (p. 594) will return `true` here as soon as the last byte of the file has been read.

Also note that this method may only be called for opened files and may crash if the file is

not opened.

**See also**

*IsOpened* (p. 586)

**wxFile::Error**

Returns `true` if an error has occurred on this file, similar to the standard `feof()` function.

Please note that this method may only be called for opened files and may crash if the file is not opened.

**See also**

*IsOpened* (p. 586)

**wxFile::Flush****bool Flush()**

Flushes the file and returns `true` on success.

**wxFile::GetKind****wxFileKind GetKind() const**

Returns the type of the file. Possible return values are:

```
enum wxFileKind
{
    wxFILE_KIND_UNKNOWN,
    wxFILE_KIND_DISK,      // a file supporting seeking to arbitrary
offsets
    wxFILE_KIND_TERMINAL, // a tty
    wxFILE_KIND_PIPE      // a pipe
};
```

**wxFile::IsOpened****bool IsOpened() const**

Returns `true` if the file is opened. Most of the methods of this class may only be used for an opened file.

**wxFile::Length****wxFileOffset Length() const**

Returns the length of the file.

**wxFile::Open****bool Open(const char\* filename, const char\* mode = "r")**

Opens the file, returning `true` if successful.

**Parameters***filename*

The filename.

*mode*

The mode in which to open the file.

**wxFile::Read****size\_t Read(void\* buffer, size\_t count)**

Reads the specified number of bytes into a buffer, returning the actual number read.

**Parameters***buffer*

A buffer to receive the data.

*count*

The number of bytes to read.

**Return value**

The number of bytes read.

**wxFile::ReadAll****bool ReadAll(wxString \* str, wxMBConv& conv = wxConvUTF8)**

Reads the entire contents of the file into a string.

**Parameters***str*

String to read data into.

*conv*

Conversion object to use in Unicode build; by default supposes that file contents is encoded in UTF-8.

**Return value**

`true` if file was read successfully, `false` otherwise.

### **wxFile::Seek**

**bool Seek(wxFileOffset ofs, wxSeekMode mode = wxFromStart)**

Seeks to the specified position and returns `true` on success.

#### **Parameters**

*ofs*

Offset to seek to.

*mode*

One of **wxFromStart**, **wxFromEnd**, **wxFromCurrent**.

### **wxFile::SeekEnd**

**bool SeekEnd(wxFileOffset ofs = 0)**

Moves the file pointer to the specified number of bytes before the end of the file and returns `true` on success.

#### **Parameters**

*ofs*

Number of bytes before the end of the file.

### **wxFile::Tell**

**wxFileOffset Tell() const**

Returns the current position.

### **wxFile::Write**

**size\_t Write(const void\* buffer, size\_t count)**

Writes the specified number of bytes from a buffer.

#### **Parameters**

*buffer*

A buffer containing the data.

*count*

The number of bytes to write.

**Return value**

Number of bytes written.

**wxFile::Write**

**bool Write(const wxString& s, wxMBConv& conv = wxConvUTF8)**

Writes the contents of the string to the file, returns `true` on success.

The second argument is only meaningful in Unicode build of wxWidgets when `conv` is used to convert `s` to multibyte representation.

**wxFileInputStream**

This class represents data read in from a file. There are actually two such groups of classes: this one is based on *wxFile* (p. 584) whereas *wxFileInputStream* (p. 608) is based in the *wxFile* (p. 591) class.

Note that *SeekI()* (p. 943) can seek beyond the end of the stream (file) and will thus not return *wxInvalidOffset* for that.

**Derived from**

*wxInputStream* (p. 941)

**Include files**

<wx/wfstream.h>

**See also**

*wxBufferedInputStream* (p. 161), *wxFileOutputStream* (p. 590), *wxFileOutputStream* (p. 628)

**wxFileInputStream::wxFileInputStream**

**wxFileInputStream(const wxString& filename, const wxChar \* mode = "rb")**

Opens the specified file using its *filename* name using the specified mode.

**wxFileInputStream(wxFile& file)**

Initializes a file stream in read-only mode using the file I/O object *file*.

**wxFileInputStream(FILE \* fp)**

Initializes a file stream in read-only mode using the specified file pointer *fp*.

**wxFileInputStream::~~wxFileInputStream**

**~wxFFileInputStream()**

Destructor.

**wxFFileInputStream::IsOk**

**bool IsOk() const**

Returns true if the stream is initialized and ready.

## wxFFileOutputStream

This class represents data written to a file. There are actually two such groups of classes: this one is based on *wxFFile* (p. 584) whereas *wxFileInputStream* (p. 589) is based in the *wxFile* (p. 591) class.

Note that *SeekO()* (p. 1157) can seek beyond the end of the stream (file) and will thus not return *wxInvalidOffset* for that.

### Derived from

*wxOutputStream* (p. 1156)

### Include files

<wx/wfstream.h>

### See also

*wxBufferedOutputStream* (p. 161), *wxFFileInputStream* (p. 589), *wxFileInputStream* (p. 608)

### wxFFileOutputStream::wxFFileOutputStream

**wxFFileOutputStream(const wxString& filename, const wxChar \* mode="w+b")**

Opens the file with the given *filename* name in the specified mode.

**wxFFileOutputStream(wxFFile& file)**

Initializes a file stream in write-only mode using the file I/O object *file*.

**wxFFileOutputStream(FILE \* fp)**

Initializes a file stream in write-only mode using the file descriptor *fp*.

### wxFFileOutputStream::~~wxFFileOutputStream

**~wxFFileOutputStream()**

Destructor.

### **wxFFileOutputStream::IsOk**

**bool IsOk() const**

Returns true if the stream is initialized and ready.

## **wxFFileStream**

**Derived from**

*wxFFileOutputStream* (p. 590), *wxFFileInputStream* (p. 589)

**Include files**

<wx/wfstream.h>

**See also**

*wxStreamBuffer* (p. 1547)

### **wxFFileStream::wxFFileStream**

**wxFFileStream(const wxString& *iofileName*)**

Initializes a new file stream in read-write mode using the specified *iofilename* name.

## **wxFile**

A `wxFile` performs raw file I/O. This is a very small class designed to minimize the overhead of using it - in fact, there is hardly any overhead at all, but using it brings you automatic error checking and hides differences between platforms and compilers. `wxFile` also automatically closes the file in its destructor making it unnecessary to worry about forgetting to do it. `wxFile` is a wrapper around `file_descriptor`. - see also *wxFFile* (p. 584) for a wrapper around `FILE` structure.

`wxFileOffset` is used by the `wxFile` functions which require offsets as parameter or return them. If the platform supports it, `wxFileOffset` is a typedef for a native 64 bit integer, otherwise a 32 bit integer is used for `wxFileOffset`.

**Derived from**

None.

**Include files**

<wx/file.h>

## Constants

wx/file.h defines the following constants:

```
#define wxS_IRUSR 00400
#define wxS_IWUSR 00200
#define wxS_IXUSR 00100

#define wxS_IRGRP 00040
#define wxS_IWGRP 00020
#define wxS_IXGRP 00010

#define wxS_IROTH 00004
#define wxS_IWOTH 00002
#define wxS_IXOTH 00001

// default mode for the new files: corresponds to umask 022
#define wxS_DEFAULT (wxS_IRUSR | wxS_IWUSR | wxS_IRGRP | wxS_IWGRP
| wxS_IROTH | wxS_IWOTH)
```

These constants define the file access rights and are used with *wxFile::Create* (p. 594) and *wxFile::Open* (p. 595).

The *OpenMode* enumeration defines the different modes for opening a file, it is defined inside *wxFile* class so its members should be specified with *wxFile::* scope resolution prefix. It is also used with *wxFile::Access* (p. 593) function.

<b>wxFile::read</b>	Open file for reading or test if it can be opened for reading with <i>Access()</i>
<b>wxFile::write</b>	Open file for writing deleting the contents of the file if it already exists or test if it can be opened for writing with <i>Access()</i>
<b>wxFile::read_write</b>	Open file for reading and writing; can not be used with <i>Access()</i>
<b>wxFile::write_append</b>	Open file for appending: the file is opened for writing, but the old contents of the file is not erased and the file pointer is initially placed at the end of the file; can not be used with <i>Access()</i> . This is the same as <b>wxFile::write</b> if the file doesn't exist.
<b>wxFile::write_excl</b>	Open the file securely for writing (Uses <i>O_EXCL</i>   <i>O_CREAT</i> ). Will fail if the file already exists, else create and open it atomically. Useful for opening temporary files without being vulnerable to race exploits.

Other constants defined elsewhere but used by *wxFile* functions are *wxInvalidOffset* which represents an invalid value of type *wxFileOffset* and is returned by functions returning *wxFileOffset* on error and the seek mode constants used with *Seek()* (p. 596):



<b>wxFromStart</b>	Count offset from the start of the file
<b>wxFromCurrent</b>	Count offset from the current position of the file pointer
<b>wxFromEnd</b>	Count offset from the end of the file (backwards)

## **wxFile::wxFile**

### **wxFile()**

Default constructor.

**wxFile(const char\* filename, wxFile::OpenMode mode = wxFile::read)**

Opens a file with the given mode. As there is no way to return whether the operation was successful or not from the constructor you should test the return value of *IsOpened* (p. 595) to check that it didn't fail.

### **wxFile(int fd)**

Associates the file with the given file descriptor, which has already been opened.

## **Parameters**

*filename*

The filename.

*mode*

The mode in which to open the file. May be one of **wxFile::read**, **wxFile::write** and **wxFile::read\_write**.

*fd*

An existing file descriptor (see *Attach()* (p. 594) for the list of predefined descriptors)

## **wxFile::~~wxFile**

### **~wxFile()**

Destructor will close the file.

**NB:** it is not virtual so you should not use **wxFile** polymorphically.

## **wxFile::Access**

**static bool Access(const char \* name, OpenMode mode)**

This function verifies if we may access the given file in specified mode. Only values of **wxFile::read** or **wxFile::write** really make sense here.

**wxFile::Attach****void Attach(int fd)**

Attaches an existing file descriptor to the wxFile object. Example of predefined file descriptors are 0, 1 and 2 which correspond to stdin, stdout and stderr (and have symbolic names of **wxFile::fd\_stdin**, **wxFile::fd\_stdout** and **wxFile::fd\_stderr**).

The descriptor should be already opened and it will be closed by wxFile object.

**wxFile::Close****void Close()**

Closes the file.

**wxFile::Create****bool Create(const char\* filename, bool overwrite = false, int access = wxS\_DEFAULT)**

Creates a file for writing. If the file already exists, setting **overwrite** to true will ensure it is overwritten.

**wxFile::Detach****void Detach()**

Get back a file descriptor from wxFile object - the caller is responsible for closing the file if this descriptor is opened. *IsOpened()* (p. 595) will return false after call to Detach().

**wxFile::fd****int fd() const**

Returns the file descriptor associated with the file.

**wxFile::Eof****bool Eof() const**

Returns true if the end of the file has been reached.

Note that the behaviour of the file pointer based class *wxFile* (p. 584) is different as *wxFile::Eof* (p. 585) will return true here only if an attempt has been made to read *past* the last byte of the file, while *wxFile::Eof()* will return true even before such attempt is made if the file pointer is at the last position in the file.

Note also that this function doesn't work on unseekable file descriptors (examples include pipes, terminals and sockets under Unix) and an attempt to use it will result in an error message in such case. So, to read the entire file into memory, you should write a loop which uses *Read* (p. 596) repeatedly and tests its return condition instead of using *Eof()* as

this will not work for special files under Unix.

### **wxFile::Exists**

**static bool Exists(const char\* filename)**

Returns true if the given name specifies an existing regular file (not a directory or a link)

### **wxFile::Flush**

**bool Flush()**

Flushes the file descriptor.

Note that wxFile::Flush is not implemented on some Windows compilers due to a missing fsync function, which reduces the usefulness of this function (it can still be called but it will do nothing on unsupported compilers).

### **wxFile::GetKind**

**wxFileKind GetKind() const**

Returns the type of the file. Possible return values are:

```
enum wxFileKind
{
    wxFILE_KIND_UNKNOWN,
    wxFILE_KIND_DISK,      // a file supporting seeking to arbitrary
offsets
    wxFILE_KIND_TERMINAL, // a tty
    wxFILE_KIND_PIPE      // a pipe
};
```

### **wxFile::IsOpened**

**bool IsOpened() const**

Returns true if the file has been opened.

### **wxFile::Length**

**wxFileOffset Length() const**

Returns the length of the file.

### **wxFile::Open**

**bool Open(const char\* filename, wxFile::OpenMode mode = wxFile::read)**

Opens the file, returning true if successful.

**Parameters***filename*

The filename.

*mode*

The mode in which to open the file. May be one of **wxFile::read**, **wxFile::write** and **wxFile::read\_write**.

**wxFile::Read****size\_t Read(void\* *buffer*, size\_t *count*)**

Reads the specified number of bytes into a buffer, returning the actual number read.

**Parameters***buffer*

A buffer to receive the data.

*count*

The number of bytes to read.

**Return value**

The number of bytes read, or the symbol **wxInvalidOffset** (-1) if there was an error.

**wxFile::Seek****wxFileOffset Seek(wxFileOffset *ofs*, wxSeekMode *mode* = wxFromStart)**

Seeks to the specified position.

**Parameters***ofs*

Offset to seek to.

*mode*

One of **wxFromStart**, **wxFromEnd**, **wxFromCurrent**.

**Return value**

The actual offset position achieved, or **wxInvalidOffset** on failure.

**wxFile::SeekEnd****wxFileOffset SeekEnd(wxFileOffset *ofs* = 0)**

Moves the file pointer to the specified number of bytes relative to the end of the file. For example, `SeekEnd(-5)` would position the pointer 5bytes before the end.

**Parameters**

*ofs*

Number of bytes before the end of the file.

**Return value**

The actual offset position achieved, or `wxInvalidOffset` on failure.

**wxFile::Tell****wxFileOffset Tell() const**

Returns the current position or `wxInvalidOffset` if file is not opened or if another error occurred.

**wxFile::Write****size\_t Write(const void\* buffer, size\_t count)**

Writes the specified number of bytes from a buffer.

**Parameters**

*buffer*

A buffer containing the data.

*count*

The number of bytes to write.

**Return value**

the number of bytes actually written

**wxFile::Write****bool Write(const wxString& s, wxMBConv& conv = wxConvUTF8)**

Writes the contents of the string to the file, returns true on success.

The second argument is only meaningful in Unicode build of wxWidgets when *conv* is used to convert *s* to multibyte representation.

Note that this method only works with NUL-terminated strings, if you want to write data with embedded NULs to the file you should use the other *Write() overload* (p. 597).

## wxFileConfig

wxFileConfig implements *wxConfigBase* (p. 262) interface for storing and retrieving configuration information using plain text files. The files have a simple format reminiscent of Windows INI files with lines of the form `key = value` defining the keys and lines of special form `[group]` indicating the start of each group.

This class is used by default for wxConfig on Unix platforms but may also be used explicitly if you want to use files and not the registry even under Windows.

### Derived from

*wxConfigBase* (p. 262)

### Include files

<wx/fileconf.h>

## wxFileConfig::wxFileConfig

**wxFileConfig**(wxInputStream& *is*, wxMBConv& *conv* = wxConvUTF8)

Read the config data from the specified stream instead of the associated file, as usual.

### See also

*Save* (p. 598)

## wxFileConfig::Save

**bool** **Save**(wxOutputStream& *os*, wxMBConv& *conv* = wxConvUTF8)

Saves all config data to the given stream, returns `true` if data was saved successfully or `false` on error.

Note the interaction of this function with the internal "dirty flag": the data is saved unconditionally, i.e. even if the object is not dirty. However after saving it successfully, the dirty flag is reset so no changes will be written back to the file this object is associated with until you change its contents again.

### See also

*Flush* (p. 271)

## wxFileConfig::SetUmask

**void** **SetUmask**(int *mode*)

Allows to set the mode to be used for the config file creation. For example, to create a config file which is not readable by other users (useful if it stores some sensitive

information, such as passwords), you could use `SetUmask(0077)`.

This function doesn't do anything on non-Unix platforms.

#### See also

`wxCHANGE_UMASK` (p. 1922)

## wxFileDataObject

`wxFileDataObject` is a specialization of `wxDataObject` (p. 311) for file names. The program works with it just as if it were a list of absolute file names, but internally it uses the same format as Explorer and other compatible programs under Windows or GNOME/KDE filemanager under Unix which makes it possible to receive files from them using this class.

**Warning:** Under all non-Windows platforms this class is currently "input-only", i.e. you can receive the files from another application, but copying (or dragging) file(s) from a `wxWidgets` application is not currently supported. PS: GTK2 should work as well.

#### Virtual functions to override

None.

#### Derived from

`wxDataObjectSimple` (p. 335)

`wxDataObject` (p. 311)

#### Include files

<wx/dataobj.h>

#### See also

`wxDataObject` (p. 311), `wxDataObjectSimple` (p. 335), `wxTextDataObject` (p. 1653), `wxBitmapDataObject` (p. 145), `wxDataObject` (p. 311)

## wxFileDataObject

### wxFileDataObject()

Constructor.

### wxFileDataObject::AddFile

**virtual void AddFile(const wxString& file)**

**MSW only:** adds a file to the file list represented by this data object.

### wxFileDataObject::GetFileNames

**const wxArrayString& GetFileNames() const**

Returns the *array* (p. 83) of file names.

## wxFileDialog

This class represents the file chooser dialog.

### Derived from

*wxDialog* (p. 496)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/filedlg.h>

### Window styles

<b>wxFD_DEFAULT_STYLE</b>	Equivalent to <b>wxFD_OPEN</b> .
<b>wxFD_OPEN</b>	This is an open dialog; usually this means that the default button's label of the dialog is "Open". Cannot be combined with <b>wxFD_SAVE</b> .
<b>wxFD_SAVE</b>	This is a save dialog; usually this means that the default button's label of the dialog is "Save". Cannot be combined with <b>wxFD_OPEN</b> .
<b>wxFD_OVERWRITE_PROMPT</b>	For save dialog only: prompt for a confirmation if a file will be overwritten.
<b>wxFD_FILE_MUST_EXIST</b>	For open dialog only: the user may only select files that actually exist.
<b>wxFD_MULTIPLE</b>	For open dialog only: allows selecting multiple files.
<b>wxFD_CHANGE_DIR</b>	Change the current working directory to the directory where the file(s) chosen by the user are.
<b>wxFD_PREVIEW</b>	Show the preview of the selected files (currently only supported by wxGTK using GTK+ 2.4 or later).

**NB:** Previous versions of wxWidgets used **wxFD\_CHANGE\_DIR** by default under MS Windows which allowed the program to simply remember the last directory where user selected the files to open/save. This (desired) functionality must be implemented in the program itself now (manually remember the last path used and pass it to the dialog the next time it is called) or by using this flag.



**See also**

*wxFileDialog* overview (p. 2130), *wxFileSelector* (p. 1936)

**Remarks**

Pops up a file selector box. In Windows and GTK2.4+, this is the common file selector dialog. In X, this is a file selector box with somewhat less functionality. The path and filename are distinct elements of a full file pathname. If path is "", the current directory will be used. If filename is "", no default filename will be supplied. The wildcard determines what files are displayed in the file selector, and file extension supplies a type extension for the required filename.

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. The wildcard may be a specification for multiple types of file with a description for each, such as:

```
"BMP and GIF files (*.bmp;*.gif)|*.bmp;*.gif|PNG files  
(*.png)|*.png"
```

It must be noted that wildcard support in the native Motif file dialog is quite limited: only one alternative is supported, and it is displayed without the descriptive test; "BMP files (\*.bmp)|\*.bmp" is displayed as "\*.bmp", and both "BMP files (\*.bmp)|\*.bmp|GIF files (\*.gif)|\*.gif" and "Image files|\*.bmp;\*.gif" are errors.

**wxFileDialog::wxFileDialog**

```
wxFileDialog(wxWindow* parent, const wxString& message = "Choose a file", const  
wxString& defaultDir = "", const wxString& defaultFile = "", const wxString& wildcard =  
"*. *", long style = wxFD_DEFAULT_STYLE, const wxPoint& pos = wxDefaultPosition,  
const wxSize& sz = wxDefaultSize, const wxString& name = "filedlg")
```

Constructor. Use *wxFileDialog::ShowModal* (p. 604) to show the dialog.

**Parameters**

*parent*

Parent window.

*message*

Message to show on the dialog.

*defaultDir*

The default directory, or the empty string.

*defaultFile*

The default filename, or the empty string.

*wildcard*

A wildcard, such as `"*.*"` or `"BMP files (*.bmp)|*.bmp|GIF files (*.gif)|*.gif"`.

Note that the native Motif dialog has some limitations with respect to wildcards; see the Remarks section above.

*style*

A dialog style. See `wxFD_*` styles for more info.

*pos*

Dialog position. Not implemented.

*size*

Dialog size. Not implemented.

*name*

Dialog name. Not implemented.

**wxFileDialog::~wxFileDialog****~wxFileDialog()**

Destructor.

**wxFileDialog::GetDirectory****wxString GetDirectory() const**

Returns the default directory.

**wxFileDialog::GetFilename****wxString GetFilename() const**

Returns the default filename.

**wxFileDialog::GetFileNames****void GetFileNames(wxArrayString& *filenames*) const**

Fills the array *filenames* with the names of the files chosen. This function should only be used with the dialogs which have `wxFD_MULTIPLE` style, use *GetFilename* (p. 602) for the others.

Note that under Windows, if the user selects shortcuts, the filenames include paths, since the application cannot determine the full path of each referenced file by appending the directory containing the shortcuts to the filename.

**wxFileDialog::GetFilterIndex****int GetFilterIndex() const**

Returns the index into the list of filters supplied, optionally, in the wildcard parameter. Before the dialog is shown, this is the index which will be used when the dialog is first displayed. After the dialog is shown, this is the index selected by the user.

**wxFileDialog::GetMessage****wxString GetMessage() const**

Returns the message that will be displayed on the dialog.

**wxFileDialog::GetPath****wxString GetPath() const**

Returns the full path (directory and filename) of the selected file.

**wxFileDialog::GetPaths****void GetPaths(wxArrayString& *paths*) const**

Fills the array *paths* with the full paths of the files chosen. This function should only be used with the dialogs which have `wxFD_MULTIPLE` style, use *GetPath* (p. 603) for the others.

**wxFileDialog::GetWildcard****wxString GetWildcard() const**

Returns the file dialog wildcard.

**wxFileDialog::SetDirectory****void SetDirectory(const wxString& *directory*)**

Sets the default directory.

**wxFileDialog::SetFilename****void SetFilename(const wxString& *setfilename*)**

Sets the default filename.

**wxFileDialog::SetFilterIndex****void SetFilterIndex(int *filterIndex*)**

Sets the default filter index, starting from zero.

### **wxFileDialog::SetMessage**

**void SetMessage(const wxString& message)**

Sets the message that will be displayed on the dialog.

### **wxFileDialog::SetPath**

**void SetPath(const wxString& path)**

Sets the path (the combined directory and filename that will be returned when the dialog is dismissed).

### **wxFileDialog::SetWildcard**

**void SetWildcard(const wxString& wildCard)**

Sets the wildcard, which can contain multiple file types, for example:

"BMP files (\*.bmp)|\*.bmp|GIF files (\*.gif)|\*.gif"

Note that the native Motif dialog has some limitations with respect to wildcards; see the Remarks section above.

### **wxFileDialog::ShowModal**

**int ShowModal()**

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

## **wxFileDropTarget**

This is a *drop target* (p. 561) which accepts files (dragged from File Manager or Explorer).

### **Derived from**

*wxDropTarget* (p. 561)

### **Include files**

<wx/dnd.h>

### **See also**

*Drag and drop overview* (p. 2150), *wxDropSource* (p. 558), *wxDropTarget* (p. 561), *wxTextDropTarget* (p. 1654)

**wxFileDropTarget::wxFileDropTarget****wxFileDropTarget()**

Constructor.

**wxFileDropTarget::OnDrop**

**virtual bool OnDrop(long x, long y, const void \*data, size\_t size)**

See *wxDropTarget::OnDrop* (p. 562). This function is implemented appropriately for files, and calls *wxFileDropTarget::OnDropFiles* (p. 605).

**wxFileDropTarget::OnDropFiles**

**virtual bool OnDropFiles(wxCoord x, wxCoord y, const wxString& filenames)**

Override this function to receive dropped files.

**Parameters**

*x*

The x coordinate of the mouse.

*y*

The y coordinate of the mouse.

*filenames*

An array of filenames.

**Return value**

Return true to accept the data, false to veto the operation.

**wxFileHistory**

The *wxFileHistory* encapsulates a user interface convenience, the list of most recently visited files as shown on a menu (usually the File menu).

*wxFileHistory* can manage one or more file menus. More than one menu may be required in an MDI application, where the file history should appear on each MDI child menu as well as the MDI parent frame.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/docview.h>

**See also**

*wxFileHistory* overview (p. 2136), *wxDocManager* (p. 527)

**wxFileHistory::m\_fileHistory**

**char\*\* m\_fileHistory**

A character array of strings corresponding to the most recently opened files.

**wxFileHistory::m\_fileHistoryN**

**size\_t m\_fileHistoryN**

The number of files stored in the history array.

**wxFileHistory::m\_fileMaxFiles**

**size\_t m\_fileMaxFiles**

The maximum number of files to be stored and displayed on the menu.

**wxFileHistory::m\_fileMenu**

**wxMenu\* m\_fileMenu**

The file menu used to display the file history list (if enabled).

**wxFileHistory::wxFileHistory**

**wxFileHistory(size\_t maxFiles = 9, wxWindowID idBase = wxID\_FILE1)**

Constructor. Pass the maximum number of files that should be stored and displayed.

*idBase* defaults to `wxID_FILE1` and represents the id given to the first history menu item. Since menu items can't share the same ID you should change *idBase* (To one of your own defined IDs) when using more than one `wxFileHistory` in your application.

**wxFileHistory::~wxFileHistory**

**~wxFileHistory()**

Destructor.

**wxFileHistory::AddFileToHistory**

**void AddFileToHistory(const wxString& filename)**

Adds a file to the file history list, if the object has a pointer to an appropriate file menu.

**wxFileHistory::AddFilesToMenu****void AddFilesToMenu()**

Appends the files in the history list, to all menus managed by the file history object.

**void AddFilesToMenu(wxMenu\* menu)**

Appends the files in the history list, to the given menu only.

**wxFileHistory::GetBaseId****wxWindowID GetBaseId() const**

Returns the base identifier for the range used for appending items.

**wxFileHistory::GetCount****size\_t GetCount() const**

Returns the number of files currently stored in the file history.

**wxFileHistory::GetHistoryFile****wxString GetHistoryFile(size\_t index) const**

Returns the file at this index (zero-based).

**wxFileHistory::GetMaxFiles****int GetMaxFiles() const**

Returns the maximum number of files that can be stored.

**wxFileHistory::GetMenus****const wxList& GetMenus() const**

Returns the list of menus that are managed by this file history object.

**See also**

*wxFileHistory::UseMenu* (p. 608)

**wxFileHistory::Load****void Load(wxConfigBase& config)**

Loads the file history from the given config object. This function should be called explicitly by the application.

**See also**

*wxConfig* (p. 262)

**wxFileHistory::RemoveFileFromHistory**

**void RemoveFileFromHistory(size\_t i)**

Removes the specified file from the history.

**wxFileHistory::RemoveMenu**

**void RemoveMenu(wxMenu\* menu)**

Removes this menu from the list of those managed by this object.

**wxFileHistory::Save**

**void Save(wxConfigBase& config)**

Saves the file history into the given config object. This must be called explicitly by the application.

**See also**

*wxConfig* (p. 262)

**wxFileHistory::SetBaseId**

**void SetBaseId(wxWindowID baseId)**

Sets the base identifier for the range used for appending items.

**wxFileHistory::UseMenu**

**void UseMenu(wxMenu\* menu)**

Adds this menu to the list of those menus that are managed by this file history object. Also see *AddFilesToMenu()* (p. 607) for initializing the menu with filenames that are already in the history when this function is called, as this is not done automatically.

**wxFileInputStream**

This class represents data read in from a file. There are actually two such groups of classes: this one is based on *wxFile* (p. 591) whereas *wxFFileInputStream* (p. 589) is based in the *wxFFile* (p. 584) class.

Note that *SeekI()* (p. 943) can seek beyond the end of the stream (file) and will thus not return *wxInvalidOffset* for that.



**Derived from**

*wxInputStream* (p. 941)

**Include files**

<wx/wfstream.h>

**See also**

*wxBufferedInputStream* (p. 161), *wxFileOutputStream* (p. 628), *wxFFileOutputStream* (p. 590)

**wxFileInputStream::wxFileInputStream**

**wxFileInputStream(const wxString& *ifileName*)**

Opens the specified file using its *ifilename* name in read-only mode.

**wxFileInputStream(wxFile& *file*)**

Initializes a file stream in read-only mode using the file I/O object *file*.

**wxFileInputStream(int *fd*)**

Initializes a file stream in read-only mode using the specified file descriptor.

**wxFileInputStream::~~wxFileInputStream**

**~wxFileInputStream()**

Destructor.

**wxFileInputStream::IsOk**

**bool IsOk() const**

Returns true if the stream is initialized and ready.

**wxFileName**

*wxFileName* encapsulates a file name. This class serves two purposes: first, it provides the functions to split the file names into components and to recombine these components in the full file name which can then be passed to the OS file functions (and *wxWidgets functions* (p. 1919) wrapping them). Second, it includes the functions for working with the files itself. Note that to change the file data you should use *wxFile* (p. 591) class instead. *wxFileName* provides functions for working with the file attributes.

When working with directory names (i.e. without filename and extension) make sure not to misuse the file name part of this class with the last directory. Instead initialize the

`wxFileName` instance like this:

```
wxFileName dirname( wxT("C:\\mydir"), wxEmptyString );
MyMethod( dirname.GetPath() );
```

The same can be done using the static method `wxFileName::DirName` (p. 615):

```
wxFileName dirname = wxFileName::DirName( wxT("C:\\mydir") );
MyMethod( dirname.GetPath() );
```

Accordingly, methods dealing with directories or directory names like *IsDirReadable* (p. 621) use *GetPath* (p. 618) whereas methods dealing with file names like *IsFileReadable* (p. 622) use *GetFullPath* (p. 617).

If it is not known whether a string contains a directory name or a complete file name (such as when interpreting user input) you need to use the static function `wxFileName::DirExists` (p. 615) (or its identical variants `wxDir::Exists` (p. 511) and `wxDirExists` (p. 1922)) and construct the `wxFileName` instance accordingly. This will only work if the directory actually exists, of course:

```
wxString user_input;
// get input from user

wxFileName fname;
if (wxDirExists(user_input))
    fname.AssignDir( user_input );
else
    fname.Assign( user_input );
```

## Derived from

No base class

## Include files

<wx/filename.h>

## Data structures

Many `wxFileName` methods accept the path format argument which is by `wxPATH_NATIVE` by default meaning to use the path format native for the current platform.

The path format affects the operation of `wxFileName` functions in several ways: first and foremost, it defines the path separator character to use, but it also affects other things such as whether the path has the drive part or not.

```
enum wxPathFormat
{
    wxPATH_NATIVE = 0,      // the path format for the current platform
    wxPATH_UNIX,
    wxPATH_BEOS = wxPATH_UNIX,
    wxPATH_MAC,
    wxPATH_DOS,
    wxPATH_WIN = wxPATH_DOS,
```

```
wxPATH_OS2 = wxPATH_DOS ,  
wxPATH_VMS ,  
  
wxPATH_MAX // Not a valid value for specifying path format  
}
```

## File name format

`wxFileName` currently supports the file names in the Unix, DOS/Windows, Mac OS and VMS formats. Although these formats are quite different, `wxFileName` tries to treat them all in the same generic way. It supposes that all file names consist of the following parts: the volume (also known as drive under Windows or device under VMS), the path which is a sequence of directory names separated by the *path separators* (p. 618) and the full filename itself which, in turn, is composed from the base file name and the extension. All of the individual components of the file name may be empty and, for example, the volume name is always empty under Unix, but if they are all empty simultaneously, the filename object is considered to be in an invalid state and *IsOk* (p. 622) returns `false` for it.

File names can be case-sensitive or not, the function *IsCaseSensitive* (p. 621) allows to determine this.

The rules for determining whether the file name is absolute or relative also depend on the file name format and the only portable way to answer this question is to use *IsAbsolute* (p. 621) or *IsRelative* (p. 622) method. Note that on Windows, "X:" refers to the current working directory on drive X. Therefore, a `wxFileName` instance constructed from for example "X:dir/file.ext" treats the portion beyond drive separator as being relative to that directory.

To ensure that the filename is absolute, you may use *MakeAbsolute* (p. 623). There is also an inverse function *MakeRelativeTo* (p. 623) which undoes what *Normalize(wxPATH\_NORM\_DOTS)* (p. 624) does.

Other functions returning information about the file format provided by this class are *GetVolumeSeparator* (p. 620), *IsPathSeparator* (p. 622).

## File name construction

You can initialize a `wxFileName` instance using one of the following functions:

- wxFileName* constructors (p. 613)
- Assign* (p. 613)
- AssignCwd* (p. 614)
- AssignDir* (p. 614)
- AssignHomeDir* (p. 614)
- AssignHomeTempFileName* (p. 614)
- DirName* (p. 615)
- FileName* (p. 616)
- operator =* (p. 628)

## File tests

Before doing other tests, you should use *IsOk* (p. 622) to verify that the filename is well defined. If it is, *FileExists* (p. 616) can be used to test whether a file with such name exists and *DirExists* (p. 615) can be used to test for directory existence.

File names should be compared using *SameAs* (p. 625) method or *operator ==* (p. 628).

For testing basic access modes, you can use:

*IsDirWritable* (p. 621)  
*IsDirReadable* (p. 621)  
*IsFileWritable* (p. 622)  
*IsFileReadable* (p. 622)  
*IsFileExecutable* (p. 621)

## File name components

These functions allow to examine and modify the individual directories of the path:

*AppendDir* (p. 613)  
*InsertDir* (p. 621)  
*GetDirCount* (p. 616) *PrependDir* (p. 625)  
*RemoveDir* (p. 625)  
*RemoveLastDir* (p. 625)

To change the components of the file name individually you can use the following functions:

*GetExt* (p. 617)  
*GetName* (p. 617)  
*GetVolume* (p. 620)  
*HasExt* (p. 620)  
*HasName* (p. 620)  
*HasVolume* (p. 620)  
*SetExt* (p. 626)  
*ClearExt* (p. 615)  
*SetEmptyExt* (p. 626)  
*SetName* (p. 626)  
*SetVolume* (p. 627)

## Operations

These methods allow to work with the file creation, access and modification times. Note that not all filesystems under all platforms implement these times in the same way. For example, the access time under Windows has a resolution of one day (so it is really the access date and not time). The access time may be updated when the file is executed or not depending on the platform.

*GetModificationTime* (p. 617)  
*GetTimes* (p. 620)

*SetTimes* (p. 626)

*Touch* (p. 627)

Other file system operations functions are:

*Mkdir* (p. 624)

*Rmdir* (p. 625)

## **wxFileName::wxFileName**

### **wxFileName()**

Default constructor.

### **wxFileName(const wxFileName& filename)**

Copy constructor.

### **wxFileName(const wxString& fullpath, wxPathFormat format = wxPATH\_NATIVE)**

Constructor taking a full filename. If it terminates with a '/', a directory path is constructed (the name will be empty), otherwise a file name and extension are extracted from it.

### **wxFileName(const wxString& path, const wxString& name, wxPathFormat format = wxPATH\_NATIVE)**

Constructor from a directory name and a file name.

### **wxFileName(const wxString& path, const wxString& name, const wxString& ext, wxPathFormat format = wxPATH\_NATIVE)**

Constructor from a directory name, base file name and extension.

### **wxFileName(const wxString& volume, const wxString& path, const wxString& name, const wxString& ext, wxPathFormat format = wxPATH\_NATIVE)**

Constructor from a volume name, a directory name, base file name and extension.

## **wxFileName::AppendDir**

### **void AppendDir(const wxString& dir)**

Appends a directory component to the path. This component should contain a single directory name level, i.e. not contain any path or volume separators nor should it be empty, otherwise the function does nothing (and generates an assert failure in debug build).

## **wxFileName::Assign**

### **void Assign(const wxFileName& filepath)**

### **void Assign(const wxString& fullpath, wxPathFormat format = wxPATH\_NATIVE)**

**void Assign(const wxString& volume, const wxString& path, const wxString& name, const wxString& ext, bool hasExt, wxPathFormat format = wxPATH\_NATIVE)**

**void Assign(const wxString& volume, const wxString& path, const wxString& name, const wxString& ext, wxPathFormat format = wxPATH\_NATIVE)**

**void Assign(const wxString& path, const wxString& name, wxPathFormat format = wxPATH\_NATIVE)**

**void Assign(const wxString& path, const wxString& name, const wxString& ext, wxPathFormat format = wxPATH\_NATIVE)**

Creates the file name from various combinations of data.

### **wxFileName::AssignCwd**

**static void AssignCwd(const wxString& volume = wxEmptyString)**

Makes this object refer to the current working directory on the specified volume (or current volume if *volume* is empty).

#### **See also**

*GetCwd* (p. 616)

### **wxFileName::AssignDir**

**void AssignDir(const wxString& dir, wxPathFormat format = wxPATH\_NATIVE)**

Sets this file name object to the given directory name. The name and extension will be empty.

### **wxFileName::AssignHomeDir**

**void AssignHomeDir()**

Sets this file name object to the home directory.

### **wxFileName::AssignTempFileName**

**void AssignTempFileName(const wxString& prefix, wxFile \*fileTemp = NULL)**

The function calls *CreateTempFileName* (p. 615) to create a temporary file and sets this object to the name of the file. If a temporary file couldn't be created, the object is put into the *invalid* (p. 622) state.

### **wxFileName::Clear**

**void Clear()**

Reset all components to default, uninitialized state.

**wxFileName::ClearExt****void SetClearExt()**

Removes the extension from the file name resulting in a file name with no trailing dot.

**See also**

*SetExt* (p. 626) *SetEmptyExt* (p. 626)

**wxFileName::CreateTempFileName****static wxString CreateTempFileName(const wxString& *prefix*, wxFile \**fileTemp* = NULL)**

Returns a temporary file name starting with the given *prefix*. If the *prefix* is an absolute path, the temporary file is created in this directory, otherwise it is created in the default system directory for the temporary files or in the current directory.

If the function succeeds, the temporary file is actually created. If *fileTemp* is not `NULL`, this file will be opened using the name of the temporary file. When possible, this is done in an atomic way ensuring that no race condition occurs between the temporary file name generation and opening it which could often lead to security compromise on the multiuser systems. If *fileTemp* is `NULL`, the file is only created, but not opened.

Under Unix, the temporary file will have read and write permissions for the owner only to minimize the security problems.

**Parameters**

*prefix*

Prefix to use for the temporary file name construction

*fileTemp*

The file to open or `NULL` to just get the name

**Return value**

The full temporary file name or an empty string on error.

**wxFileName::DirExists****bool DirExists() const****static bool DirExists(const wxString& *dir*)**

Returns `true` if the directory with this name exists.

**wxFileName::DirName****static wxFileName DirName(const wxString& *dir*, wxPathFormat *format* =**

`wxPATH_NATIVE)`

Returns the object corresponding to the directory with the given name. The *dir* parameter may have trailing path separator or not.

### **wxFileName::FileExists**

**bool FileExists() const**

**static bool FileExists(const wxString& file)**

Returns `true` if the file with this name exists.

#### **See also**

*DirExists* (p. 615)

### **wxFileName::FileName**

**static wxFileName FileName(const wxString& file, wxPathFormat format = wxPATH\_NATIVE)**

Returns the file name object corresponding to the given *file*. This function exists mainly for symmetry with *DirName* (p. 615).

### **wxFileName::GetCwd**

**static wxString GetCwd(const wxString& volume = "")**

Retrieves the value of the current working directory on the specified volume. If the volume is empty, the program's current working directory is returned for the current volume.

#### **Return value**

The string containing the current working directory or an empty string on error.

#### **See also**

*AssignCwd* (p. 614)

### **wxFileName::GetDirCount**

**size\_t GetDirCount() const**

Returns the number of directories in the file name.

### **wxFileName::GetDirs**

**const wxArrayString& GetDirs() const**

Returns the directories in string array form.



**wxFileName::GetExt****wxString GetExt() const**

Returns the file name extension.

**wxFileName::GetForbiddenChars****static wxString GetForbiddenChars(wxPathFormat *format* = wxPATH\_NATIVE)**

Returns the characters that can't be used in filenames and directory names for the specified format.

**wxFileName::GetFormat****static wxPathFormat GetFormat(wxPathFormat *format* = wxPATH\_NATIVE)**

Returns the canonical path format for this platform.

**wxFileName::GetFullName****wxString GetFullName() const**

Returns the full name (including extension but excluding directories).

**wxFileName::GetFullPath****wxString GetFullPath(wxPathFormat *format* = wxPATH\_NATIVE) const**

Returns the full path with name and extension.

**wxFileName::GetHomeDir****static wxString GetHomeDir()**

Returns the home directory.

**wxFileName::GetLongPath****wxString GetLongPath() const**

Return the long form of the path (returns identity on non-Windows platforms)

**wxFileName::GetModificationTime****wxDateTime GetModificationTime() const**

Returns the last time the file was last modified.

**wxFileName::GetName**

**wxString GetName() const**

Returns the name part of the filename (without extension).

**See also**

*GetFullName* (p. 617)

**wxFileName::GetPath**

**wxString GetPath(int flags = wxPATH\_GET\_VOLUME, wxPathFormat format = wxPATH\_NATIVE) const**

Returns the path part of the filename (without the name or extension). The possible flags values are:

**wxPATH\_GET\_VOLUME**     Return the path with the volume (does nothing for the filename formats without volumes), otherwise the path without volume part is returned.

**wxPATH\_GET\_SEPARATOR**     Return the path with the trailing separator, if this flag is not given there will be no separator at the end of the path.

**wxFileName::GetPathSeparator**

**static wxChar GetPathSeparator(wxPathFormat format = wxPATH\_NATIVE)**

Returns the usually used path separator for this format. For all formats but `wxPATH_DOS` there is only one path separator anyhow, but for DOS there are two of them and the native one, i.e. the backslash is returned by this method.

**See also**

*GetPathSeparators* (p. 618)

**wxFileName::GetPathSeparators**

**static wxString GetPathSeparators(wxPathFormat format = wxPATH\_NATIVE)**

Returns the string containing all the path separators for this format. For all formats but `wxPATH_DOS` this string contains only one character but for DOS and Windows both `'/'` and `'\'` may be used as separators.

**See also**

*GetPathSeparator* (p. 618)

**wxFileName::GetPathTerminators**

**static wxString GetPathTerminators(wxPathFormat format = wxPATH\_NATIVE)**

Returns the string of characters which may terminate the path part. This is the same as

*GetPathSeparators* (p. 618) except for VMS path format where ] is used at the end of the path part.

### **wxFileName::GetPathWithSep**

**wxString GetPathWithSep(wxPathFormat *format* = wxPATH\_NATIVE) const**

Returns the path with the trailing separator, useful for appending the name to the given path.

This is the same as calling *GetPath* (p. 618) (wxPATH\_GET\_VOLUME | wxPATH\_GET\_SEPARATOR, *format*).

### **wxFileName::GetShortPath**

**wxString GetShortPath() const**

Return the short form of the path (returns identity on non-Windows platforms).

### **wxFileName::GetSize**

**wxULongLong GetSize() const**

**static wxULongLong GetSize(const wxString& *filename*)**

Returns the size of this file (first form) or the size of the given file (second form). If the file does not exist or its size could not be read (because e.g. the file is locked by another process) the returned value is *wxInvalidSize*.

### **wxFileName::GetHumanReadableSize**

**wxString GetHumanReadableSize(const wxString& *failmsg* = "Not available", int *precision* = 1) const**

**static wxString GetHumanReadableSize(const wxULongLong& *bytes*, const wxString& *nullsize* = "Not available", int *precision* = 1)**

Returns the size of this file (first form) or the given number of bytes (second form) in a human-readable form.

If the size could not be retrieved the *failmsg* string is returned (first form). If *bytes* is *wxInvalidSize* or zero, then *nullsize* is returned (second form).

In case of success, the returned string is a floating-point number with *precision* decimal digits followed by the size unit (B, kB, MB, GB, TB: respectively bytes, kilobytes, megabytes, gigabytes, terabytes).

### **wxFileName::GetTempDir**

**static wxString GetTempDir()**

Returns the directory used for temporary files.

### **wxFileName::GetTimes**

**bool GetTimes(wxDateTime\* dtAccess, wxDateTime\* dtMod, wxDateTime\* dtCreate)  
const**

Returns the last access, last modification and creation times. The last access time is updated whenever the file is read or written (or executed in the case of Windows), last modification time is only changed when the file is written to. Finally, the creation time is indeed the time when the file was created under Windows and the inode change time under Unix (as it is impossible to retrieve the real file creation time there anyhow) which can also be changed by many operations after the file creation.

If no filename or extension is specified in this instance of wxFileName (and therefore *IsDir* (p. 622) returns `true`) then this function will return the directory times of the path specified by *GetPath* (p. 618), otherwise the file times of the file specified by *GetFullPath* (p. 617).

Any of the pointers may be `NULL` if the corresponding time is not needed.

#### **Return value**

`true` on success, `false` if we failed to retrieve the times.

### **wxFileName::GetVolume**

**wxString GetVolume() const**

Returns the string containing the volume for this file name, empty if it doesn't have one or if the file system doesn't support volumes at all (for example, Unix).

### **wxFileName::GetVolumeSeparator**

**static wxString GetVolumeSeparator(wxPathFormat format = wxPATH\_NATIVE)**

Returns the string separating the volume from the path for this format.

### **wxFileName::HasExt**

**bool HasExt() const**

Returns `true` if an extension is present.

### **wxFileName::HasName**

**bool HasName() const**

Returns `true` if a name is present.

### **wxFileName::HasVolume**

**bool HasVolume() const**

Returns `true` if a volume specifier is present.

**wxFileName::InsertDir****void InsertDir(size\_t before, const wxString& dir)**

Inserts a directory component before the zero-based position in the directory list. Please see *AppendDir* (p. 613) for important notes.

**wxFileName::IsAbsolute****bool IsAbsolute(wxPathFormat format = wxPATH\_NATIVE)**

Returns `true` if this filename is absolute.

**wxFileName::IsCaseSensitive****static bool IsCaseSensitive(wxPathFormat format = wxPATH\_NATIVE)**

Returns `true` if the file names of this type are case-sensitive.

**wxFileName::IsDirReadable****bool IsDirReadable() const****static bool IsDirReadable(const wxString& dir)**

Returns `true` if the directory component of this instance (or given *dir*) is an existing directory and this process has read permissions on it. Read permissions on a directory mean that you can list the directory contents but it doesn't imply that you have read permissions on the files contained.

**wxFileName::IsDirWritable****bool IsDirWritable() const****static bool IsDirWritable(const wxString& dir)**

Returns `true` if the directory component of this instance (or given *dir*) is an existing directory and this process has write permissions on it. Write permissions on a directory mean that you can create new files in the directory.

**wxFileName::IsFileExecutable****bool IsFileExecutable() const****static bool IsFileExecutable(const wxString& file)**

Returns `true` if a file with this name exists and if this process has execute permissions on

it.

### **wxFileName::IsFileReadable**

**bool IsFileReadable() const**

**static bool IsFileReadable(const wxString& file)**

Returns `true` if a file with this name exists and if this process has read permissions on it.

### **wxFileName::IsFileWritable**

**bool IsFileWritable() const**

**static bool IsFileWritable(const wxString& file)**

Returns `true` if a file with this name exists and if this process has write permissions on it.

### **wxFileName::IsOk**

**bool IsOk() const**

Returns `true` if the filename is valid, `false` if it is not initialized yet. The assignment functions and *Clear* (p. 614) may reset the object to the uninitialized, invalid state (the former only do it on failure).

### **wxFileName::IsPathSeparator**

**static bool IsPathSeparator(wxChar ch, wxPathFormat format = wxPATH\_NATIVE)**

Returns `true` if the char is a path separator for this format.

### **wxFileName::IsRelative**

**bool IsRelative(wxPathFormat format = wxPATH\_NATIVE)**

Returns `true` if this filename is not absolute.

### **wxFileName::IsDir**

**bool IsDir() const**

Returns `true` if this object represents a directory, `false` otherwise (i.e. if it is a file). Note that this method doesn't test whether the directory or file really exists, you should use *DirExists* (p. 615) or *FileExists* (p. 616) for this.

### **wxFileName::MacFindDefaultTypeAndCreator**

**static bool MacFindDefaultTypeAndCreator(const wxString& ext, wxUint32\* type, wxUint32\* creator)**

On Mac OS, gets the common type and creator for the given extension.

### **wxFileName::MacRegisterDefaultTypeAndCreator**

**static void MacRegisterDefaultTypeAndCreator(const wxString& ext, wxUint32 type, wxUint32 creator)**

On Mac OS, registers application defined extensions and their default type and creator.

### **wxFileName::MacSetDefaultTypeAndCreator**

**bool MacSetDefaultTypeAndCreator()**

On Mac OS, looks up the appropriate type and creator from the registration and then sets it.

### **wxFileName::MakeAbsolute**

**bool MakeAbsolute(const wxString& cwd = wxEmptyString, wxPathFormat format = wxPATH\_NATIVE)**

Make the file name absolute. This is a shortcut for `Normalize (p. 624) (wxPATH_NORM_DOTS | wxPATH_NORM_ABSOLUTE | wxPATH_NORM_TILDE, cwd, format)`.

#### **See also**

*MakeRelativeTo* (p. 623), *Normalize* (p. 624), *IsAbsolute* (p. 621)

### **wxFileName::MakeRelativeTo**

**bool MakeRelativeTo(const wxString& pathBase = wxEmptyString, wxPathFormat format = wxPATH\_NATIVE)**

This function tries to put this file name in a form relative to *pathBase*. In other words, it returns the file name which should be used to access this file if the current directory were *pathBase*.

*pathBase*

the directory to use as root, current directory is used by default

*format*

the file name format, native by default

#### **Return value**

`true` if the file name has been changed, `false` if we failed to do anything with it (currently this only happens if the file name is on a volume different from the volume specified by *pathBase*).

**See also**

*Normalize* (p. 624)

**wxFileName::Mkdir**

**bool Mkdir**(int *perm* = 0777, int *flags* = 0)

**static bool Mkdir**(const wxString& *dir*, int *perm* = 0777, int *flags* = 0)

*dir*

the directory to create

*perm*

the permissions for the newly created directory

*flags*

if the flags contain `wxPATH_MKDIR_FULL` flag, try to create each directory in the path and also don't return an error if the target directory already exists.

**Return value**

Returns `true` if the directory was successfully created, `false` otherwise.

**wxFileName::Normalize**

**bool Normalize**(int *flags* = `wxPATH_NORM_ALL`, const wxString& *cwd* = `wxEmptyString`, wxPathFormat *format* = `wxPATH_NATIVE`)

Normalize the path. With the default flags value, the path will be made absolute, without any `".."` and `"."` and all environment variables will be expanded in it.

*flags*

The kind of normalization to do with the file name. It can be any or-combination of the following constants:

**wxPATH\_NORM\_ENV\_VARS** replace env vars with their values

**wxPATH\_NORM\_DOTS** squeeze all `..` and `.` when possible; if there are too many `..` and thus they cannot be all removed, `false` will be returned

**wxPATH\_NORM\_CASE** if filesystem is case insensitive, transform to lower case

**wxPATH\_NORM\_ABSOLUTE** make the path absolute prepending *cwd*

**wxPATH\_NORM\_LONG** make the path the long form

**wxPATH\_NORM\_SHORTCUT** resolve if it is a shortcut (Windows only)



<b>wxPATH_NORM_TILDE</b>	replace ~ and ~user (Unix only)
<b>wxPATH_NORM_ALL</b>	all of previous flags except wxPATH_NORM_CASE

*cwd*

If not empty, this directory will be used instead of current working directory in normalization (see wxPATH\_NORM\_ABSOLUTE).

*format*

The file name format to use when processing the paths, native by default.

### Return value

`true` if normalization was successfully or `false` otherwise.

### wxFileName::PrependDir

**void PrependDir(const wxString& dir)**

Prepends a directory to the file path. Please see *AppendDir* (p. 613) for important notes.

### wxFileName::RemoveDir

**void RemoveDir(size\_t pos)**

Removes the specified directory component from the path.

### See also

*GetDirCount* (p. 616)

### wxFileName::RemoveLastDir

**void RemoveLastDir()**

Removes last directory component from the path.

### wxFileName::Rmdir

**bool Rmdir()**

**static bool Rmdir(const wxString& dir)**

Deletes the specified directory from the file system.

### wxFileName::SameAs

**bool SameAs(const wxFileName& filepath, wxPathFormat format = wxPATH\_NATIVE) const**

Compares the filename using the rules of this platform.

### **wxFileName::SetCwd**

**bool SetCwd()**

**static bool SetCwd(const wxString& cwd)**

Changes the current working directory.

### **wxFileName::SetExt**

**void SetExt(const wxString& ext)**

Sets the extension of the file name. Setting an empty string as the extension will remove the extension resulting in a file name without a trailing dot, unlike a call to *SetEmptyExt* (p. 626).

#### **See also**

*SetEmptyExt* (p. 626) *ClearExt* (p. 615)

### **wxFileName::SetEmptyExt**

**void SetEmptyExt()**

Sets the extension of the file name to be an empty extension. This is different from having no extension at all as the file name will have a trailing dot after a call to this method.

#### **See also**

*SetExt* (p. 626) *ClearExt* (p. 615)

### **wxFileName::SetFullName**

**void SetFullName(const wxString& fullname)**

The full name is the file name and extension (but without the path).

### **wxFileName::SetName**

**void SetName(const wxString& name)**

Sets the name part (without extension).

#### **See also**

*SetFullName* (p. 626)

### **wxFileName::SetTimes**

**bool SetTimes(const wxDateTime\* dtAccess, const wxDateTime\* dtMod, const wxDateTime\* dtCreate)**

Sets the file creation and last access/modification times (any of the pointers may be NULL).

### **wxFileName::SetVolume**

**void SetVolume(const wxString& volume)**

Sets the volume specifier.

### **wxFileName::SplitPath**

**static void SplitPath(const wxString& fullpath, wxString\* volume, wxString\* path, wxString\* name, wxString\* ext, bool \*hasExt = NULL, wxPathFormat format = wxPATH\_NATIVE)**

**static void SplitPath(const wxString& fullpath, wxString\* volume, wxString\* path, wxString\* name, wxString\* ext, wxPathFormat format = wxPATH\_NATIVE)**

**static void SplitPath(const wxString& fullpath, wxString\* path, wxString\* name, wxString\* ext, wxPathFormat format = wxPATH\_NATIVE)**

This function splits a full file name into components: the volume (with the first version) path (including the volume in the second version), the base name and the extension. Any of the output parameters (*volume*, *path*, *name* or *ext*) may be NULL if you are not interested in the value of a particular component. Also, *fullpath* may be empty on entry.

On return, *path* contains the file path (without the trailing separator), *name* contains the file name and *ext* contains the file extension without leading dot. All three of them may be empty if the corresponding component is. The old contents of the strings pointed to by these parameters will be overwritten in any case (if the pointers are not NULL).

Note that for a filename "foo." the extension is present, as indicated by the trailing dot, but empty. If you need to cope with such cases, you should use *hasExt* instead of relying on testing whether *ext* is empty or not.

### **wxFileName::SplitVolume**

**static void SplitVolume(const wxString& fullpath, wxString\* volume, wxString\* path, wxPathFormat format = wxPATH\_NATIVE)**

Splits the given *fullpath* into the volume part (which may be empty) and the pure path part, not containing any volume.

### **See also**

*SplitPath* (p. 627)

### **wxFileName::Touch**

**bool Touch()**

Sets the access and modification times to the current moment.

**wxFileName::operator=**

**wxFileName& operator operator=(const wxFileName& filename)**

**wxFileName& operator operator=(const wxString& filename)**

Assigns the new value to this filename object.

**wxFileName::operator==**

**bool operator operator==(const wxFileName& filename) const**

**bool operator operator==(const wxString& filename) const**

Returns `true` if the filenames are equal. The string *filenames* is interpreted as a path in the native filename format.

**wxFileName::operator!=**

**bool operator operator!=(const wxFileName& filename) const**

**bool operator operator!=(const wxString& filename) const**

Returns `true` if the filenames are different. The string *filenames* is interpreted as a path in the native filename format.

## **wxFileOutputStream**

This class represents data written to a file. There are actually two such groups of classes: this one is based on *wxFile* (p. 591) whereas *wxFileInputStream* (p. 589) is based in the *wxFile* (p. 584) class.

Note that *SeekO()* (p. 1157) can seek beyond the end of the stream (file) and will thus not return *wxInvalidOffset* for that.

**Derived from**

*wxOutputStream* (p. 1156)

**Include files**

<wx/wfstream.h>

**See also**

*wxBufferedOutputStream* (p. 161), *wxFileInputStream* (p. 608), *wxFileInputStream* (p. 589)

**wxFileOutputStream::wxFileOutputStream****wxFileOutputStream(const wxString& ofileName)**

Creates a new file with *ofilename* name and initializes the stream in write-only mode.

**wxFileOutputStream(wxFile& file)**

Initializes a file stream in write-only mode using the file I/O object *file*.

**wxFileOutputStream(int fd)**

Initializes a file stream in write-only mode using the file descriptor *fd*.

**wxFileOutputStream::~~wxFileOutputStream****~wxFileOutputStream()**

Destructor.

**wxFileOutputStream::IsOk****bool IsOk() const**

Returns true if the stream is initialized and ready.

**wxFilePickerCtrl**

This control allows the user to select a file. The generic implementation is a button which brings up a *wxFileDialog* (p. 600) when clicked. Native implementation may differ but this is usually a (small) widget which give access to the file-chooser dialog. It is only available if `wxUSE_FILEPICKERCTRL` is set to 1 (the default).

**Derived from***wxPickerBase* (p. 1183)*wxControl* (p. 285)*wxWindow* (p. 1795)*wxEvtHandler* (p. 576)*wxObject* (p. 1148)**Include files**`<wx/filepicker.h>`**Window styles**

**wxFLP\_DEFAULT\_STYLE** The default style: includes `wxFLP_OPEN` | `wxFLP_FILE_MUST_EXIST` and, under `wxMSW` only,

wxFLP\_USE\_TEXTCTRL.

- wxFLP\_USE\_TEXTCTRL** Creates a text control to the left of the picker button which is completely managed by the *wxFilePickerCtrl* (p. 629) and which can be used by the user to specify a path (see *SetPath* (p. 632)). The text control is automatically synchronized with button's value. Use functions defined in *wxPickerBase* (p. 1183) to modify the text control.
- wxFLP\_OPEN** Creates a picker which allows the user to select a file to open.
- wxFLP\_SAVE** Creates a picker which allows the user to select a file to save.
- wxFLP\_OVERWRITE\_PROMPT** Can be combined with *wxFLP\_SAVE* only: ask confirmation to the user before selecting a file.
- wxFLP\_FILE\_MUST\_EXIST** Can be combined with *wxFLP\_OPEN* only: the selected file must be an existing file.
- wxFLP\_CHANGE\_DIR** Change current working directory on each user file selection change.
- NB: the *wxFD\_MULTIPLE* style of *wxFileDialog* is not supported!

### Event handling

To process a file picker event, use these event handler macros to direct input to member functions that take a *wxFileDirPickerEvent* (p. 632) argument.

- EVT\_FILEPICKER\_CHANGED(id, func)** The user changed the file selected in the control either using the button or using text control (see *wxFLP\_USE\_TEXTCTRL*; note that in this case the event is fired only if the user's input is valid, e.g. an existing file path if *wxFLP\_FILE\_MUST\_EXIST* was given).

### See also

*wxFileDialog* (p. 600),  
*wxFileDirPickerEvent* (p. 632)

### wxFilePickerCtrl::wxFilePickerCtrl

```
wxFilePickerCtrl(wxWindow *parent, wxWindowID id, const wxString& path =  
wxEmptyString, const wxString& message = "Select a file", const wxString& wildcard =  
".*.*", const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,  
long style = wxFLP_DEFAULT_STYLE, const wxValidator& validator =  
wxDefaultValidator, const wxString& name = "filepickerctrl")
```

Initializes the object and calls *Create* (p. 631) with all the parameters.

### **wxFilePickerCtrl::Create**

```
bool Create(wxWindow *parent, wxWindowID id, const wxString& path =  
wxEmptyString, const wxString& message = "Select a file", const wxString& wildcard =  
"*.*", const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,  
long style = wxFLP_DEFAULT_STYLE, const wxValidator& validator =  
wxDefaultValidator, const wxString& name = "filepickerctrl")
```

#### **Parameters**

*parent*

Parent window, must not be non-NULL.

*id*

The identifier for the control.

*path*

The initial file shown in the control. Must be a valid path to a file or the empty string.

*message*

The message shown to the user in the *wxFileDialog* (p. 600) shown by the control.

*wildcard*

A wildcard which defines user-selectable files (use the same syntax as for *wxFileDialog* (p. 600)'s wildcards).

*pos*

Initial position.

*size*

Initial size.

*style*

The window style, see `wxFLP_*` flags.

*validator*

Validator which can be used for additional date checks.

*name*

Control name.

#### **Return value**

`true` if the control was successfully created or `false` if creation failed.

### **wxFilePickerCtrl::GetPath**

**wxString GetPath() const**

Returns the absolute path of the currently selected file.

### **wxFilePickerCtrl::SetPath**

**void SetPath(const wxString &filename)**

Sets the absolute path of the currently selected file. This must be a valid file if the `wxFLP_FILE_MUST_EXIST` style was given.

## **wxFileDirPickerEvent**

This event class is used for the events generated by *wxFilePickerCtrl* (p. 629) and by *wxDirPickerCtrl* (p. 515).

### **Derived from**

*wxCommandEvent* (p. 250)

*wxEvent* (p. 572)

*wxObject* (p. 1148)

### **Include files**

<wx/filepicker.h>

### **Event handling**

To process input from a *wxFilePickerCtrl* or from a *wxDirPickerCtrl*, use one of these event handler macros to direct input to member function that take a *wxFileDirPickerEvent* (p. 632) argument:

**EVT\_FILEPICKER\_CHANGED(id, func)** Generated whenever the selected file changes.

**EVT\_DIRPICKER\_CHANGED(id, func)** Generated whenever the selected directory changes.

### **See also**

*wxfilepickerctrl* (p. 629)

### **wxFileDirPickerEvent::wxFileDirPickerEvent**

**wxFileDirPickerEvent(wxEventType type, wxObject \* generator, int id, const wxString& path)**



The constructor is not normally used by the user code.

### **wxFileDirPickerEvent::GetPath**

**wxString GetPath() const**

Retrieve the absolute path of the file/directory the user has just selected.

### **wxFileDirPickerEvent::SetPath**

**void SetPath(const wxString &path)**

Set the absolute path of the file/directory associated with the event.

## **wxFileStream**

**Derived from**

*wxFileOutputStream* (p. 628), *wxFileInputStream* (p. 608)

**Include files**

<wx/wfstream.h>

**See also**

*wxStreamBuffer* (p. 1547)

### **wxFileStream::wxFileStream**

**wxFileStream(const wxString& iofilename)**

Initializes a new file stream in read-write mode using the specified *iofilename* name.

## **wxFileSystem**

This class provides an interface for opening files on different file systems. It can handle absolute and/or local filenames. It uses a system of *handlers* (p. 636) to provide access to user-defined virtual file systems.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/filesys.h>

**See Also**

*wxFileSystemHandler* (p. 636), *wxFSFile* (p. 692), *Overview* (p. 2075)

## **wxFileSystem::wxFileSystem**

### **wxFileSystem()**

Constructor.

### **wxFileSystem::AddHandler**

#### **static void AddHandler(wxFileSystemHandler \*handler)**

This static function adds new handler into the list of *handlers* (p. 636) which provide access to virtual FS. Note that if two handlers for the same protocol are added, the last one added takes precedence.

#### **Note**

You can call:

```
wxFileSystem::AddHandler(new My_FS_Handler);
```

This is because (a) `AddHandler` is a static method, and (b) the handlers are deleted in `wxFileSystem`'s destructor so that you don't have to care about it.

### **wxFileSystem::HasHandlerForPath**

#### **static bool HasHandlerForPath(const wxString & location)**

This static function returns `true` if there is a registered handler which can open the given location.

### **wxFileSystem::ChangePathTo**

#### **void ChangePathTo(const wxString& location, bool is\_dir = false)**

Sets the current location. *location* parameter passed to *OpenFile* (p. 636) is relative to this path.

**Caution!** Unless *is\_dir* is true the *location* parameter is not the directory name but the name of the file in this directory. All these commands change the path to "dir/subdir/":

```
ChangePathTo("dir/subdir/xh.htm");  
ChangePathTo("dir/subdir", true);  
ChangePathTo("dir/subdir/", true);
```

#### **Parameters**

*location*

the new location. Its meaning depends on the value of *is\_dir*

*is\_dir*

if true *location* is new directory. If false (default) *location* is **file** in the new directory.

### Example

```
f = fs -> OpenFile("hello.htm"); // opens file 'hello.htm'
fs -> ChangePathTo("subdir/folder", true);
f = fs -> OpenFile("hello.htm"); // opens file
'subdir/folder/hello.htm' !!
```

### **wxFileSystem::GetPath**

#### **wxString GetPath()**

Returns actual path (set by *ChangePathTo* (p. 634)).

### **wxFileSystem::FileNameToURL**

#### **static wxString FileNameToURL(wxFileName filename)**

Converts filename into URL.

#### **See also**

*wxFileSystem::URLToFileName* (p. 636), *wxFileName* (p. 609)

### **wxFileSystem::FindFileInPath**

#### **bool FindFileInPath(wxString \*str, const wxChar \*path, const wxChar \*file)**

Looks for the file with the given name *file* in a colon or semi-colon (depending on the current platform) separated list of directories in *path*. If the file is found in any directory, returns `true` and the full path of the file in *str*, otherwise returns `false` and doesn't modify *str*.

#### **Parameters**

*str*

Receives the full path of the file, must not be `NULL`

*path*

`wxPATH_SEP`-separated list of directories

*file*

the name of the file to look for

### **wxFileSystem::FindFirst**

**wxString FindFirst(const wxString& wildcard, int flags = 0)**

Works like *wxFindFirstFile* (p. 1920). Returns name of the first filename (within filesystem's current path) that matches *wildcard*. *flags* may be one of *wxFILE* (only files), *wxDIR* (only directories) or 0 (both).

**wxFileSystem::FindNext****wxString FindNext()**

Returns the next filename that matches parameters passed to *FindFirst* (p. 635).

**wxFileSystem::OpenFile****wxFSFile\* OpenFile(const wxString& location, int flags = wxFS\_READ)**

Opens the file and returns a pointer to a *wxFSFile* (p. 692) object or NULL if failed. It first tries to open the file in relative scope (based on value passed to *ChangePathTo()* method) and then as an absolute path. Note that the user is responsible for deleting the returned *wxFSFile*.

*flags* can be one or more of the following bit values ored together:

```
// Open Bit Flags
enum {
    wxFS_READ = 1,          // Open for reading
    wxFS_SEEKABLE = 4       // Returned stream will be seekable
};
```

A stream opened with just the default *wxFS\_READ* flag may or may not be seekable depending on the underlying source. Passing *wxFS\_READ | wxFS\_SEEKABLE* for *flags* will back a stream that is not natively seekable with memory or a file and return a stream that is always seekable.

**wxFileSystem::URLToFileName****static wxFileName URLToFileName(const wxString& url)**

Converts URL into a well-formed filename. The URL must use the *file* protocol.

**See also**

*wxFileSystem::FileNameToURL* (p. 635), *wxFileName* (p. 609)

**wxFileSystemHandler**

Classes derived from *wxFileSystemHandler* are used to access virtual file systems. Its public interface consists of two methods: *CanOpen* (p. 637) and *OpenFile* (p. 639). It provides additional protected methods to simplify the process of opening the file: *GetProtocol*, *GetLeftLocation*, *GetRightLocation*, *GetAnchor*, *GetMimeTypeFromExt*.

Please have a look at *overview* (p. 2075) if you don't know how locations are constructed.

Also consult *list of available handlers* (p. 2075).

**wxPerl note:** In wxPerl, you need to derive your file system handler class from `Wx::PIFileSystemHandler`.

#### Notes

- The handlers are shared by all instances of `wxFileSystem`.
- `wxHTML` library provides handlers for local files and HTTP or FTP protocol
- The *location* parameter passed to `OpenFile` or `CanOpen` methods is always an **absolute** path. You don't need to check the FS's current path.

#### Derived from

`wxObject` (p. 1148)

#### Include files

`<wx/filesys.h>`

#### See also

`wxFileSystem` (p. 633), `wxFSFile` (p. 692), *Overview* (p. 2075)

### **wxFileSystemHandler::wxFileSystemHandler**

**wxFileSystemHandler()**

Constructor.

### **wxFileSystemHandler::CanOpen**

**virtual bool CanOpen(const wxString& location)**

Returns true if the handler is able to open this file. This function doesn't check whether the file exists or not, it only checks if it knows the protocol. Example:

```
bool MyHand::CanOpen(const wxString& location)
{
    return (GetProtocol(location) == "http");
}
```

Must be overridden in derived handlers.

### **wxFileSystemHandler::GetAnchor**

**wxString GetAnchor(const wxString& location) const**

Returns the anchor if present in the location. See *wxFSFile* (p. 693) for details.

Example: `GetAnchor("index.htm#chapter2") == "chapter2"`

**Note:** the anchor is NOT part of the left location.

### **wxFileSystemHandler::GetLeftLocation**

**wxString GetLeftLocation(const wxString& location) const**

Returns the left location string extracted from *location*.

Example: `GetLeftLocation("file:myzipfile.zip#zip:index.htm") == "file:myzipfile.zip"`

### **wxFileSystemHandler::GetMimeTypeFromExt**

**wxString GetMimeTypeFromExt(const wxString& location)**

Returns the MIME type based on **extension** of *location*. (While *wxFSFile::GetMimeType* returns real MIME type - either extension-based or queried from HTTP.)

Example : `GetMimeTypeFromExt("index.htm") == "text/html"`

### **wxFileSystemHandler::GetProtocol**

**wxString GetProtocol(const wxString& location) const**

Returns the protocol string extracted from *location*.

Example: `GetProtocol("file:myzipfile.zip#zip:index.htm") == "zip"`

### **wxFileSystemHandler::GetRightLocation**

**wxString GetRightLocation(const wxString& location) const**

Returns the right location string extracted from *location*.

Example : `GetRightLocation("file:myzipfile.zip#zip:index.htm") == "index.htm"`

### **wxFileSystemHandler::FindFirst**

**virtual wxString FindFirst(const wxString& wildcard, int flags = 0)**

Works like *wxFindFirstFile* (p. 1920). Returns name of the first filename (within filesystem's current path) that matches *wildcard*. *flags* may be one of *wxFILE* (only files), *wxDIR* (only directories) or 0 (both).

This method is only called if *CanOpen* (p. 637) returns true.

### **wxFileSystemHandler::FindNext**

**virtual wxString FindNext()**

Returns next filename that matches parameters passed to *FindFirst* (p. 635).

This method is only called if *CanOpen* (p. 637) returns true and *FindFirst* returned a non-empty string.

**wxFileSystemHandler::OpenFile**

**virtual wxFSFile\* OpenFile(wxFileSystem& fs, const wxString& location)**

Opens the file and returns wxFSFile pointer or NULL if failed.

Must be overridden in derived handlers.

**Parameters**

*fs*

Parent FS (the FS from that *OpenFile* was called). See ZIP handler for details of how to use it.

*location*

The **absolute** location of file.

**wxFileType**

This class holds information about a given *file type*. File type is the same as MIME type under Unix, but under Windows it corresponds more to an extension than to MIME type (in fact, several extensions may correspond to a file type). This object may be created in several different ways: the program might know the file extension and wish to find out the corresponding MIME type or, conversely, it might want to find the right extension for the file to which it writes the contents of given MIME type. Depending on how it was created some fields may be unknown so the return value of all the accessors **must** be checked: `false` will be returned if the corresponding information couldn't be found.

The objects of this class are never created by the application code but are returned by *wxMimeTypeManager::GetFileTypeFromMimeType* (p. 1112) and *wxMimeTypeManager::GetFileTypeFromExtension* (p. 1112) methods. But it is your responsibility to delete the returned pointer when you're done with it!

A brief reminder about what the MIME types are (see the RFC 1341 for more information): basically, it is just a pair category/type (for example, "text/plain") where the category is a basic indication of what a file is. Examples of categories are "application", "image", "text", "binary", and type is a precise definition of the document format: "plain" in the example above means just ASCII text without any formatting, while "text/html" is the HTML document source.

A MIME type may have one or more associated extensions: "text/plain" will typically correspond to the extension ".txt", but may as well be associated with ".ini" or ".conf".

**Derived from**

None

**Include files**

&lt;wx/mimetype.h&gt;

**See also***wxMimeTypeManager* (p. 1110)**MessageParameters class**

One of the most common usages of MIME is to encode an e-mail message. The MIME type of the encoded message is an example of a *message parameter*. These parameters are found in the message headers ("Content-XXX"). At the very least, they must specify the MIME type and the version of MIME used, but almost always they provide additional information about the message such as the original file name or the charset (for the text documents).

These parameters may be useful to the program used to open, edit, view or print the message, so, for example, an e-mail client program will have to pass them to this program. Because *wxFileType* itself can not know about these parameters, it uses *MessageParameters* class to query them. The default implementation only requires the caller to provide the file name (always used by the program to be called - it must know which file to open) and the MIME type and supposes that there are no other parameters. If you wish to supply additional parameters, you must derive your own class from *MessageParameters* and override *GetParamValue()* function, for example:

```
// provide the message parameters for the MIME type manager
class MailMessageParameters : public wxFileType::MessageParameters
{
public:
    MailMessageParameters(const wxString& filename,
                          const wxString& mimetype)
        : wxFileType::MessageParameters(filename, mimetype)
    {
    }

    virtual wxString GetParamValue(const wxString& name) const
    {
        // parameter names are not case-sensitive
        if ( name.CmpNoCase("charset") == 0 )
            return "US-ASCII";
        else
            return
wxFileType::MessageParameters::GetParamValue(name);
    }
};
```

Now you only need to create an object of this class and pass it to, for example,



*GetOpenCommand* (p. 642) like this:

```
wxString command;
if ( filetype->GetOpenCommand(&command,
                             MailMessageParameters("foo.txt",
"text/plain")) )
{
    // the full command for opening the text documents is in 'command'
    // (it might be "notepad foo.txt" under Windows or "cat foo.txt"
under Unix)
}
else
{
    // we don't know how to handle such files...
}
```

**Windows:** As only the file name is used by the program associated with the given extension anyhow (but no other message parameters), there is no need to ever derive from *MessageParameters* class for a Windows-only program.

### **wxFileType::wxFileType**

#### **wxFileType()**

The default constructor is private because you should never create objects of this type: they are only returned by *wxMimeTypeManager* (p. 1110) methods.

### **wxFileType::~~wxFileType**

#### **~wxFileType()**

The destructor of this class is not virtual, so it should not be derived from.

### **wxFileType::GetMimeType**

#### **bool GetMimeType(wxString\* mimeType)**

If the function returns `true`, the string pointed to by *mimeType* is filled with full MIME type specification for this file type: for example, "text/plain".

### **wxFileType::GetMimeTypes**

#### **bool GetMimeTypes(wxArrayString& mimeTypes)**

Same as *GetMimeType* (p. 641) but returns array of MIME types. This array will contain only one item in most cases but sometimes, notably under Unix with KDE, may contain more MIME types. This happens when one file extension is mapped to different MIME types by KDE, mailcap and mime.types.

### **wxFileType::GetExtensions**

#### **bool GetExtensions(wxArrayString& extensions)**

If the function returns `true`, the array *extensions* is filled with all extensions associated with this file type: for example, it may contain the following two elements for the MIME type "text/html" (notice the absence of the leading dot): "html" and "htm".

**Windows:** This function is currently not implemented: there is no (efficient) way to retrieve associated extensions from the given MIME type on this platform, so it will only return `true` if the `wxFileType` object was created by *GetFileTypeFromExtension* (p. 1112) function in the first place.

### **wxFileType::GetIcon**

**bool GetIcon(wxIconLocation \* iconLoc)**

If the function returns `true`, the *iconLoc* is filled with the location of the icon for this MIME type. A *wxIcon* (p. 894) may be created from *iconLoc* later.

**Windows:** The function returns the icon shown by Explorer for the files of the specified type.

**Mac:** This function is not implemented and always returns `false`.

**Unix:** MIME manager gathers information about icons from GNOME and KDE settings and thus *GetIcon*'s success depends on availability of these desktop environments.

### **wxFileType::GetDescription**

**bool GetDescription(wxString\* desc)**

If the function returns `true`, the string pointed to by *desc* is filled with a brief description for this file type: for example, "text document" for the "text/plain" MIME type.

### **wxFileType::GetOpenCommand**

**bool GetOpenCommand(wxString\* command, MessageParameters& params)**

**wxString GetOpenCommand(const wxString& filename)**

With the first version of this method, if the `true` is returned, the string pointed to by *command* is filled with the command which must be executed (see *wxExecute* (p. 1913)) in order to open the file of the given type. In this case, the name of the file as well as any other parameters is retrieved from *MessageParameters* (p. 640) class.

In the second case, only the filename is specified and the command to be used to open this kind of file is returned directly. An empty string is returned to indicate that an error occurred (typically meaning that there is no standard way to open this kind of files).

### **wxFileType::GetPrintCommand**

**bool GetPrintCommand(wxString\* command, MessageParameters& params)**

If the function returns `true`, the string pointed to by *command* is filled with the command

which must be executed (see *wxExecute* (p. 1913)) in order to print the file of the given type. The name of the file is retrieved from *MessageParameters* (p. 640) class.

### **wxFileType::ExpandCommand**

**static wxString ExpandCommand(const wxString& command,  
MessageParameters& params)**

This function is primarily intended for *GetOpenCommand* and *GetPrintCommand* usage but may be also used by the application directly if, for example, you want to use some non-default command to open the file.

The function replaces all occurrences of

format specification	with
%s	the full file name
%t	the MIME type
%{param}	the value of the parameter <i>param</i>

using the *MessageParameters* object you pass to it.

If there is no '%s' in the command string (and the string is not empty), it is assumed that the command reads the data on stdin and so the effect is the same as "< %s" were appended to the string.

Unlike all other functions of this class, there is no error return for this function.

## **wxFilterClassFactory**

Allows the creation of filter streams to handle compression formats such as gzip and bzip2.

For example, given a filename you can search for a factory that will handle it and create a stream to decompress it:

```
factory = wxFilterClassFactory::Find(filename,  
wxSTREAM_FILEEXT);  
if (factory)  
    stream = factory->NewStream(new  
    wxFFileInputStream(filename));
```

*Find()* (p. 644) can also search for a factory by MIME type, HTTP encoding or by *wxFileSystem* protocol. The available factories can be enumerated using *GetFirst()* and *GetNext()* (p. 644).

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/stream.h>

### Data structures

```
enum wxStreamProtocolType
{
    wxSTREAM_PROTOCOL, // wxFileSystem protocol (should be only one)
    wxSTREAM_MIMETYPE, // MIME types the stream handles
    wxSTREAM_ENCODING, // The HTTP Content-Encodings the stream
handles
    wxSTREAM_FILEEXT // File extensions the stream handles
};
```

### See also

*wxFilterInputStream* (p. 646)

*wxFilterOutputStream* (p. 647)

*wxArchiveClassFactory* (p. 58)

*Archive formats such as zip* (p. 2223)

### **wxFilterClassFactory::CanHandle**

**bool CanHandle(const wxChar\* protocol, wxStreamProtocolType type =  
wxSTREAM\_PROTOCOL) const**

Returns true if this factory can handle the given protocol, MIME type, HTTP encoding or file extension.

When using wxSTREAM\_FILEEXT for the second parameter, the first parameter can be a complete filename rather than just an extension.

### **wxFilterClassFactory::Find**

**static const wxFilterClassFactory\* Find(const wxChar\* protocol,  
wxStreamProtocolType type = wxSTREAM\_PROTOCOL)**

A static member that finds a factory that can handle a given protocol, MIME type, HTTP encoding or file extension. Returns a pointer to the class factory if found, or NULL otherwise. It does not give away ownership of the factory.

When using wxSTREAM\_FILEEXT for the second parameter, the first parameter can be a complete filename rather than just an extension.

### **wxFilterClassFactory::GetFirst/GetNext**

**static const wxFilterClassFactory\* GetFirst()**

**const wxFilterClassFactory\* GetNext() const**

GetFirst and GetNext can be used to enumerate the available factories.

For example, to list them:

```
wxString list;
const wxFilterClassFactory *factory =
wxFilterClassFactory::GetFirst();

while (factory) {
    list << factory->GetProtocol() << _T("\n");
    factory = factory->GetNext();
}
```

GetFirst()/GetNext() return a pointer to a factory or NULL if no more are available. They do not give away ownership of the factory.

### **wxFilterClassFactory::GetProtocol**

#### **wxString GetProtocol() const**

Returns the wxFileSystem protocol supported by this factory. Equivalent to wxString(\*GetProtocols()).

### **wxFilterClassFactory::GetProtocols**

**const wxChar \* const\* GetProtocols(wxStreamProtocolType type =  
wxSTREAM\_PROTOCOL) const**

Returns the protocols, MIME types, HTTP encodings or file extensions supported by this factory, as an array of null terminated strings. It does not give away ownership of the array or strings.

For example, to list the file extensions a factory supports:

```
wxString list;
const wxChar *const *p;

for (p = factory->GetProtocols(wxSTREAM_FILEEXT); *p; p++)
    list << *p << _T("\n");
```

### **wxFilterClassFactory::NewStream**

**wxFilterInputStream\* NewStream(wxInputStream& stream) const**

**wxFilterOutputStream\* NewStream(wxOutputStream& stream) const**

**wxFilterInputStream\* NewStream(wxInputStream\* stream) const**

**wxFilterOutputStream\* NewStream(wxOutputStream\* stream) const**

Create a new input or output stream to decompress or compress a given stream.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it.

If it is passed by reference then it does not.

### **wxFilterClassFactory::PopExtension**

**wxString PopExtension(const wxString& *location*) const**

Remove the file extension of *location* if it is one of the file extensions handled by this factory.

### **wxFilterClassFactory::PushFront**

**void PushFront()**

Adds this class factory to the list returned by *GetFirst()/GetNext()* (p. 644).

It is not necessary to do this to use the filter streams. It is usually used when implementing streams, typically the implementation will add a static instance of its factory class.

It can also be used to change the order of a factory already in the list, bringing it to the front. This isn't a thread safe operation so can't be done when other threads are running that will be using the list.

The list does not take ownership of the factory.

### **wxFilterClassFactory::Remove**

**void Remove()**

Removes this class factory from the list returned by *GetFirst()/GetNext()* (p. 644).

Removing from the list isn't a thread safe operation so can't be done when other threads are running that will be using the list.

The list does not own the factories, so removing a factory does not delete it.

## **wxFilterInputStream**

A filter stream has the capability of a normal stream but it can be placed on top of another stream. So, for example, it can uncompress or decrypt the data which are read from another stream and pass it to the requester.

### **Derived from**

*wxInputStream* (p. 941)

*wxStreamBase* (p. 1544)

### **Include files**

<wx/stream.h>

### **Note**

The interface of this class is the same as that of `wxInputStream`. Only a constructor differs and it is documented below.

**See also**

*wxFilterClassFactory* (p. 643)

*wxFilterOutputStream* (p. 647)

**wxFilterInputStream::wxFilterInputStream**

**wxFilterInputStream**(`wxInputStream& stream`)

**wxFilterInputStream**(`wxInputStream* stream`)

Initializes a "filter" stream.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

**wxFilterOutputStream**

A filter stream has the capability of a normal stream but it can be placed on top of another stream. So, for example, it can compress, encrypt the data which are passed to it and write them to another stream.

**Derived from**

*wxOutputStream* (p. 1156)

*wxStreamBase* (p. 1544)

**Include files**

<wx/stream.h>

**Note**

The use of this class is exactly the same as of `wxOutputStream`. Only a constructor differs and it is documented below.

**See also**

*wxFilterClassFactory* (p. 643)

*wxFilterInputStream* (p. 646)

**wxFilterOutputStream::wxFilterOutputStream**

**wxFilterOutputStream**(`wxOutputStream& stream`)

**wxFilterOutputStream(wxOutputStream\* stream)**

Initializes a "filter" stream.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

## wxFindDialogEvent

wxFindReplaceDialog events

### Derived from

*wxCommandEvent* (p. 250)

### Include files

<wx/fdrepldg.h>

### Event table macros

To process a command event from *wxFindReplaceDialog* (p. 651), use these event handler macros to direct input to member functions that take a *wxFindDialogEvent* argument. The *id* parameter is the identifier of the find dialog and you may usually specify -1 for it unless you plan to have several find dialogs sending events to the same owner window simultaneously.

<b>EVT_FIND(id, func)</b>	Find button was pressed in the dialog.
<b>EVT_FIND_NEXT(id, func)</b>	Find next button was pressed in the dialog.
<b>EVT_FIND_REPLACE(id, func)</b>	Replace button was pressed in the dialog.
<b>EVT_FIND_REPLACE_ALL(id, func)</b>	Replace all button was pressed in the dialog.
<b>EVT_FIND_CLOSE(id, func)</b>	The dialog is being destroyed, any pointers to it cannot be used any longer.

## wxFindDialogEvent::wxFindDialogEvent

**wxFindDialogEvent(wxEventType commandType = wxEVT\_NULL, int id = 0)**

Constructor used by wxWidgets only.

## wxFindDialogEvent::GetFlags

**int GetFlags() const**

Get the currently selected flags: this is the combination of *wxFR\_DOWN*, *wxFR\_WHOLEWORD* and *wxFR\_MATCHCASE* flags.



**wxFindDialogEvent::GetFindString****wxString GetFindString() const**

Return the string to find (never empty).

**wxFindDialogEvent::GetReplaceString****const wxString& GetReplaceString() const**

Return the string to replace the search string with (only for replace and replace all events).

**wxFindDialogEvent::GetDialog****wxFindReplaceDialog\* GetDialog() const**

Return the pointer to the dialog which generated this event.

**wxFindReplaceData**

`wxFindReplaceData` holds the data for `wxFindReplaceDialog` (p. 651). It is used to initialize the dialog with the default values and will keep the last values from the dialog when it is closed. It is also updated each time a `wxFindDialogEvent` (p. 648) is generated so instead of using the `wxFindDialogEvent` methods you can also directly query this object.

Note that all `SetXXX()` methods may only be called before showing the dialog and calling them has no effect later.

**Include files**

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

**Derived from**`wxObject` (p. 1148)**Data structures**

Flags used by `wxFindReplaceData::GetFlags()` (p. 650) and `wxFindDialogEvent::GetFlags()` (p. 648):

```
enum wxFindReplaceFlags
{
    // downward search/replace selected (otherwise - upwards)
    wxFR_DOWN          = 1,

    // whole word search/replace selected
    wxFR_WHOLEWORD     = 2,

    // case sensitive search/replace selected (otherwise - case
    insensitive)
```

```
        wxFR_MATCHCASE = 4  
    }
```

These flags can be specified in *wxFindReplaceDialog* ctor (p. 651) or *Create()* (p. 651):

```
enum wxFindReplaceDialogStyles  
{  
    // replace dialog (otherwise find dialog)  
    wxFR_REPLACEDIALOG = 1,  
  
    // don't allow changing the search direction  
    wxFR_NOUPDOWN      = 2,  
  
    // don't allow case sensitive searching  
    wxFR_NOMATCHCASE   = 4,  
  
    // don't allow whole word searching  
    wxFR_NOWHOLEWORD   = 8  
}
```

### **wxFindReplaceData::wxFindReplaceData**

**wxFindReplaceData(wxUint32 flags = 0)**

Constructor initializes the flags to default value (0).

### **wxFindReplaceData::GetFindString**

**const wxString& GetFindString()**

Get the string to find.

### **wxFindReplaceData::GetReplaceString**

**const wxString& GetReplaceString()**

Get the replacement string.

### **wxFindReplaceData::GetFlags**

**int GetFlags() const**

Get the combination of *wxFindReplaceFlags* values.

### **wxFindReplaceData::SetFlags**

**void SetFlags(wxUint32 flags)**

Set the flags to use to initialize the controls of the dialog.

**wxFindReplaceData::SetFindString****void SetFindString(const wxString& str)**

Set the string to find (used as initial value by the dialog).

**wxFindReplaceData::SetReplaceString****void SetReplaceString(const wxString& str)**

Set the replacement string (used as initial value by the dialog).

**wxFindReplaceDialog**

`wxFindReplaceDialog` is a standard modeless dialog which is used to allow the user to search for some text (and possibly replace it with something else). The actual searching is supposed to be done in the owner window which is the parent of this dialog. Note that it means that unlike for the other standard dialogs this one **must** have a parent window. Also note that there is no way to use this dialog in a modal way; it is always, by design and implementation, modeless.

Please see the dialogs sample for an example of using it.

**Include files**

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

**Derived from**

`wxDialog` (p. 496)

**wxFindReplaceDialog::wxFindReplaceDialog****wxFindReplaceDialog()****wxFindReplaceDialog(wxWindow \* parent, wxFindReplaceData\* data, const wxString& title, int style = 0)**

After using default constructor `Create()` (p. 651) must be called.

The *parent* and *data* parameters must be non-NULL.

**wxFindReplaceDialog::~wxFindReplaceDialog****~wxFindReplaceDialog()**

Destructor.

**wxFindReplaceDialog::Create**

```
bool Create(wxWindow * parent, wxFindReplaceData* data, const wxString& title, int
style = 0)
```

Creates the dialog; use *Show* (p. 1850) to show it on screen.

**The *parent* and *data* parameters must be non-NULL.**  
**`wxFindReplaceDialog::GetData`**

```
const wxFindReplaceData* GetData() const
```

Get the *wxFindReplaceData* (p. 649) object used by this dialog.

## **wxFlexGridSizer**

A flex grid sizer is a sizer which lays out its children in a two-dimensional table with all table fields in one row having the same height and all fields in one column having the same width, but all rows or all columns are not necessarily the same height or width as in the *wxGridSizer* (p. 797).

Since *wxWidgets* 2.5.0, *wxFlexGridSizer* can also size items equally in one direction but unequally ("flexibly") in the other. If the sizer is only flexible in one direction (this can be changed using *SetFlexibleDirection* (p. 654)), it needs to be decided how the sizer should grow in the other ("non-flexible") direction in order to fill the available space. The *SetNonFlexibleGrowMode* (p. 654) method serves this purpose.

### **Derived from**

*wxGridSizer* (p. 797)

*wxSizer* (p. 1444)

*wxObject* (p. 1148)

### **Include files**

<wx/sizer.h>

### **See also**

*wxSizer* (p. 1444), *Sizer overview* (p. 2098)

## **wxFlexGridSizer::wxFlexGridSizer**

```
wxFlexGridSizer(int rows, int cols, int vgap, int hgap)
```

```
wxFlexGridSizer(int cols, int vgap = 0, int hgap = 0)
```

Constructor for a *wxGridSizer*. *rows* and *cols* determine the number of columns and rows in the sizer - if either of the parameters is zero, it will be calculated to form the total number of children in the sizer, thus making the sizer grow dynamically. *vgap* and *hgap* define extra space between all children.

**wxFlexGridSizer::AddGrowableCol****void AddGrowableCol(size\_t idx, int proportion = 0)**

Specifies that column *idx* (starting from zero) should be grown if there is extra space available to the sizer.

The *proportion* parameter has the same meaning as the stretch factor for the *sizers* (p. 2098) except that if all proportions are 0, then all columns are resized equally (instead of not being resized at all).

**wxFlexGridSizer::AddGrowableRow****void AddGrowableRow(size\_t idx, int proportion = 0)**

Specifies that row *idx* (starting from zero) should be grown if there is extra space available to the sizer.

See *AddGrowableCol* (p. 653) for the description of *proportion* parameter.

**wxFlexGridSizer::GetFlexibleDirection****int GetFlexibleDirection() const**

Returns a *wxOrientation* value that specifies whether the sizer flexibly resizes its columns, rows, or both (default).

**Return value**

One of the following values:

<i>wxVERTICAL</i>	Rows are flexibly sized.
<i>wxHORIZONTAL</i>	Columns are flexibly sized.
<i>wxBOTH</i>	Both rows and columns are flexibly sized (this is the default value).

**See also**

*SetFlexibleDirection* (p. 654)

**wxFlexGridSizer::GetNonFlexibleGrowMode****int GetNonFlexibleGrowMode() const**

Returns the value that specifies how the sizer grows in the "non-flexible" direction if there is one.

**Return value**

One of the following values:

<code>wxFLEX_GROWMODE_NONE</code>	Sizer doesn't grow in the non-flexible direction.
<code>wxFLEX_GROWMODE_SPECIFIED</code>	Sizer honors growable columns/rows set with <i>AddGrowCol</i> (p. 653) and <i>AddGrowRow</i> (p. 653). In this case equal sizing applies to minimum sizes of columns or rows (this is the default value).
<code>wxFLEX_GROWMODE_ALL</code>	Sizer equally stretches all columns or rows in the non-flexible direction, whether they are growable or not in the flexible direction.

**See also**

*SetFlexibleDirection* (p. 654), *SetNonFlexibleGrowMode* (p. 654)

**`wxFlexGridSizer::RemoveGrowCol`**

**`void RemoveGrowCol(size_t idx)`**

Specifies that column `idx` is no longer growable.

**`wxFlexGridSizer::RemoveGrowRow`**

**`void RemoveGrowRow(size_t idx)`**

Specifies that row `idx` is no longer growable.

**`wxFlexGridSizer::SetFlexibleDirection`**

**`void SetFlexibleDirection(int direction)`**

Specifies whether the sizer should flexibly resize its columns, rows, or both. Argument `direction` can be `wxVERTICAL`, `wxHORIZONTAL` or `wxBOTH` (which is the default value). Any other value is ignored. See *GetFlexibleDirection()* (p. 653) for the explanation of these values.

Note that this method does not trigger relayout.

**`wxFlexGridSizer::SetNonFlexibleGrowMode`**

**`void SetNonFlexibleGrowMode(wxFlexSizerGrowMode mode)`**

Specifies how the sizer should grow in the non-flexible direction if there is one (so *SetFlexibleDirection()* (p. 654) must have been called previously). Argument `mode` can be one of those documented in *GetNonFlexibleGrowMode* (p. 653), please see there for their explanation.

Note that this method does not trigger relayout.

## wxFocusEvent

A focus event is sent when a window's focus changes. The window losing focus receives a "kill focus" event while the window gaining it gets a "set focus" one.

Notice that the set focus event happens both when the user gives focus to the window (whether using the mouse or keyboard) and when it is done from the program itself using *SetFocus* (p. 1839).

### Derived from

*wxEvt* (p. 572)

*wxObject* (p. 1148)

### Include files

<wx/event.h>

### Event table macros

To process a focus event, use these event handler macros to direct input to a member function that takes a *wxFocusEvent* argument.

**EVT\_SET\_FOCUS(func)**                      Process a *wxEVT\_SET\_FOCUS* event.

**EVT\_KILL\_FOCUS(func)**                      Process a *wxEVT\_KILL\_FOCUS* event.

### See also

*Event handling overview* (p. 2077)

## *wxFocusEvent::wxFocusEvent*

**wxFocusEvent(WXTYPE eventType = 0, int id = 0)**

Constructor.

## *wxFocusEvent::GetWindow*

Returns the window associated with this event, that is the window which had the focus before for the *wxEVT\_SET\_FOCUS* event and the window which is going to receive focus for the *wxEVT\_KILL\_FOCUS* one.

Warning: the window pointer may be *NULL*!

## wxFont

A font is an object which determines the appearance of text. Fonts are used for drawing text to a device context, and setting the appearance of a window's text.

This class uses *reference counting and copy-on-write* (p. 2046) internally so that assignments between two instances of this class are very cheap. You can therefore use actual objects instead of pointers without efficiency problems. If an instance of this class is changed it will create its own data internally so that other instances, which previously shared the data using the reference counting, are not affected.

You can retrieve the current system font settings with *wxSystemSettings* (p. 1594).

*wxSystemSettings* (p. 1594)

### Derived from

*wxGDIObject* (p. 709)

*wxObject* (p. 1148)

### Include files

<wx/font.h>

### Constants

The possible values for the *family* parameter of *wxFont constructor* (p. 658) are (the old names are for compatibility only):

```
enum wxFontFamily
{
    wxFONTFAMILY_DEFAULT = wxDEFAULT,
    wxFONTFAMILY_DECORATIVE = wxDECORATIVE,
    wxFONTFAMILY_ROMAN = wxROMAN,
    wxFONTFAMILY_SCRIPT = wxSCRIPT,
    wxFONTFAMILY_SWISS = wxSWISS,
    wxFONTFAMILY_MODERN = wxMODERN,
    wxFONTFAMILY_TELETYPE = wxTELETYPE,
    wxFONTFAMILY_MAX
};
```

The possible values for the *weight* parameter are (the old names are for compatibility only):

```
enum wxFontWeight
{
    wxFONTWEIGHT_NORMAL = wxNORMAL,
    wxFONTWEIGHT_LIGHT = wxLIGHT,
    wxFONTWEIGHT_BOLD = wxBOLD,
    wxFONTWEIGHT_MAX
};
```

The font flags which can be used during the font creation are:

```
enum
{
    // no special flags: font with default weight/slant/anti-aliasing
    wxFONTFLAG_DEFAULT = 0,

    // slant flags (default: no slant)
    wxFONTFLAG_ITALIC = 1 << 0,
```



```
wxFONTFLAG_SLANT          = 1 << 1,

// weight flags (default: medium)
wxFONTFLAG_LIGHT          = 1 << 2,
wxFONTFLAG_BOLD           = 1 << 3,

// anti-aliasing flag: force on or off (default: the current system
default)
wxFONTFLAG_ANTI_ALIASSED  = 1 << 4,
wxFONTFLAG_NOT_ANTI_ALIASSED = 1 << 5,

// underlined/strikethrough flags (default: no lines)
wxFONTFLAG_UNDERLINED     = 1 << 6,
wxFONTFLAG_STRIKETHROUGH  = 1 << 7,
};
```

The known font encodings are:

```
enum wxFontEncoding
{
    wxFONTENCODING_SYSTEM = -1,          // system default
    wxFONTENCODING_DEFAULT,              // current default encoding

    // ISO8859 standard defines a number of single-byte charsets
    wxFONTENCODING_ISO8859_1,            // West European (Latin1)
    wxFONTENCODING_ISO8859_2,            // Central and East European
    (Latin2)
    wxFONTENCODING_ISO8859_3,            // Esperanto (Latin3)
    wxFONTENCODING_ISO8859_4,            // Baltic (old) (Latin4)
    wxFONTENCODING_ISO8859_5,            // Cyrillic
    wxFONTENCODING_ISO8859_6,            // Arabic
    wxFONTENCODING_ISO8859_7,            // Greek
    wxFONTENCODING_ISO8859_8,            // Hebrew
    wxFONTENCODING_ISO8859_9,            // Turkish (Latin5)
    wxFONTENCODING_ISO8859_10,           // Variation of Latin4 (Latin6)
    wxFONTENCODING_ISO8859_11,           // Thai
    wxFONTENCODING_ISO8859_12,           // doesn't exist currently, but
    put it                               // here anyhow to make all
    ISO8859                              // consecutive numbers

    wxFONTENCODING_ISO8859_13,           // Baltic (Latin7)
    wxFONTENCODING_ISO8859_14,           // Latin8
    wxFONTENCODING_ISO8859_15,           // Latin9 (a.k.a. Latin0, includes
    euro)
    wxFONTENCODING_ISO8859_MAX,

    // Cyrillic charset soup (see
    http://czyborra.com/charsets/cyrillic.html)
    wxFONTENCODING_KOI8,                 // we don't support any of KOI8
    variants
    wxFONTENCODING_ALTERNATIVE,          // same as MS-DOS CP866
    wxFONTENCODING_BULGARIAN,            // used under Linux in Bulgaria

    // what would we do without Microsoft? They have their own
    encodings
    // for DOS
```

```
        wxFONTENCODING_CP437,           // original MS-DOS codepage
        wxFONTENCODING_CP850,           // CP437 merged with Latin1
        wxFONTENCODING_CP852,           // CP437 merged with Latin2
        wxFONTENCODING_CP855,           // another cyrillic encoding
        wxFONTENCODING_CP866,           // and another one
        // and for Windows
        wxFONTENCODING_CP874,           // WinThai
        wxFONTENCODING_CP1250,          // WinLatin2
        wxFONTENCODING_CP1251,          // WinCyrillic
        wxFONTENCODING_CP1252,          // WinLatin1
        wxFONTENCODING_CP1253,          // WinGreek (8859-7)
        wxFONTENCODING_CP1254,          // WinTurkish
        wxFONTENCODING_CP1255,          // WinHebrew
        wxFONTENCODING_CP1256,          // WinArabic
        wxFONTENCODING_CP1257,          // WinBaltic (same as Latin 7)
        wxFONTENCODING_CP12_MAX,

        wxFONTENCODING_UTF7,            // UTF-7 Unicode encoding
        wxFONTENCODING_UTF8,            // UTF-8 Unicode encoding

        wxFONTENCODING_UNICODE,         // Unicode - currently used only
    by                                     // wxEncodingConverter class

        wxFONTENCODING_MAX
};
```

## Predefined objects

Objects:

### **wxNullFont**

Pointers:

**wxNORMAL\_FONT**  
**wxSMALL\_FONT**  
**wxITALIC\_FONT**  
**wxSWISS\_FONT**

### See also

*wxFont* overview (p. 2121), *wxDC::SetFont* (p. 473), *wxDC::DrawText* (p. 464),  
*wxDC::GetTextExtent* (p. 469), *wxFontDialog* (p. 670), *wxSystemSettings* (p. 1594)

## **wxFont::wxFont**

### **wxFont()**

Default constructor.

### **wxFont(const wxFont& font)**

Copy constructor, uses *reference counting* (p. 2046).

```
wxFont(int pointSize, wxFontFamily family, int style, wxFontWeight weight, const  
bool underline = false, const wxString& faceName = "", wxFontEncoding encoding =  
wxFONTENCODING_DEFAULT)
```

```
wxFont(const wxSize& pixelSize, wxFontFamily family, int style, wxFontWeight  
weight, const bool underline = false, const wxString& faceName = "",  
wxFontEncoding encoding = wxFONTENCODING_DEFAULT)
```

Creates a font object with the specified attributes.

### Parameters

*pointSize*

Size in points.

*pixelSize*

Size in pixels: this is directly supported only under MSW currently where this constructor can be used directly, under other platforms a font with the closest size to the given one is found using binary search and the static *New* (p. 663) method must be used.

*family*

Font family, a generic way of referring to fonts without specifying actual facename. One of:

**wxFONTFAMILY\_DEFAULT** Chooses a default font.

**wxFONTFAMILY\_DECORATIVE** A decorative font.

**wxFONTFAMILY\_ROMAN** A formal, serif font.

**wxFONTFAMILY\_SCRIPT** A handwriting font.

**wxFONTFAMILY\_SWISS** A sans-serif font.

**wxFONTFAMILY\_MODERN** A fixed pitch font.

**wxFONTFAMILY\_TELETYPE** A teletype font.

*style*

One of **wxFONTSTYLE\_NORMAL**, **wxFONTSTYLE\_SLANT** and **wxFONTSTYLE\_ITALIC**.

*weight*

Font weight, sometimes also referred to as font boldness. One of:

**wxFONTWEIGHT\_NORMAL** Normal font.

**wxFONTWEIGHT\_LIGHT** Light font.

**wxFONTWEIGHT\_BOLD** Bold font.

#### *underline*

The value can be true or false. At present this has an effect on Windows and Motif 2.x only.

#### *faceName*

An optional string specifying the actual typeface to be used. If it is an empty string, a default typeface will be chosen based on the family.

#### *encoding*

An encoding which may be one of **wxFONTENCODING\_SYSTEM** Default system encoding.

**wxFONTENCODING\_DEFAULT** Default application encoding: this is the encoding set by calls to *SetDefaultEncoding* (p. 663) and which may be set to, say, KOI8 to create all fonts by default with KOI8 encoding. Initially, the default application encoding is the same as default system encoding.

**wxFONTENCODING\_ISO8859\_1...15** ISO8859 encodings.

**wxFONTENCODING\_KOI8** The standard Russian encoding for Internet.

**wxFONTENCODING\_CP1250...1252** Windows encodings similar to ISO8859 (but not identical).

If the specified encoding isn't available, no font is created (see also *font encoding overview* (p. 2122)).

#### **Remarks**

If the desired font does not exist, the closest match will be chosen. Under Windows, only scalable TrueType fonts are used.

See also *wxDC::SetFont* (p. 473), *wxDC::DrawText* (p. 464) and *wxDC::GetTextExtent* (p. 469).

#### **wxFont::~~wxFont**

##### **~wxFont()**

Destructor. See *reference-counted object destruction* (p. 2046) for more info.

#### **Remarks**

Although all remaining fonts are deleted when the application exits, the application should try to clean up all fonts itself. This is because wxWidgets cannot know if a pointer to the

font object is stored in an application data structure, and there is a risk of double deletion.

### **wxFont::IsFixedWidth**

#### **bool IsFixedWidth() const**

Returns `true` if the font is a fixed width (or monospaced) font, `false` if it is a proportional one or font is invalid.

### **wxFont::GetDefaultEncoding**

#### **static wxFontEncoding GetDefaultEncoding()**

Returns the current application's default encoding.

#### **See also**

*Font encoding overview* (p. 2122), *SetDefaultEncoding* (p. 663)

### **wxFont::GetFaceName**

#### **wxString GetFaceName() const**

Returns the typeface name associated with the font, or the empty string if there is no typeface information.

#### **See also**

*wxFont::SetFaceName* (p. 663)

### **wxFont::GetFamily**

#### **wxFontFamily GetFamily() const**

Gets the font family. See *wxFont::SetFamily* (p. 664) for a list of valid family identifiers.

#### **See also**

*wxFont::SetFamily* (p. 664)

### **wxFont::GetNativeFontInfoDesc**

#### **wxString GetNativeFontInfoDesc() const**

Returns the platform-dependent string completely describing this font. Returned string is always non-empty. Note that the returned string is not meant to be shown or edited by the user: a typical use of this function is for serializing in string-form a `wxFont` object.

#### **See also**

*wxFont::SetNativeFontInfo* (p. 664), *wxFont::GetNativeFontInfoUserDesc* (p. 662)

**wxFont::GetNativeFontInfoUserDesc****wxString GetNativeFontInfoUserDesc()**

Returns a user-friendly string for this font object. Returned string is always non-empty. Some examples of the formats of returned strings (which are platform-dependent) are in *SetNativeFontInfoUserDesc* (p. 665).

**See also**

*wxFont::GetNativeFontInfoDesc* (p. 661)

**wxFont::GetPointSize****int GetPointSize() const**

Gets the point size.

**See also**

*wxFont::SetPointSize* (p. 666)

**wxFont::GetStyle****int GetStyle() const**

Gets the font style. See *wxFont::wxFont* (p. 658) for a list of valid styles.

**See also**

*wxFont::SetStyle* (p. 666)

**wxFont::GetUnderlined****bool GetUnderlined() const**

Returns true if the font is underlined, false otherwise.

**See also**

*wxFont::SetUnderlined* (p. 667)

**wxFont::GetWeight****wxFontWeight GetWeight() const**

Gets the font weight. See *wxFont::wxFont* (p. 658) for a list of valid weight identifiers.

**See also**

*wxFont::SetWeight* (p. 667)

**wxFont::New**

```
static wxFont * New(int pointSize, wxFontFamily family, int style, wxFontWeight weight, const bool underline = false, const wxString& faceName = "", wxFontEncoding encoding = wxFONTENCODING_DEFAULT)
```

```
static wxFont * New(int pointSize, wxFontFamily family, int flags = wxFONTFLAG_DEFAULT, const wxString& faceName = "", wxFontEncoding encoding = wxFONTENCODING_DEFAULT)
```

```
static wxFont * New(const wxSize& pixelSize, wxFontFamily family, int style, wxFontWeight weight, const bool underline = false, const wxString& faceName = "", wxFontEncoding encoding = wxFONTENCODING_DEFAULT)
```

```
static wxFont * New(const wxSize& pixelSize, wxFontFamily family, int flags = wxFONTFLAG_DEFAULT, const wxString& faceName = "", wxFontEncoding encoding = wxFONTENCODING_DEFAULT)
```

These functions take the same parameters as *wxFont constructor* (p. 658) and return a new font object allocated on the heap.

Using `New( )` is currently the only way to directly create a font with the given size in pixels on platforms other than wxMSW.

**wxFont::IsOk**

```
bool IsOk() const
```

Returns `true` if this object is a valid font, `false` otherwise.

**wxFont::SetDefaultEncoding**

```
static void SetDefaultEncoding(wxFontEncoding encoding)
```

Sets the default font encoding.

**See also**

*Font encoding overview* (p. 2122), *GetDefaultEncoding* (p. 661)

**wxFont::SetFaceName**

```
bool SetFaceName(const wxString& faceName)
```

Sets the facename for the font. Returns `true` if the given face name exists; `false` otherwise.

**Parameters**

*faceName*

A valid facename, which should be on the end-user's system.

**Remarks**

To avoid portability problems, don't rely on a specific face, but specify the font family instead or as well. A suitable font will be found on the end-user's system. If both the family and the facename are specified, `wxWidgets` will first search for the specific face, and then for a font belonging to the same family.

**See also**

`wxFont::GetFaceName` (p. 661), `wxFont::SetFamily` (p. 664)

**wxFont::SetFamily**

**void SetFamily(wxFontFamily family)**

Sets the font family.

**Parameters**

*family*

One of:

**wxFONTFAMILY\_DEFAULT** Chooses a default font.

**wxFONTFAMILY\_DECORATIVE** A decorative font.

**wxFONTFAMILY\_ROMAN** A formal, serif font.

**wxFONTFAMILY\_SCRIPT** A handwriting font.

**wxFONTFAMILY\_SWISS** A sans-serif font.

**wxFONTFAMILY\_MODERN** A fixed pitch font.

**wxFONTFAMILY\_TELETYPE** A teletype font.

**See also**

`wxFont::GetFamily` (p. 661), `wxFont::SetFaceName` (p. 663)

**wxFont::SetNativeFontInfo**

**bool SetNativeFontInfo(const wxString& info)**

Creates the font corresponding to the given native font description string and returns `true` if the creation was successful. `info` must have been previously returned by `GetNativeFontInfoDesc` (p. 661). If the string is invalid, font state is undefined (it becomes invalid in 2.8 but this shouldn't be relied on as the next `wxWidgets` version leaves it unchanged instead). This function is typically used for de-serializing a `wxFont` object previously saved in a string-form.

**See also**



*wxFont::SetNativeFontInfoUserDesc* (p. 665)

## **wxFont::SetNativeFontInfoUserDesc**

**bool SetNativeFontInfoUserDesc(const wxString& info)**

Creates the font corresponding to the given native font description string and returns `true` if the creation was successful. Unlike *SetNativeFontInfo* (p. 664), this function accepts strings which are user-friendly. Examples of accepted string formats are:

Generic syntax

E  
x  
a  
m  
p  
l  
e

on **wxGTK2**: [FACE-NAME] [bold] [oblique|italic] [POINTSIZ]E]

M  
o  
n  
o  
s  
p  
a  
c  
e

b  
o  
l  
d

1  
0

on **wxMSW**: [light|bold] [italic] [FACE-NAME] [POINTSIZ]E] [ENCODING]

T  
a  
h  
o  
m  
a

1  
0

on **wxMac**: FIXME

For more detailed information about the allowed syntaxes you can look at the documentation of the native API used for font-rendering (e.g. `pango_font_description_from_string` (<http://developer.gnome.org/doc/API/2.0/pango/pango-Fonts.html#pango-font-description-from-string>)).

**See also**

`wxFont::SetNativeFontInfo` (p. 664)

**wxFont::SetPointSize**

**void SetPointSize(int *pointSize*)**

Sets the point size.

**Parameters**

*pointSize*

Size in points.

**See also**

`wxFont::GetPointSize` (p. 662)

**wxFont::SetStyle**

**void SetStyle(int *style*)**

Sets the font style.

#### Parameters

*style*

One of **wxFONTSTYLE\_NORMAL**, **wxFONTSTYLE\_SLANT** and **wxFONTSTYLE\_ITALIC**.

#### See also

*wxFont::GetStyle* (p. 662)

#### **wxFont::SetUnderlined**

**void SetUnderlined(const bool *underlined*)**

Sets underlining.

#### Parameters

*underlining*

true to underline, false otherwise.

#### See also

*wxFont::GetUnderlined* (p. 662)

#### **wxFont::SetWeight**

**void SetWeight(wxFontWeight *weight*)**

Sets the font weight.

#### Parameters

*weight*

One of:

**wxFONTWEIGHT\_NORMAL** Normal font.

**wxFONTWEIGHT\_LIGHT** Light font.

**wxFONTWEIGHT\_BOLD** Bold font.

#### See also

*wxFont::GetWeight* (p. 662)

#### **wxFont::operator =**

**wxFont& operator =(const wxFont& font)**

Assignment operator, using *reference counting* (p. 2046).

**wxFont::operator ==**

**bool operator ==(const wxFont& font)**

Equality operator. See *reference-counted object comparison* (p. 2046) for more info.

**wxFont::operator !=**

**bool operator !=(const wxFont& font)**

Inequality operator. See *reference-counted object comparison* (p. 2046) for more info.

## wxFontData

*wxFontDialog* overview (p. 2129)

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

### Derived from

*wxObject* (p. 1148)

### Include files

<wx/cmndata.h>

### See also

*Overview* (p. 2129), *wxFont* (p. 655), *wxFontDialog* (p. 670)

## wxFontData::wxFontData

**wxFontData()**

Constructor. Initializes *fontColour* to black, *showHelp* to black, *allowSymbols* to true, *enableEffects* to true, *minSize* to 0 and *maxSize* to 0.

## wxFontData::EnableEffects

**void EnableEffects(bool enable)**

Enables or disables 'effects' under MS Windows or generic only. This refers to the controls for manipulating colour, strikethrough and underline properties.

The default value is true.

**wxFontData::GetAllowSymbols****bool GetAllowSymbols()**

Under MS Windows, returns a flag determining whether symbol fonts can be selected. Has no effect on other platforms.

The default value is true.

**wxFontData::GetColour****wxColour& GetColour()**

Gets the colour associated with the font dialog.

The default value is black.

**wxFontData::GetChosenFont****wxFont GetChosenFont()**

Gets the font chosen by the user if the user pressed OK (wxFontDialog::ShowModal returned wxID\_OK).

**wxFontData::GetEnableEffects****bool GetEnableEffects()**

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()**

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()**

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(const wxFont& font)**

Sets the font that will be returned to the user (for internal use only).

### **wxFontData::SetColour**

**void SetColour(const wxColour& colour)**

Sets the colour that will be used for the font foreground colour.

The default colour is black.

### **wxFontData::SetInitialFont**

**void SetInitialFont(const 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.

## **wxFontDialog**

This class represents the font chooser dialog.

**Derived from**

*wxDialog* (p. 496)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/fontdlg.h>

**See also**

*Overview* (p. 2129),  
*wxFontData* (p. 668),  
*wxGetFontFromUser* (p. 1938)

**wxFontDialog::wxFontDialog**

**wxFontDialog()**

**wxFontDialog(wxWindow\* parent)**

**wxFontDialog(wxWindow\* parent, const wxFontData& data)**

Constructor. Pass a parent window, and optionally the *font data* (p. 668) object to be used to initialize the dialog controls. If the default constructor is used, *Create()* (p. 671) must be called before the dialog can be shown.

**wxFontDialog::Create**

**bool Create(wxWindow\* parent)**

**bool Create(wxWindow\* parent, const wxFontData& data)**

Creates the dialog if it the *wxFontDialog* object had been initialized using the default constructor. Returns *true* on success and *false* if an error occurred.

**wxFontDialog::GetFontData**

**const wxFontData& GetFontData() const**

**wxFontData& GetFontData()**

Returns the *font data* (p. 668) associated with the font dialog.

**wxFontDialog::ShowModal**

**int ShowModal()**

Shows the dialog, returning `wxID_OK` if the user pressed Ok, and `wxID_CANCEL` otherwise.

If the user cancels the dialog (`ShowModal` returns `wxID_CANCEL`), no font will be created. If the user presses OK, a new `wxFont` will be created and stored in the font dialog's `wxFontData` structure.

## wxFontEnumerator

`wxFontEnumerator` enumerates either all available fonts on the system or only the ones with given attributes - either only fixed-width (suited for use in programs such as terminal emulators and the like) or the fonts available in the given *encoding* (p. 2122).

To do this, you just have to call one of `EnumerateXXX()` functions - either *EnumerateFacenames* (p. 672) or *EnumerateEncodings* (p. 673) and the corresponding callback (*OnFacename* (p. 673) or *OnFontEncoding* (p. 673)) will be called repeatedly until either all fonts satisfying the specified criteria are exhausted or the callback returns false.

### Virtual functions to override

Either *OnFacename* (p. 673) or *OnFontEncoding* (p. 673) should be overridden depending on whether you plan to call *EnumerateFacenames* (p. 672) or *EnumerateEncodings* (p. 673). Of course, if you call both of them, you should override both functions.

### Derived from

None

### Include files

<wx/fontenum.h>

### See also

*Font encoding overview* (p. 2122), *Font sample* (p. 2035), *wxFont* (p. 655), *wxFontMapper* (p. 674)

## wxFontEnumerator::EnumerateFacenames

**virtual bool EnumerateFacenames( wxFontEncoding encoding = wxFONTENCODING\_SYSTEM, bool fixedWidthOnly = false)**

Call *OnFacename* (p. 673) for each font which supports given encoding (only if it is not `wxFONTENCODING_SYSTEM`) and is of fixed width (if *fixedWidthOnly* is true).

Calling this function with default arguments will result in enumerating all fonts available on the system.



### **wxFontEnumerator::EnumerateEncodings**

**virtual bool EnumerateEncodings(const wxString& font = "")**

Call *OnFontEncoding* (p. 673) for each encoding supported by the given font - or for each encoding supported by at least some font if *font* is not specified.

### **wxFontEnumerator::GetEncodings**

**static wxArrayString GetEncodings(const wxString& facename = "")**

Return array of strings containing all encodings found by *EnumerateEncodings* (p. 673).

### **wxFontEnumerator::GetFacenames**

**static wxArrayString GetFacenames(wxFontEncoding encoding = wxFONTENCODING\_SYSTEM, bool fixedWidthOnly = false)**

Return array of strings containing all facenames found by *EnumerateFacenames* (p. 672).

### **wxFontEnumerator::IsValidFacename**

**static bool IsValidFacename(const wxString & facename)**

Returns `true` if the given string is valid face name, i.e. it's the face name of an installed font and it can safely be used with *wxFont::SetFaceName* (p. 663).

### **wxFontEnumerator::OnFacename**

**virtual bool OnFacename(const wxString& font)**

Called by *EnumerateFacenames* (p. 672) for each match. Return `true` to continue enumeration or `false` to stop it.

### **wxFontEnumerator::OnFontEncoding**

**virtual bool OnFontEncoding(const wxString& font, const wxString& encoding)**

Called by *EnumerateEncodings* (p. 673) for each match. Return `true` to continue enumeration or `false` to stop it.

## **wxFontList**

A font list is a list containing all fonts which have been created. There is only one instance of this class: **wxTheFontList**. Use this object to search for a previously created font of the desired type and create it if not already found. In some windowing systems, the font may be a scarce resource, so it is best to reuse old resources if possible. When an application finishes, all fonts will be deleted and their resources freed, eliminating the possibility of 'memory leaks'.

**Derived from**

*wxList* (p. 966)

*wxObject* (p. 1148)

**Include files**

<wx/gdicmn.h>

**See also**

*wxFont* (p. 655)

**wxFontList::wxFontList****wxFontList()**

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

**wxFontList::FindOrCreateFont**

**wxFont \* FindOrCreateFont(int point\_size, int family, int style, int weight, bool underline = false, const wxString& facename = NULL, wxFontEncoding encoding = wxFONTENCODING\_DEFAULT)**

Finds a font of the given specification, or creates one and adds it to the list. See the *wxFont constructor* (p. 658) for details of the arguments.

**wxFontMapper**

wxFontMapper manages user-definable correspondence between logical font names and the fonts present on the machine.

The default implementations of all functions will ask the user if they are not capable of finding the answer themselves and store the answer in a config file (configurable via SetConfigXXX functions). This behaviour may be disabled by giving the value of false to "interactive" parameter.

However, the functions will always consult the config file to allow the user-defined values override the default logic and there is no way to disable this - which shouldn't be ever needed because if "interactive" was never true, the config file is never created anyhow.

In case everything else fails (i.e. there is no record in config file and "interactive" is false or user denied to choose any replacement), the class queries *wxEncodingConverter* (p. 568) for "equivalent" encodings (e.g. iso8859-2 and cp1250) and tries them.

**Using wxFontMapper in conjunction with wxMBConv classes**

If you need to display text in encoding which is not available at host system (see

*IsEncodingAvailable* (p. 677)), you may use these two classes to find font in some similar encoding (see *GetAltForEncoding* (p. 676)) and convert the text to this encoding (*wxMBConv* classes (p. 2059)).

Following code snippet demonstrates it:

```
if (!wxFontMapper::Get()->IsEncodingAvailable(enc, facename))
{
    wxFontEncoding alternative;
    if (wxFontMapper::Get()->GetAltForEncoding(enc, &alternative,
                                                facename, false))
    {
        wxCSConv
        convFrom(wxFontMapper::Get()->GetEncodingName(enc));
        wxCSConv
        convTo(wxFontMapper::Get()->GetEncodingName(alternative));
        text = wxString(text.mb_str(convFrom), convTo);
    }
    else
        ...failure (or we may try iso8859-1/7bit ASCII)...
}
...display text...
```

### Derived from

No base class

### Include files

<wx/fontmap.h>

### See also

*wxEncodingConverter* (p. 568), *Writing non-English applications* (p. 2063)

## wxFontMapper::wxFontMapper

### wxFontMapper()

Default ctor.

### Note

The preferred way of creating a *wxFontMapper* instance is to call *wxFontMapper::Get* (p. 676).

## wxFontMapper::~~wxFontMapper

### ~wxFontMapper()

Virtual dtor for a base class.

### **wxFontMapper::CharsetToEncoding**

**wxFontEncoding** **CharsetToEncoding**(const wxString& charset, bool interactive = true)

Returns the encoding for the given charset (in the form of RFC 2046) or `wxFONTENCODING_SYSTEM` if couldn't decode it.

Be careful when using this function with *interactive* set to `true` (default value) as the function then may show a dialog box to the user which may lead to unexpected reentrancies and may also take a significantly longer time than a simple function call. For these reasons, it is almost always a bad idea to call this function from the event handlers for repeatedly generated events such as `EVT_PAINT`.

### **wxFontMapper::Get**

**static wxFontMapper \* Get()**

Get the current font mapper object. If there is no current object, creates one.

#### **See also**

*wxFontMapper::Set* (p. 678)

### **wxFontMapper::GetAllEncodingNames**

**static const wxChar\*\* GetAllEncodingNames**(wxFontEncoding encoding)

Returns the array of all possible names for the given encoding. The array is `NULL`-terminated. If it isn't empty, the first name in it is the canonical encoding name, i.e. the same string as returned by *GetEncodingName()* (p. 677).

### **wxFontMapper::GetAltForEncoding**

**bool GetAltForEncoding**(wxFontEncoding encoding, wxNativeEncodingInfo\* info, const wxString& facename = wxEmptyString, bool interactive = true)

**bool GetAltForEncoding**(wxFontEncoding encoding, wxFontEncoding\* alt\_encoding, const wxString& facename = wxEmptyString, bool interactive = true)

Find an alternative for the given encoding (which is supposed to not be available on this system). If successful, return true and fill info structure with the parameters required to create the font, otherwise return false.

The first form is for wxWidgets' internal use while the second one is better suitable for general use -- it returns `wxFontEncoding` which can consequently be passed to `wxFont` constructor.

### **wxFontMapper::GetEncoding**

**static wxFontEncoding GetEncoding**(size\_t n)

Returns the  $n$ -th supported encoding. Together with *GetSupportedEncodingsCount()* (p. 677) this method may be used to get all supported encodings.

### **wxFontMapper::GetEncodingDescription**

**static wxString GetEncodingDescription(wxFontEncoding encoding)**

Return user-readable string describing the given encoding.

### **wxFontMapper::GetEncodingFromName**

**static wxFontEncoding GetEncodingFromName(const wxString& encoding)**

Return the encoding corresponding to the given internal name. This function is the inverse of *GetEncodingName* (p. 677) and is intentionally less general than *CharsetToEncoding* (p. 676), i.e. it doesn't try to make any guesses nor ever asks the user. It is meant just as a way of restoring objects previously serialized using *GetEncodingName* (p. 677).

### **wxFontMapper::GetEncodingName**

**static wxString GetEncodingName(wxFontEncoding encoding)**

Return internal string identifier for the encoding (see also *GetEncodingDescription()* (p. 677))

#### **See also**

*GetEncodingFromName* (p. 677)

### **wxFontMapper::GetSupportedEncodingsCount**

**static size\_t GetSupportedEncodingsCount()**

Returns the number of the font encodings supported by this class. Together with *GetEncoding* (p. 676) this method may be used to get all supported encodings.

### **wxFontMapper::IsEncodingAvailable**

**bool IsEncodingAvailable(wxFontEncoding encoding, const wxString& facename = wxEmptyString)**

Check whether given encoding is available in given face or not. If no facename is given, find *any* font in this encoding.

### **wxFontMapper::SetDialogParent**

**void SetDialogParent(wxWindow\* parent)**

The parent window for modal dialogs.

**wxFontMapper::SetDialogTitle****void SetDialogTitle(const wxString& title)**

The title for the dialogs (note that default is quite reasonable).

**wxFontMapper::Set****static wxFontMapper \* Set(wxFontMapper \*mapper)**

Set the current font mapper object and return previous one (may be NULL). This method is only useful if you want to plug-in an alternative font mapper into wxWidgets.

**See also**

*wxFontMapper::Get* (p. 676)

**wxFontMapper::SetConfig****void SetConfig(wxConfigBase\* config)**

Set the config object to use (may be NULL to use default).

By default, the global one (from *wxConfigBase::Get()* will be used) and the default root path for the config settings is the string returned by *GetDefaultConfigPath()*.

**wxFontMapper::SetConfigPath****void SetConfigPath(const wxString& prefix)**

Set the root config path to use (should be an absolute path).

**wxFontPickerCtrl**

This control allows the user to select a font. The generic implementation is a button which brings up a *wxFontDialog* (p. 670) when clicked. Native implementation may differ but this is usually a (small) widget which give access to the font-chooser dialog. It is only available if *wxUSE\_FONTPICKERCTRL* is set to 1 (the default).

**Derived from**

*wxPickerBase* (p. 1183)

*wxControl* (p. 285)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

**Include files**

<wx/fontpicker.h>

## Window styles

**wxFNTP\_DEFAULT\_STYLE** The default style: `wxFNTP_FONTDESC_AS_LABEL | wxFNTP_USEFONT_FOR_LABEL`.

**wxFNTP\_USE\_TEXTCTRL** Creates a text control to the left of the picker button which is completely managed by the *wxFontPickerCtrl* (p. 678) and which can be used by the user to specify a font (see *SetSelectedFont* (p. 680)). The text control is automatically synchronized with button's value. Use functions defined in *wxPickerBase* (p. 1183) to modify the text control.

**wxFNTP\_FONTDESC\_AS\_LABEL** Keeps the label of the button updated with the fontface name and the font size. E.g. choosing "Times New Roman bold, italic with size 10" from the fontdialog, will update the label (overwriting any previous label) with the "Times New Roman, 10" text.

**wxFNTP\_USEFONT\_FOR\_LABEL** Uses the currently selected font to draw the label of the button.

## Event handling

To process a font picker event, use these event handler macros to direct input to member functions that take a *wxFontPickerEvent* (p. 681) argument.

**EVT\_FONTPICKER\_CHANGED(id, func)** The user changed the font selected in the control either using the button or using text control (see `wxFNTP_USE_TEXTCTRL`; note that in this case the event is fired only if the user's input is valid, i.e. recognizable).

## See also

*wxFontDialog* (p. 670),  
*wxFontPickerEvent* (p. 681)

## **wxFontPickerCtrl::wxFontPickerCtrl**

```
wxFontPickerCtrl(wxWindow *parent, wxWindowID id, const wxFont& font =  
wxNullFont, const wxPoint& pos = wxDefaultPosition, const wxSize& size =  
wxDefaultSize, long style = wxFNTP_DEFAULT_STYLE, const wxValidator& validator  
= wxDefaultValidator, const wxString& name = "fontpickerctrl")
```

Initializes the object and calls *Create* (p. 679) with all the parameters.

## **wxFontPickerCtrl::Create**

```
bool Create(wxWindow *parent, wxWindowID id, const wxFont& font = wxNullFont,  
const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long
```

```
style = wxFNTF_DEFAULT_STYLE, const wxValidator& validator =  
wxDefaultValidator, const wxString& name = "fontpickerctrl")
```

### Parameters

*parent*

Parent window, must not be non-NULL.

*id*

The identifier for the control.

*font*

The initial font shown in the control. If `wxNullFont` is given, the default font is used.

*pos*

Initial position.

*size*

Initial size.

*style*

The window style, see `wxFNTF_*` flags.

*validator*

Validator which can be used for additional date checks.

*name*

Control name.

### Return value

`true` if the control was successfully created or `false` if creation failed.

### **wxFontPickerCtrl::GetSelectedFont**

**wxFont GetSelectedFont() const**

Returns the currently selected font. Note that this function is completely different from `wxWindow::GetFont` (p. 1813).

### **wxFontPickerCtrl::SetSelectedFont**

**void SetSelectedFont(const wxFont &font)**

Sets the currently selected font. Note that this function is completely different from `wxWindow::SetFont` (p. 1840).



**wxFontPickerCtrl::GetMaxPointSize****unsigned int GetMaxPointSize() const**

Returns the maximum point size value allowed for the user-chosen font.

**wxFontPickerCtrl::SetMaxPointSize****void GetMaxPointSize(unsigned int *max*)**

Sets the maximum point size value allowed for the user-chosen font. The default value is 100. Note that big fonts can require a lot of memory and CPU time both for creation and for rendering; thus, specially because the user has the option to specify the fontsize through a text control (see `wxFNTF_USE_TEXTCTRL`), it's a good idea to put a limit to the maximum font size when huge fonts do not make much sense.

**wxFontPickerEvent**

This event class is used for the events generated by *wxFontPickerCtrl* (p. 678).

**Derived from***wxCommandEvent* (p. 250)*wxEvent* (p. 572)*wxObject* (p. 1148)**Include files**`<wx/fontpicker.h>`**Event handling**

To process input from a *wxFontPickerCtrl*, use one of these event handler macros to direct input to member function that take a *wxFontPickerEvent* (p. 681) argument:

**EVT\_FONTPICKER\_CHANGED(id, func)** Generated whenever the selected font changes.

**See also***wxFontPickerCtrl* (p. 678)**wxFontPickerEvent::wxFontPickerEvent****wxFontPickerEvent(wxObject \* *generator*, int *id*, const wxFont& *font*)**

The constructor is not normally used by the user code.

**wxFontPickerEvent::GetFont**

**wxFont GetFont() const**

Retrieve the font the user has just selected.

**wxFontPickerEvent::SetFont****void SetFont(const wxFont & f)**

Set the font associated with the event.

## wxFrame

A frame is a window whose size and position can (usually) be changed by the user. It usually has thick borders and a title bar, and can optionally contain a menu bar, toolbar and status bar. A frame can contain any window that is not a frame or dialog.

A frame that has a status bar and toolbar created via the `CreateStatusBar/CreateToolBar` functions manages these windows, and adjusts the value returned by `GetClientSize` to reflect the remaining size available to application windows.

**Derived from**

*wxTopLevelWindow* (p. 1713)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

**Include files**

<wx/frame.h>

**Window styles**

**wxDEFAULT\_FRAME\_STYLE** Defined as **wxMINIMIZE\_BOX | wxMAXIMIZE\_BOX | wxRESIZE\_BORDER | wxSYSTEM\_MENU | wxCAPTION | wxCLOSE\_BOX | wxCLIP\_CHILDREN**.

**wxICONIZE** Display the frame iconized (minimized). Windows only.

**wxCAPTION** Puts a caption on the frame.

**wxMINIMIZE** Identical to **wxICONIZE**. Windows only.

**wxMINIMIZE\_BOX** Displays a minimize box on the frame.

**wxMAXIMIZE** Displays the frame maximized. Windows only.

**wxMAXIMIZE\_BOX** Displays a maximize box on the frame.

**wxCLOSE\_BOX** Displays a close box on the frame.

**wxSTAY\_ON\_TOP** Stay on top of all other windows, see also

	<code>wxFRAME_FLOAT_ON_PARENT</code> .
<b><code>wxSYSTEM_MENU</code></b>	Displays a system menu.
<b><code>wxRESIZE_BORDER</code></b>	Displays a resizable border around the window.
<b><code>wxFRAME_TOOL_WINDOW</code></b>	Causes a frame with a small titlebar to be created; the frame does not appear in the taskbar under Windows or GTK+.
<b><code>wxFRAME_NO_TASKBAR</code></b>	Creates an otherwise normal frame but it does not appear in the taskbar under Windows or GTK+ (note that it will minimize to the desktop window under Windows which may seem strange to the users and thus it might be better to use this style only without <code>wxMINIMIZE_BOX</code> style). In <code>wxGTK</code> , the flag is respected only if GTK+ is at least version 2.2 and the window manager supports <code>_NET_WM_STATE_SKIP_TASKBAR</code> ( <a href="http://freedesktop.org/Standards/wm-spec/1.3/ar01s05.html">http://freedesktop.org/Standards/wm-spec/1.3/ar01s05.html</a> ) hint. Has no effect under other platforms.
<b><code>wxFRAME_FLOAT_ON_PARENT</code></b>	The frame will always be on top of its parent (unlike <code>wxSTAY_ON_TOP</code> ). A frame created with this style must have a non-NULL parent.
<b><code>wxFRAME_EX_CONTEXTHELP</code></b>	Under Windows, puts a query button on the caption. When pressed, Windows will go into a context-sensitive help mode and <code>wxWidgets</code> will send a <code>wxEVT_HELP</code> event if the user clicked on an application window. <i>Note</i> that this is an extended style and must be set by calling <code>SetExtraStyle</code> (p. 1839) before <code>Create</code> is called (two-step construction). You cannot use this style together with <code>wxMAXIMIZE_BOX</code> or <code>wxMINIMIZE_BOX</code> , so you should use <code>wxDEFAULT_FRAME_STYLE &amp; ~ (wxMINIMIZE_BOX   wxMAXIMIZE_BOX)</code> for the frames having this style (the dialogs don't have a minimize or a maximize box by default)
<b><code>wxFRAME_SHAPED</code></b>	Windows with this style are allowed to have their shape changed with the <code>SetShape</code> (p. 1719) method.
<b><code>wxFRAME_EX_METAL</code></b>	On Mac OS X, frames with this style will be shown with a metallic look. This is an <i>extra</i> style.

The default frame style is for normal, resizable frames. To create a frame which can not be resized by user, you may use the following combination of styles:

`wxDEFAULT_FRAME_STYLE & ~ (wxRESIZE_BORDER | wxRESIZE_BOX | wxMAXIMIZE_BOX)`. See also *window styles overview* (p. 2089).

### Default event processing

`wxFrame` processes the following events:

- wxEVT\_SIZE* (p. 1443)      If the frame has exactly one child window, not counting the status and toolbar, this child is resized to take the entire frame client area. If two or more windows are present, they should be laid out explicitly either by manually handling *wxEVT\_SIZE* or *usingsizers* (p. 2098)
- wxEVT\_MENU\_HIGHLIGHT* (p. 1098)      The default implementation displays the *help string* (p. 1101) associated with the selected item in the first pane of the status bar, if there is one.

### Remarks

An application should normally define an *wxCloseEvent* (p. 200) handler for the frame to respond to system close events, for example so that related data and subwindows can be cleaned up.

### See also

*wxMDIParentFrame* (p. 1051), *wxMDIChildFrame* (p. 1047), *wxMiniFrame* (p. 1113), *wxDialog* (p. 496)

## **wxFrame::wxFrame**

### **wxFrame()**

Default constructor.

**wxFrame(wxWindow\* parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT\_FRAME\_STYLE, const wxString& name = "frame")**

Constructor, creating the window.

### Parameters

*parent*

The window parent. This may be NULL. If it is non-NULL, the frame will always be displayed on top of the parent window on Windows.

*id*

The window identifier. It may take a value of -1 to indicate a default value.

*title*

The caption to be displayed on the frame's title bar.

*pos*

The window position. A value of (-1, -1) indicates a default position, chosen by either the windowing system or *wxWidgets*, depending on platform.

*size*

The window size. A value of (-1, -1) indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform.

*style*

The window style. See *wxFrame* (p. 682).

*name*

The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

**Remarks**

For Motif, MWM (the Motif Window Manager) should be running for any window styles to work (otherwise all styles take effect).

**See also**

*wxFrame::Create* (p. 685)

**wxFrame::~~wxFrame****void ~wxFrame()**

Destructor. Destroys all child windows and menu bar if present.

**wxFrame::Centre****void Centre(int direction = wxBOTH)**

Centres the frame on the display.

**Parameters***direction*

The parameter may be `wxHORIZONTAL`, `wxVERTICAL` or `wxBOTH`.

**wxFrame::Create**

```
bool Create(wxWindow* parent, wxWindowID id, const wxString& title, const  
wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxDEFAULT_FRAME_STYLE, const wxString& name = "frame")
```

Used in two-step frame construction. See *wxFrame::wxFrame* (p. 684) for further details.

**wxFrame::CreateStatusBar**

```
virtual wxStatusBar* CreateStatusBar(int number = 1, long style = wxST_SIZEGRIP |  
wxFULL_REPAINT_ON_RESIZE, wxWindowID id = 0, const wxString& name =
```

*"statusBar")*

Creates a status bar at the bottom of the frame.

### Parameters

*number*

The number of fields to create. Specify a value greater than 1 to create a multi-field status bar.

*style*

The status bar style. See *wxStatusBar* (p. 1536) for a list of valid styles.

*id*

The status bar window identifier. If -1, an identifier will be chosen by wxWidgets.

*name*

The status bar window name.

### Return value

A pointer to the status bar if it was created successfully, NULL otherwise.

### Remarks

The width of the status bar is the whole width of the frame (adjusted automatically when resizing), and the height and text size are chosen by the host windowing system.

By default, the status bar is an instance of *wxStatusBar*. To use a different class, override *wxFrame::OnCreateStatusBar* (p. 688).

Note that you can put controls and other windows on the status bar if you wish.

### See also

*wxFrame::SetStatusText* (p. 691), *wxFrame::OnCreateStatusBar* (p. 688),  
*wxFrame::GetStatusBar* (p. 687)

### **wxFrame::CreateToolBar**

**virtual wxToolBar\* CreateToolBar(long style = wxNO\_BORDER |  
wxTB\_HORIZONTAL, wxWindowID id = -1, const wxString& name = "toolBar")**

Creates a toolbar at the top or left of the frame.

### Parameters

*style*

The toolbar style. See *wxToolBar* (p. 1694) for a list of valid styles.

*id*

The toolbar window identifier. If -1, an identifier will be chosen by wxWidgets.

*name*

The toolbar window name.

### **Return value**

A pointer to the toolbar if it was created successfully, NULL otherwise.

### **Remarks**

By default, the toolbar is an instance of wxToolBar (which is defined to be a suitable toolbar class on each platform, such as wxToolBar95). To use a different class, override *wxFrame::OnCreateToolBar* (p. 689).

When a toolbar has been created with this function, or made known to the frame with *wxFrame::SetToolBar* (p. 692), the frame will manage the toolbar position and adjust the return value from *wxWindow::GetClientSize* (p. 1811) to reflect the available space for application windows.

Under Pocket PC, you should *always* use this function for creating the toolbar to be managed by the frame, so that wxWidgets can use a combined menubar and toolbar. Where you manage your own toolbars, create a wxToolBar as usual.

### **See also**

*wxFrame::CreateStatusBar* (p. 685), *wxFrame::OnCreateToolBar* (p. 689), *wxFrame::SetToolBar* (p. 692), *wxFrame::GetToolBar* (p. 688)

## **wxFrame::GetClientAreaOrigin**

**wxPoint GetClientAreaOrigin() const**

Returns the origin of the frame client area (in client coordinates). It may be different from (0, 0) if the frame has a toolbar.

## **wxFrame::GetMenuBar**

**wxMenuBar\* GetMenuBar() const**

Returns a pointer to the menubar currently associated with the frame (if any).

### **See also**

*wxFrame::SetMenuBar* (p. 690), *wxMenuBar* (p. 1088), *wxMenu* (p. 1074)

## **wxFrame::GetStatusBar**

**wxStatusBar\* GetStatusBar() const**

Returns a pointer to the status bar currently associated with the frame (if any).

**See also**

*wxFrame::CreateStatusBar* (p. 685), *wxStatusBar* (p. 1536)

**wxFrame::GetStatusBarPane**

**int GetStatusBarPane()**

Returns the status bar pane used to display menu and toolbar help.

**See also**

*wxFrame::SetStatusBarPane* (p. 690)

**wxFrame::GetToolBar**

**wxToolBar\* GetToolBar() const**

Returns a pointer to the toolbar currently associated with the frame (if any).

**See also**

*wxFrame::CreateToolBar* (p. 686), *wxToolBar* (p. 1694), *wxFrame::SetToolBar* (p. 692)

**wxFrame::OnCreateStatusBar**

**virtual wxStatusBar\* OnCreateStatusBar(int number, long style, wxWindowID id, const wxString& name)**

Virtual function called when a status bar is requested by *wxFrame::CreateStatusBar* (p. 685).

**Parameters**

*number*

The number of fields to create.

*style*

The window style. See *wxStatusBar* (p. 1536) for a list of valid styles.

*id*

The window identifier. If -1, an identifier will be chosen by wxWidgets.

*name*

The window name.

**Return value**



A status bar object.

**Remarks**

An application can override this function to return a different kind of status bar. The default implementation returns an instance of *wxStatusBar* (p. 1536).

**See also**

*wxFrame::CreateStatusBar* (p. 685), *wxStatusBar* (p. 1536).

**wxFrame::OnCreateToolBar**

**virtual wxToolBar\* OnCreateToolBar(long style, wxWindowID id, const wxString& name)**

Virtual function called when a toolbar is requested by *wxFrame::CreateToolBar* (p. 686).

**Parameters**

*style*

The toolbar style. See *wxToolBar* (p. 1694) for a list of valid styles.

*id*

The toolbar window identifier. If -1, an identifier will be chosen by wxWidgets.

*name*

The toolbar window name.

**Return value**

A toolbar object.

**Remarks**

An application can override this function to return a different kind of toolbar. The default implementation returns an instance of *wxToolBar* (p. 1694).

**See also**

*wxFrame::CreateToolBar* (p. 686), *wxToolBar* (p. 1694).

**wxFrame::ProcessCommand**

**void ProcessCommand(int id)**

Simulate a menu command.

**Parameters**

*id*

The identifier for a menu item.

### **wxFrame::SendSizeEvent**

#### **void SendSizeEvent()**

This function sends a dummy *size event* (p. 1443) to the frame forcing it to reevaluate its children positions. It is sometimes useful to call this function after adding or deleting a children after the frame creation or if a child size changes.

Note that if the frame is using either sizers or constraints for the children layout, it is enough to call *Layout()* (p. 1825) directly and this function should not be used in this case.

### **wxFrame::SetMenuBar**

#### **void SetMenuBar(wxMenuBar\* menuBar)**

Tells the frame to show the given menu bar.

#### **Parameters**

*menuBar*

The menu bar to associate with the frame.

#### **Remarks**

If the frame is destroyed, the menu bar and its menus will be destroyed also, so do not delete the menu bar explicitly (except by resetting the frame's menu bar to another frame or NULL).

Under Windows, a size event is generated, so be sure to initialize data members properly before calling **SetMenuBar**.

Note that on some platforms, it is not possible to call this function twice for the same frame object.

#### **See also**

*wxFrame::GetMenuBar* (p. 687), *wxMenuBar* (p. 1088), *wxMenu* (p. 1074).

### **wxFrame::SetStatusBar**

#### **void SetStatusBar(wxStatusBar\* statusBar)**

Associates a status bar with the frame.

#### **See also**

*wxFrame::CreateStatusBar* (p. 685), *wxStatusBar* (p. 1536), *wxFrame::GetStatusBar* (p. 687)

### **wxFrame::SetStatusBarPane**

**void SetStatusBarPane(int *n*)**

Set the status bar pane used to display menu and toolbar help. Using -1 disables help display.

**wxFrame::SetStatusText****virtual void SetStatusText(const wxString& *text*, int *number* = 0)**

Sets the status bar text and redraws the status bar.

**Parameters**

*text*

The text for the status field.

*number*

The status field (starting from zero).

**Remarks**

Use an empty string to clear the status bar.

**See also**

*wxFrame::CreateStatusBar* (p. 685), *wxStatusBar* (p. 1536)

**wxFrame::SetStatusWidths****virtual void SetStatusWidths(int *n*, int \**widths*)**

Sets the widths of the fields in the status bar.

**Parameters**

*n*

The number of fields in the status bar. It must be the same used in *CreateStatusBar* (p. 685).

*widths*

Must contain an array of *n* integers, each of which is a status field width in pixels. A value of -1 indicates that the field is variable width; at least one field must be -1. You should delete this array after calling **SetStatusWidths**.

**Remarks**

The widths of the variable fields are calculated from the total width of all fields, minus the sum of widths of the non-variable fields, divided by the number of variable fields.

**wxPython note:** Only a single parameter is required, a Python list of integers.

**wxPerl note:** In wxPerl this method takes the field widths as parameters.

### **wxFrame::SetToolBar**

**void SetToolBar**(wxToolBar\* *toolBar*)

Associates a toolbar with the frame.

#### **See also**

*wxFrame::CreateToolBar* (p. 686), *wxToolBar* (p. 1694), *wxFrame::GetToolBar* (p. 688)

## **wxFSFile**

This class represents a single file opened by *wxFileSystem* (p. 633). It provides more information than *wxWindow*'s input stream (stream, filename, mime type, anchor).

**Note:** Any pointer returned by a method of *wxFSFile* is valid only as long as the *wxFSFile* object exists. For example a call to *GetStream()* doesn't *create* the stream but only returns the pointer to it. In other words after 10 calls to *GetStream()* you will obtain ten identical pointers.

#### **Derived from**

*wxObject* (p. 1148)

#### **Include files**

<wx/filesys.h>

#### **See Also**

*wxFileSystemHandler* (p. 636), *wxFileSystem* (p. 633), *Overview* (p. 2075)

### **wxFSFile::wxFSFile**

**wxFSFile**(wxInputStream \**stream*, const wxString& *loc*, const wxString& *mimetype*, const wxString& *anchor*, wxDateTime *modif*)

Constructor. You probably won't use it. See Notes for details.

#### **Parameters**

*stream*

The input stream that will be used to access data

*location*

The full location (aka filename) of the file

**mimetype**

MIME type of this file. Mime type is either extension-based or HTTP Content-Type

**anchor**

Anchor. See *GetAnchor()* (p. 693) for details.

If you are not sure of the meaning of these params, see the description of the *GetXXXX()* functions.

**Notes**

It is seldom used by the application programmer but you will need it if you are writing your own virtual FS. For example you may need something similar to *wxMemoryInputStream*, but because *wxMemoryInputStream* doesn't free the memory when destroyed and thus passing a memory stream pointer into *wxFSFile* constructor would lead to memory leaks, you can write your own class derived from *wxFSFile*:

```
class wxMyFSFile : public wxFSFile
{
    private:
        void *m_Mem;
    public:
        wxMyFSFile(.....)
        ~wxMyFSFile() {free(m_Mem);}
        // of course dtor is virtual ;-)
};
```

**wxFSFile::DetachStream****void DetachStream()**

Detaches the stream from the *wxFSFile* object. That is, the stream obtained with *GetStream()* will continue its existence after the *wxFSFile* object is deleted. You will have to delete the stream yourself.

**wxFSFile::GetAnchor****const wxString& GetAnchor() const**

Returns anchor (if present). The term of **anchor** can be easily explained using few examples:

```
index.htm#anchor           /* 'anchor' is anchor */
index/wx001.htm           /* NO anchor here!    */
archive/main.zip#zip:index.htm#global /* 'global'          */
archive/main.zip#zip:index.htm /* NO anchor here!    */
```

Usually an anchor is presented only if the MIME type is 'text/html'. But it may have some meaning with other files; for example *myanim.avi#200* may refer to position in animation or *reality.wrl#MyView* may refer to a predefined view in VRML.

**wxFSFile::GetLocation****const wxString& GetLocation() const**

Returns full location of the file, including path and protocol. Examples :

```
http://www.wxwidgets.org
http://www.ms.mff.cuni.cz/~vsla8348/wxhtml/archive.zip#zip:info.txt
file:/home/vasek/index.htm
relative-file.htm
```

**wxFSFile::GetMimeType****const wxString& GetMimeType() const**

Returns the MIME type of the content of this file. It is either extension-based (see `wxMimeTypesManager`) or extracted from HTTP protocol Content-Type header.

**wxFSFile::GetModificationTime****wxDateTime GetModificationTime() const**

Returns time when this file was modified.

**wxFSFile::GetStream****wxInputStream\* GetStream() const**

Returns pointer to the stream. You can use the returned stream to directly access data. You may suppose that the stream provide `Seek` and `GetSize` functionality (even in the case of the HTTP protocol which doesn't provide this by default. `wxHtml` uses local cache to work around this and to speed up the connection).

**wxFTP**

`wxFTP` can be used to establish a connection to an FTP server and perform all the usual operations. Please consult the RFC 959 for more details about the FTP protocol.

To use a commands which doesn't involve file transfer (i.e. directory oriented commands) you just need to call a corresponding member function or use the generic *SendCommand* (p. 696) method. However to actually transfer files you just get or give a stream to or from this class and the actual data are read or written using the usual stream methods.

Example of using `wxFTP` for file downloading:

```
wxFTP ftp;

// if you don't use these lines anonymous login will be used
ftp.SetUser("user");
ftp.SetPassword("password");
```

```
if ( !ftp.Connect("ftp.wxwindows.org") )
{
    wxLogError("Couldn't connect");
    return;
}

ftp.ChDir("/pub");
wxInputStream *in =
ftp.GetInputStream("wxWidgets-4.2.0.tar.gz");
if ( !in )
{
    wxLogError("Coudln't get file");
}
else
{
    size_t size = in->GetSize();
    char *data = new char[size];
    if ( !in->Read(data, size) )
    {
        wxLogError("Read error");
    }
    else
    {
        // file data is in the buffer
        ...
    }

    delete [] data;
    delete in;
}
```

To upload a file you would do (assuming the connection to the server was opened successfully):

```
wxOutputStream *out = ftp.GetOutputStream("filename");
if ( out )
{
    out->Write(...); // your data
    delete out;
}
```

### Constants

wxFTP defines constants corresponding to the two supported transfer modes:

```
enum TransferMode
{
    ASCII,
    BINARY
};
```

### Derived from

*wxProtocol* (p. 1235)

**Include files**

<wx/protocol/ftp.h>

**See also**

*wxSocketBase* (p. 1472)

**wxFTP::wxFTP**

**wxFTP()**

Default constructor.

**wxFTP::~~wxFTP**

**~wxFTP()**

Destructor will close the connection if connected.

**wxFTP::Abort**

**bool Abort()**

Aborts the download currently in process, returns `true` if ok, `false` if an error occurred.

**wxFTP::CheckCommand**

**bool CheckCommand(const wxString& *command*, char *ret*)**

Send the specified *command* to the FTP server. *ret* specifies the expected result.

**Return value**

true if the command has been sent successfully, else false.

**wxFTP::SendCommand**

**char SendCommand(const wxString& *command*)**

Send the specified *command* to the FTP server and return the first character of the return code.

**wxFTP::GetLastResult**

**const wxString& GetLastResult()**

Returns the last command result, i.e. the full server reply for the last command.



**wxFTP::ChDir****bool ChDir(const wxString& *dir*)**

Change the current FTP working directory. Returns true if successful.

**wxFTP::MkDir****bool MkDir(const wxString& *dir*)**

Create the specified directory in the current FTP working directory. Returns true if successful.

**wxFTP::Rmdir****bool Rmdir(const wxString& *dir*)**

Remove the specified directory from the current FTP working directory. Returns true if successful.

**wxFTP::Pwd****wxString Pwd()**

Returns the current FTP working directory.

**wxFTP::Rename****bool Rename(const wxString& *src*, const wxString& *dst*)**

Rename the specified *src* element to *dst*. Returns true if successful.

**wxFTP::RmFile****bool RmFile(const wxString& *path*)**

Delete the file specified by *path*. Returns true if successful.

**wxFTP::SetAscii****bool SetAscii()**

Sets the transfer mode to ASCII. It will be used for the next transfer.

**wxFTP::SetBinary****bool SetBinary()**

Sets the transfer mode to binary (IMAGE). It will be used for the next transfer.

**wxFTP::SetPassive****void SetPassive**(bool *pasv*)

If *pasv* is `true`, passive connection to the FTP server is used. This is the default as it works with practically all firewalls. If the server doesn't support passive move, you may call this function with `false` argument to use active connection.

**wxFTP::SetTransferMode****bool SetTransferMode**(TransferMode *mode*)

Sets the transfer mode to the specified one. It will be used for the next transfer.

If this function is never called, binary transfer mode is used by default.

**wxFTP::SetUser****void SetUser**(const wxString& *user*)

Sets the user name to be sent to the FTP server to be allowed to log in.

**Default value**

The default value of the user name is "anonymous".

**Remark**

This parameter can be included in a URL if you want to use the URL manager. For example, you can use: "ftp://a\_user:a\_password@a.host:service/a\_directory/a\_file" to specify a user and a password.

**wxFTP::SetPassword****void SetPassword**(const wxString& *passwd*)

Sets the password to be sent to the FTP server to be allowed to log in.

**Default value**

The default value of the user name is your email address. For example, it could be "username@userhost.domain". This password is built by getting the current user name and the host name of the local machine from the system.

**Remark**

This parameter can be included in a URL if you want to use the URL manager. For example, you can use: "ftp://a\_user:a\_password@a.host:service/a\_directory/a\_file" to specify a user and a password.

**wxFTP::FileExists**

**bool FileExists(const wxString& filename)**

Returns `true` if the given remote file exists, `false` otherwise.

**wxFTP::GetFileSize****int GetFileSize(const wxString& filename)**

Returns the file size in bytes or -1 if the file doesn't exist or the size couldn't be determined. Notice that this size can be approximative size only and shouldn't be used for allocating the buffer in which the remote file is copied, for example.

**wxFTP::GetDirList****bool GetDirList(wxArrayString& files, const wxString& wildcard = "")**

The `GetList` function is quite low-level. It returns the list of the files in the current directory. The list can be filtered using the *wildcard* string. If *wildcard* is empty (default), it will return all files in directory.

The form of the list can change from one peer system to another. For example, for a UNIX peer system, it will look like this:

```
-r--r--r--  1 guilhem  lavaux      12738 Jan 16 20:17 cmndata.cpp
-r--r--r--  1 guilhem  lavaux      10866 Jan 24 16:41 config.cpp
-rw-rw-rw-  1 guilhem  lavaux      29967 Dec 21 19:17 cwlex.yy.c
-rw-rw-rw-  1 guilhem  lavaux      14342 Jan 22 19:51 cwy_tab.c
-r--r--r--  1 guilhem  lavaux      13890 Jan 29 19:18 date.cpp
-r--r--r--  1 guilhem  lavaux        3989 Feb  8 19:18 datstrm.cpp
```

But on Windows system, it will look like this:

```
winamp~1 exe      520196 02-25-1999  19:28  winamp204.exe
      1 file(s)                520 196 bytes
```

Return value: `true` if the file list was successfully retrieved, `false` otherwise.

**See also**

*GetFilesList* (p. 699)

**wxFTP::GetFilesList****bool GetFilesList(wxArrayString& files, const wxString& wildcard = "")**

This function returns the computer-parsable list of the files in the current directory (optionally only of the files matching the *wildcard*, all files by default). This list always has the same format and contains one full (including the directory path) file name per line.

Return value: `true` if the file list was successfully retrieved, `false` otherwise.

**wxFTP::GetOutputStream**

**wxOutputStream \* GetOutputStream(const wxString& file)**

Initializes an output stream to the specified *file*. The returned stream has all but the seek functionality of wxStreams. When the user finishes writing data, he has to delete the stream to close it.

**Return value**

An initialized write-only stream.

**See also**

*wxOutputStream* (p. 1156)

**wxFTP::GetInputStream****wxInputStream \* GetInputStream(const wxString& path)**

Creates a new input stream on the specified path. You can use all but the seek functionality of wxStream. Seek isn't available on all streams. For example, HTTP or FTP streams do not deal with it. Other functions like Tell are not available for this sort of stream, at present. You will be notified when the EOF is reached by an error.

**Return value**

Returns NULL if an error occurred (it could be a network failure or the fact that the file doesn't exist).

Returns the initialized stream. You will have to delete it yourself when you don't need it anymore. The destructor closes the DATA stream connection but will leave the COMMAND stream connection opened. It means that you can still send new commands without reconnecting.

**Example of a standalone connection (without wxURL)**

```
wxFTP ftp;
wxInputStream *in_stream;
char *data;

ftp.Connect("a.host.domain");
ftp.ChDir("a_directory");
in_stream = ftp.GetInputStream("a_file_to_get");

data = new char[in_stream->GetSize()];

in_stream->Read(data, in_stream->GetSize());
if (in_stream->LastError() != wxStream_NOERROR) {
    // Do something.
}

delete in_stream; /* Close the DATA connection */

ftp.Close(); /* Close the COMMAND connection */
```

**See also**

*wxInputStream* (p. 941)

## **wxGauge**

A gauge is a horizontal or vertical bar which shows a quantity (often time).

*wxGauge* supports two working modes: determinate and indeterminate progress.

The first is the usual working mode (see *SetValue* (p. 704) and *SetRange* (p. 704)) while the second can be used when the program is doing some processing but you don't know how much progress is being done. In this case, you can periodically call the *Pulse* (p. 705) function to make the progress bar switch to indeterminate mode (graphically it's usually a set of blocks which move or bounce in the bar control).

*wxGauge* supports dynamic switch between these two work modes.

There are no user commands for the gauge.

### **Derived from**

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### **Include files**

<wx/gauge.h>

### **Window styles**

<b>wxGA_HORIZONTAL</b>	Creates a horizontal gauge.
<b>wxGA_VERTICAL</b>	Creates a vertical gauge.
<b>wxGA_SMOOTH</b>	Creates smooth progress bar with one pixel wide update step (not supported by all platforms).

See also *window styles overview* (p. 2089).

### **Event handling**

*wxGauge* is read-only so generates no events.

### **See also**

*wxSlider* (p. 1462), *wxScrollBar* (p. 1408)

## **wxGauge::wxGauge**

**wxGauge()**

Default constructor.

```
wxGauge(wxWindow* parent, wxWindowID id, int range, const wxPoint& pos =  
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxGA_HORIZONTAL, const wxValidator& validator = wxDefaultValidator, const  
wxString& name = "gauge")
```

Constructor, creating and showing a gauge.

### Parameters

*parent*

Window parent.

*id*

Window identifier.

*range*

Integer range (maximum value) of the gauge. It is ignored when the gauge is used in indeterminate mode.

*pos*

Window position.

*size*

Window size.

*style*

Gauge style. See *wxGauge* (p. 701).

*name*

Window name.

### See also

*wxGauge::Create* (p. 702)

### **wxGauge::~wxGauge**

**~wxGauge()**

Destructor, destroying the gauge.

### **wxGauge::Create**

```
bool Create(wxWindow* parent, wxWindowID id, int range, const wxPoint& pos =  
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =
```

`wxGA_HORIZONTAL, const wxValidator& validator = wxDefaultValidator, const wxString& name = "gauge")`

Creates the gauge for two-step construction. See `wxGauge::wxGauge` (p. 701) for further details.

### **wxGauge::GetBezelFace**

**int GetBezelFace() const**

Returns the width of the 3D bezel face.

#### **Remarks**

This method is not implemented (returns 0) for most platforms.

#### **See also**

`wxGauge::SetBezelFace` (p. 704)

### **wxGauge::GetRange**

**int GetRange() const**

Returns the maximum position of the gauge.

#### **See also**

`wxGauge::SetRange` (p. 704)

### **wxGauge::GetShadowWidth**

**int GetShadowWidth() const**

Returns the 3D shadow margin width.

#### **Remarks**

This method is not implemented (returns 0) for most platforms.

#### **See also**

`wxGauge::SetShadowWidth` (p. 704)

### **wxGauge::GetValue**

**int GetValue() const**

Returns the current position of the gauge.

#### **See also**

`wxGauge::SetValue` (p. 704)

**wxGauge::IsVertical****bool IsVertical() const**

Returns `true` if the gauge is vertical (has `wxGA_VERTICAL` style) and `false` otherwise.

**wxGauge::SetBezelFace****void SetBezelFace(int width)**

Sets the 3D bezel face width.

**Remarks**

This method is not implemented (doesn't do anything) for most platforms.

**See also**

*wxGauge::GetBezelFace* (p. 703)

**wxGauge::SetRange****void SetRange(int range)**

Sets the range (maximum value) of the gauge. This function makes the gauge switch to determinate mode, if it's not already.

**See also**

*wxGauge::GetRange* (p. 703)

**wxGauge::SetShadowWidth****void SetShadowWidth(int width)**

Sets the 3D shadow width.

**Remarks**

This method is not implemented (doesn't do anything) for most platforms.

**wxGauge::SetValue****void SetValue(int pos)**

Sets the position of the gauge. This function makes the gauge switch to determinate mode, if it's not already.

**Parameters**

*pos*

Position for the gauge level.



**See also**

*wxGauge::GetValue* (p. 703)

**wxGauge::Pulse****void Pulse()**

Switch the gauge to indeterminate mode (if required) and makes the gauge move a bit to indicate the user that some progress has been made.

Note that after calling this function the value returned by *GetValue* (p. 703) is undefined and thus you need to explicitly call *SetValue* (p. 704) if you want to restore the determinate mode.

**wxGBPosition**

This class represents the position of an item in a virtual grid of rows and columns managed by a *wxGridBagSizer* (p. 772).

**Derived from**

No base class

**Include files**

<wx/gbsizer.h>

**wxGBPosition::wxGBPosition****wxGBPosition()****wxGBPosition(int row, int col)**

Construct a new *wxGBPosition*, optionally setting the row and column. The default is (0,0).

**wxGBPosition::GetCol****int GetCol() const**

Get the current column value.

**wxGBPosition::GetRow****int GetRow() const**

Get the current row value.

**wxGBPosition::SetCol**

**void SetCol(int col)**

Set a new column value.

**wxGBPosition::SetRow**

**void SetRow(int row)**

Set a new row value.

**wxGBPosition::operator!**

**bool operator!(const wxGBPosition& p) const**

Is the wxGBPosition valid? (An invalid wxGBPosition is (-1,-1). )

**wxGBPosition::operator==**

**bool operator operator==(const wxGBPosition& p) const**

Compare equality of two wxGBPositions.

## wxGBSizerItem

The wxGBSizerItem class is used by the *wxGridBagSizer* (p. 772) for tracking the items in the sizer. It adds grid position and spanning information to the normal *wxSizerItem* (p. 1458) by adding *wxGBPosition* (p. 705) and *wxGBSpan* (p. 708) attributes. Most of the time you will not need to use a wxGBSizerItem directly in your code, but there are a couple of cases where it is handy.

### Derived from

*wxSizerItem* (p. 1458)

### Include files

<wx/gbsizer.h>

### wxGBSizerItem::wxGBSizerItem

**wxGBSizerItem(int width, int height, const wxGBPosition& pos, const wxGBSpan& span, int flag, int border, wxObject\* userData)**

Construct a sizer item for tracking a spacer.

**wxGBSizerItem(wxWindow\* window, const wxGBPosition& pos, const wxGBSpan& span, int flag, int border, wxObject\* userData)**

Construct a sizer item for tracking a window.

**wxGBSizerItem**(*wxSizer\* sizer, const wxGBPosition& pos, const wxGBSpan& span, int flag, int border, wxObject\* userData*)

Construct a sizer item for tracking a subsizer.

### **wxGBSizerItem::GetEndPos**

**void GetEndPos**(*int& row, int& col*)

Get the row and column of the endpoint of this item

### **wxGBSizerItem::GetPos**

**wxGBPosition GetPos**() **const**

**void GetPos**(*int& row, int& col*) **const**

Get the grid position of the item.

### **wxGBSizerItem::GetSpan**

**wxGBSpan GetSpan**() **const**

**void GetSpan**(*int& rowspan, int& colspan*) **const**

Get the row and column spanning of the item.

### **wxGBSizerItem::Intersects**

**bool Intersects**(*const wxGBSizerItem& other*)

Returns true if this item and the other item intersect

**bool Intersects**(*const wxGBPosition& pos, const wxGBSpan& span*)

Returns true if the given pos/span would intersect with this item.

### **wxGBSizerItem::SetPos**

**bool SetPos**(*const wxGBPosition& pos*)

If the item is already a member of a sizer then first ensure that there is no other item that would intersect with this one at the new position, then set the new position. Returns true if the change is successful and after the next Layout the item will be moved.

### **wxGBSizerItem::SetSpan**

**bool SetSpan**(*const wxGBSpan& span*)

If the item is already a member of a sizer then first ensure that there is no other item that would intersect with this one with its new spanning size, then set the new spanning.

Returns true if the change is successful and after the next Layout the item will be resized.

## **wxGBSpan**

This class is used to hold the row and column spanning attributes of items in a *wxGridBagSizer* (p. 772).

### **Derived from**

No base class

### **Include files**

<wx/gbsizer.h>

### **wxGBSpan::wxGBSpan**

**wxGBSpan()**

**wxGBSpan(int rowspan, int colspan)**

Construct a new wxGBSpan, optionally setting the rowspan and colspan. The default is (1,1). (Meaning that the item occupies one cell in each direction.)

### **wxGBSpan::GetColspan**

**int GetColspan() const**

Get the current colspan value.

### **wxGBSpan::GetRowspan**

**int GetRowspan() const**

Get the current rowspan value.

### **wxGBSpan::SetColspan**

**void SetColspan(int colspan)**

Set a new colspan value.

### **wxGBSpan::SetRowspan**

**void SetRowspan(int rowspan)**

Set a new rowspan value.

**wxGBSpan::operator!****bool operator!(const wxGBSpan& o) const**

Is the wxGBSpan valid? (An invalid wxGBSpan is (-1,-1). )

**wxGBSpan::operator==****bool operator operator==(const wxGBSpan& o) const**

Compare equality of two wxGBSpans.

**wxGDIObject**

This class allows platforms to implement functionality to optimise GDI objects, such as wxPen, wxBrush and wxFont. On Windows, the underlying GDI objects are a scarce resource and are cleaned up when a usage count goes to zero. On some platforms this class may not have any special functionality.

Since the functionality of this class is platform-specific, it is not documented here in detail.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/gdiobj.h>

**See also**

*wxPen* (p. 1175), *wxBrush* (p. 150), *wxFont* (p. 655)

**wxGDIObject::wxGDIObject****wxGDIObject()**

Default constructor.

**wxGenericDirCtrl**

This control can be used to place a directory listing (with optional files) on an arbitrary window.

The control contains a *wxTreeCtrl* (p. 1728) window representing the directory hierarchy, and optionally, a *wxChoice* (p. 186) window containing a list of filters.

**Derived from**

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/dirctrl.h>

### Window styles

<code>wxDIRCTRL_DIR_ONLY</code>	Only show directories, and not files.
<code>wxDIRCTRL_3D_INTERNAL</code>	Use 3D borders for internal controls.
<code>wxDIRCTRL_SELECT_FIRST</code>	When setting the default path, select the first file in the directory.
<code>wxDIRCTRL_SHOW_FILTERS</code>	Show the drop-down filter list.
<code>wxDIRCTRL_EDIT_LABELS</code>	Allow the folder and file labels to be editable.

See also *Generic window styles* (p. 2089).

### Data structures

## **wxGenericDirCtrl::wxGenericDirCtrl**

### **wxGenericDirCtrl()**

Default constructor.

**wxGenericDirCtrl(wxWindow\* parent, const wxWindowID id = -1, const wxString& dir = wxDirDialogDefaultFolderStr, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDIRCTRL\_3D\_INTERNAL|wxSUNKEN\_BORDER, const wxString& filter = wxEmptyString, int defaultFilter = 0, const wxString& name = wxTreeCtrlNameStr)**

Main constructor.

### Parameters

*parent*

Parent window.

*id*

Window identifier.

*dir*

Initial folder.

*pos*

Position.

*size*

Size.

*style*

Window style. Please see *wxGenericDirCtrl* (p. 709) for a list of possible styles.

*filter*

A filter string, using the same syntax as that for *wxFileDialog* (p. 600). This may be empty if filters are not being used.

Example: "All files (\*.\*)|\*. \*|JPEG files (\*.jpg)|\*.jpg"

*defaultFilter*

The zero-indexed default filter setting.

*name*

The window name.

## **wxGenericDirCtrl::~~wxGenericDirCtrl**

**~wxGenericDirCtrl()**

Destructor.

## **wxGenericDirCtrl::Create**

**bool Create(wxWindow\* parent, const wxWindowID id = -1, const wxString& dir = wxDirDialogDefaultFolderStr, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDIRCTRL\_3D\_INTERNAL|wxSUNKEN\_BORDER, const wxString& filter = wxEmptyString, int defaultFilter = 0, const wxString& name = wxTreeCtrlNameStr)**

Create function for two-step construction. See *wxGenericDirCtrl::wxGenericDirCtrl* (p. 710) for details.

## **wxGenericDirCtrl::Init**

**void Init()**

Initializes variables.

## **wxGenericDirCtrl::CollapseTree**

**void CollapseTree()**

Collapses the entire tree.

### **wxGenericDirCtrl::ExpandPath**

**bool ExpandPath(const wxString& path)**

Tries to expand as much of the given path as possible, so that the filename or directory is visible in the tree control.

### **wxGenericDirCtrl::CollapsePath**

**bool CollapsePath(const wxString& path)**

Collapse the given path.

### **wxGenericDirCtrl::GetDefaultPath**

**wxString GetDefaultPath() const**

Gets the default path.

### **wxGenericDirCtrl::GetPath**

**wxString GetPath() const**

Gets the currently-selected directory or filename.

### **wxGenericDirCtrl::GetFilePath**

**wxString GetFilePath() const**

Gets selected filename path only (else empty string).

This function doesn't count a directory as a selection.

### **wxGenericDirCtrl::GetFilter**

**wxString GetFilter() const**

Returns the filter string.

### **wxGenericDirCtrl::GetFilterIndex**

**int GetFilterIndex() const**

Returns the current filter index (zero-based).

### **wxGenericDirCtrl::GetFilterListCtrl**

**wxDirFilterListCtrl\* GetFilterListCtrl() const**



Returns a pointer to the filter list control (if present).

**wxGenericDirCtrl::GetRootId****wxTreeItemId GetRootId()**

Returns the root id for the tree control.

**wxGenericDirCtrl::GetTreeCtrl****wxTreeCtrl\* GetTreeCtrl() const**

Returns a pointer to the tree control.

**wxGenericDirCtrl::ReCreateTree****void ReCreateTree()**

Collapse and expand the tree, thus re-creating it from scratch. May be used to update the displayed directory content.

**wxGenericDirCtrl::SetDefaultPath****void SetDefaultPath(const wxString& path)**

Sets the default path.

**wxGenericDirCtrl::SetFilter****void SetFilter(const wxString& filter)**

Sets the filter string.

**wxGenericDirCtrl::SetFilterIndex****void SetFilterIndex(int n)**

Sets the current filter index (zero-based).

**wxGenericDirCtrl::SetPath****void SetPath(const wxString& path)**

Sets the current path.

**wxGenericDirCtrl::ShowHidden****void ShowHidden(bool show)****Parameters**

*show*

If true, hidden folders and files will be displayed by the control. If false, they will not be displayed.

## **wxGenericValidator**

wxGenericValidator performs data transfer (but not validation or filtering) for the following basic controls: wxButton, wxCheckBox, wxListBox, wxStaticText, wxRadioButton, wxRadioBox, wxChoice, wxComboBox, wxGauge, wxSlider, wxScrollBar, wxSpinButton, wxTextCtrl, wxCheckListBox.

It checks the type of the window and uses an appropriate type for that window. For example, wxButton and wxTextCtrl transfer data to and from a wxString variable; wxListBox uses a wxArrayInt; wxCheckBox uses a bool.

For more information, please see *Validator overview* (p. 2092).

### **Derived from**

*wxValidator* (p. 1767)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### **Include files**

<wx/valgen.h>

### **See also**

*Validator overview* (p. 2092), *wxValidator* (p. 1767), *wxTextValidator* (p. 1668)

## **wxGenericValidator::wxGenericValidator**

**wxGenericValidator(const wxGenericValidator& validator)**

Copy constructor.

**wxGenericValidator(bool\* valPtr)**

Constructor taking a bool pointer. This will be used for wxCheckBox and wxRadioBox.

**wxGenericValidator(wxString\* valPtr)**

Constructor taking a wxString pointer. This will be used for wxButton, wxComboBox, wxStaticText, wxTextCtrl.

**wxGenericValidator(int\* valPtr)**

Constructor taking an integer pointer. This will be used for wxGauge, wxScrollBar, wxRadioBox, wxSpinButton, wxChoice.

**wxGenericValidator(wxArrayInt\* valPtr)**

Constructor taking a wxArrayInt pointer. This will be used for wxListBox, wxCheckListBox.

**Parameters**

*validator*

Validator to copy.

*valPtr*

A pointer to a variable that contains the value. This variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).

**wxGenericValidator::~~wxGenericValidator****~wxGenericValidator()**

Destructor.

**wxGenericValidator::Clone****virtual wxValidator\* Clone() const**

Clones the generic validator using the copy constructor.

**wxGenericValidator::TransferFromWindow****virtual bool TransferFromWindow()**

Transfers the value from the window to the appropriate data type.

**wxGenericValidator::TransferToWindow****virtual bool TransferToWindow()**

Transfers the value to the window.

**wxGLCanvas**

wxGLCanvas is a class for displaying OpenGL graphics.

There are two ways to use this class:

For the older (before wx 2.7.x) and simpler method, create a wxGLCanvas window using one of the three constructors that implicitly create a rendering context, call *wxGLCanvas::SetCurrent* (p. 719) to direct normal OpenGL commands to the window, and then call *wxGLCanvas::SwapBuffers* (p. 719) to show the OpenGL buffer on the window.

For the newer (wx 2.7.x+) method, create a `wxGLCanvas` window using the constructor that does *not* create an implicit rendering context, create an explicit instance of a `wxGLContext` (p. 719) that is initialized with the `wxGLCanvas` yourself, then use either `wxGLCanvas::SetCurrent` (p. 719) with the instance of the `wxGLContext` (p. 719) or `wxGLContext::SetCurrent` (p. 721) with the instance of the `wxGLCanvas` (p. 715) to bind the OpenGL state that is represented by the rendering context to the canvas, and then call `wxGLCanvas::SwapBuffers` (p. 719) to swap the buffers of the OpenGL canvas and thus show your current output.

To set up the attributes for the canvas (number of bits for the depth buffer, number of bits for the stencil buffer and so on) you should set up the correct values of the `attribList` parameter. The values that should be set up and their meanings will be described below.

To switch on `wxGLCanvas` support on under Windows, edit `setup.h` and set `wxUSE_GLCANVAS` to 1. You may also need to have to add `opengl32.lib` to the list of libraries your program is linked with. On Unix, pass `--with-opengl` to configure to compile using OpenGL or Mesa.

### Derived from

`wxWindow` (p. 1795)  
`wxEvtHandler` (p. 576)  
`wxObject` (p. 1148)

### Include files

<wx/glcanvas.h>

### Window styles

There are no specific window styles for this class.

See also *window styles overview* (p. 2089).

### Constants

The generic GL implementation doesn't support many of these options, such as stereo, auxiliary buffers, alpha channel, and accum buffer. Other implementations may support them.

<b>WX_GL_RGBA</b>	Use true colour
<b>WX_GL_BUFFER_SIZE</b>	Bits for buffer if not <b>WX_GL_RGBA</b>
<b>WX_GL_LEVEL</b>	0 for main buffer, >0 for overlay, <0 for underlay
<b>WX_GL_DOUBLEBUFFER</b>	Use doublebuffer
<b>WX_GL_STEREO</b>	Use stereoscopic display
<b>WX_GL_AUX_BUFFERS</b>	Number of auxiliary buffers (not all implementation support this option)
<b>WX_GL_MIN_RED</b>	Use red buffer with most bits (> MIN_RED bits)

<b>WX_GL_MIN_GREEN</b>	Use green buffer with most bits (> MIN_GREEN bits)
<b>WX_GL_MIN_BLUE</b>	Use blue buffer with most bits (> MIN_BLUE bits)
<b>WX_GL_MIN_ALPHA</b>	Use alpha buffer with most bits (> MIN_ALPHA bits)
<b>WX_GL_DEPTH_SIZE</b>	Bits for Z-buffer (0,16,32)
<b>WX_GL_STENCIL_SIZE</b>	Bits for stencil buffer
<b>WX_GL_MIN_ACCUM_RED</b>	Use red accum buffer with most bits (> MIN_ACCUM_RED bits)
<b>WX_GL_MIN_ACCUM_GREEN</b>	Use green buffer with most bits (> MIN_ACCUM_GREEN bits)
<b>WX_GL_MIN_ACCUM_BLUE</b>	Use blue buffer with most bits (> MIN_ACCUM_BLUE bits)
<b>WX_GL_MIN_ACCUM_ALPHA</b>	Use blue buffer with most bits (> MIN_ACCUM_ALPHA bits)

**See also**

*wxGLContext* (p. 719)

### **wxGLCanvas::wxGLCanvas**

**void wxGLCanvas(wxWindow\* parent, wxWindowID id = -1, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style=0, const wxString& name="GLCanvas", int\* attribList = 0, const wxPalette& palette = wxNullPalette)**

**void wxGLCanvas(wxWindow\* parent, wxGLContext\* sharedContext, wxWindowID id = -1, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style=0, const wxString& name="GLCanvas", int\* attribList = 0, const wxPalette& palette = wxNullPalette)**

**void wxGLCanvas(wxWindow\* parent, wxGLCanvas\* sharedCanvas, wxWindowID id = -1, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style=0, const wxString& name="GLCanvas", int\* attribList = 0, const wxPalette& palette = wxNullPalette)**

**void wxGLCanvas(wxWindow\* parent, wxWindowID id = wxID\_ANY, int\* attribList = 0, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style=0, const wxString& name="GLCanvas", const wxPalette& palette = wxNullPalette)**

Constructors. The first three constructors implicitly create an instance of *wxGLContext* (p. 719). The fourth constructor is identical to the first, except for the fact that it does *not* create such an implicit rendering context, which means that you have to create an explicit instance of *wxGLContext* (p. 719) yourself (highly recommended for future compatibility with wxWidgets and the flexibility of your own program!).

Note that if you used one of the first three constructors, `wxGLCanvas::GetContext` (p. 719) returns the pointer to the implicitly created instance, and the `wxGLCanvas::SetCurrent` (p. 719) method *without* the parameter should be used. If however you used the fourth constructor, `wxGLCanvas::GetContext` (p. 719) always returns NULL and the `wxGLCanvas::SetCurrent` (p. 719) method *with* the parameter must be used!

*parent*

Pointer to a parent window.

*sharedContext*

Context to share object resources with.

*id*

Window identifier. If -1, will automatically create an identifier.

*pos*

Window position. `wxDefaultPosition` is (-1, -1) which indicates that `wxWidgets` should generate a default position for the window.

*size*

Window size. `wxDefaultSize` is (-1, -1) which indicates that `wxWidgets` should generate a default size for the window. If no suitable size can be found, the window will be sized to 20x20 pixels so that the window is visible but obviously not correctly sized.

*style*

Window style.

*name*

Window name.

*attribList*

Array of int. With this parameter you can set the device context attributes associated to this window. This array is zero-terminated: it should be set up with constants described in the table above. If a constant should be followed by a value, put it in the next array position. For example, the `WX_GL_DEPTH_SIZE` should be followed by the value that indicates the number of bits for the depth buffer, so:

```
attribList[index]= WX_GL_DEPTH_SIZE;  
attribList[index+1]=32;  
and so on.
```

*palette*

If the window has the palette, it should by pass this value. Note: palette and `WX_GL_RGBA` are mutually exclusive.

**wxGLCanvas::GetContext****wxGLContext\* GetContext()**

Obtains the context that is associated with this canvas if one was implicitly created (by use of one of the first three constructors). Always returns NULL if the canvas was constructed with the fourth constructor.

**wxGLCanvas::SetCurrent****void SetCurrent()**

If this canvas was created with one of the first three constructors, a call to this method makes the implicit rendering context of this canvas current with this canvas, so that subsequent OpenGL calls modify the OpenGL state of the implicit rendering context.

If this canvas was created with the fourth constructor, this method should not be called (use the `SetCurrent` method below with the parameter instead)!

Note that this function may only be called *after* the window has been shown.

**wxGLCanvas::SetCurrent****void SetCurrent( const wxGLContext& RC )**

If this canvas was created with one of the first three constructors, this method should not be called (use the `SetCurrent` above without the parameter instead)!

If this canvas was created with the fourth constructor, a call to this method makes the OpenGL state that is represented by the OpenGL rendering context `RC` current with this canvas, and if `win` is an object of type `wxGLCanvas`, the statements `win.SetCurrent(RC);` and `RC.SetCurrent(win);` are equivalent, see `wxGLContext::SetCurrent` (p. 721).

Note that this function may only be called *after* the window has been shown.

**wxGLCanvas::SetColour****void SetColour(const char\* colour)**

Sets the current colour for this window, using the `wxWidgets` colour database to find a named colour.

**wxGLCanvas::SwapBuffers****void SwapBuffers()**

Swaps the double-buffer of this window, making the back-buffer the front-buffer and vice versa, so that the output of the previous OpenGL commands is displayed on the window.

**wxGLContext**

An instance of a `wxGLContext` represents the state of an OpenGL state machine and the connection between OpenGL and the system.

The OpenGL state includes everything that can be set with the OpenGL API: colors, rendering variables, display lists, texture objects, etc. Although it is possible to have multiple rendering contexts share display lists in order to save resources, this method is hardly used today any more, because display lists are only a tiny fraction of the overall state.

Therefore, one rendering context is usually used with or bound to multiple output windows in turn, so that the application has access to the *complete and identical* state while rendering into each window.

Binding (making current) a rendering context with another instance of a `wxGLCanvas` however works only if the other `wxGLCanvas` was created with the same attributes as the `wxGLCanvas` from which the `wxGLContext` was initialized. (This applies to sharing display lists among contexts analogously.)

Note that some `wxGLContext` features are extremely platform-specific - its best to check your native platform's `glcanvas` header (on windows include `wx/msw/glcanvas.h`) to see what features your native platform provides.

#### **Derived from**

`wxObject` (p. 1148)

#### **Include files**

`<wx/glcanvas.h>`

#### **See also**

`wxGLCanvas` (p. 715)

### **wxGLContext::wxGLContext**

**wxGLContext( wxGLCanvas\* win, const wxGLContext\* other=NULL )**

Constructor.

*win*

The canvas that is used to initialize this context. This parameter is needed only temporarily, and the caller may do anything with it (e.g. destroy the window) after the constructor returned.

It will be possible to bind (make current) this context to any other `wxGLCanvas` that has been created with equivalent attributes as *win*.

*other*

Context to share display lists with or NULL (the default) for no sharing.



**wxGLContext::SetCurrent****void SetCurrent(const wxGLCanvas& win)**

Makes the OpenGL state that is represented by this rendering context current with the wxGLCanvas *win*. Note that *win* can be a different wxGLCanvas window than the one that was passed to the constructor of this rendering context. If *RC* is an object of type wxGLContext, the statements *RC.SetCurrent(win)*; and *win.SetCurrent(RC)*; are equivalent, see *wxGLCanvas::SetCurrent* (p. 719).

**wxGraphicsBrush****Derived from***wxGraphicsObject* (p. 729)

A wxGraphicsBrush is a native representation of a brush. It is used for filling a path on a graphics context. The contents are specific and private to the respective renderer. Instances are ref counted and can therefore be assigned as usual. The only way to get a valid instance is via a Create...Brush call on the graphics context or the renderer instance.

**Include files**

&lt;wx/graphics.h&gt;

**wxGraphicsContext**

A wxGraphicsContext instance is the object that is drawn upon. It is created by a renderer using the CreateContext calls., this can be either directly using a renderer instance, or indirectly using the static convenience CreateXXX functions of wxGraphicsContext that always delegate the task to the default renderer.

**Derived from***wxGraphicsObject* (p. 729)**Include files**

&lt;wx/graphics.h&gt;

**wxGraphicsContext::Create****wxGraphicsContext\* Create(const wxWindowDC& dc)**

Creates a wxGraphicsContext from a wxWindowDC (eg a wxPaintDC).

**wxGraphicsContext\* Create(wxWindow\* window)**

Creates a wxGraphicsContext from a wxWindow.

**See also**

*wxGraphicsRenderer:: CreateContext* (p. 733)

**wxGraphicsContext::CreateFromNative**

Creates a `wxGraphicsContext` from a native context. This native context must be eg a `CGContextRef` for Core Graphics, a Graphics pointer for GDIPlus or a `cairo_t` pointer for cairo.

**wxGraphicsContext\* CreateFromNative(void \* context)**

Creates a `wxGraphicsContext` from a native window.

**See also**

*wxGraphicsRenderer:: CreateContextFromNativeContext* (p. 734)

**wxGraphicsContext::CreateFromNativeWindow**

**wxGraphicsContext\* CreateFromNativeWindow(void \* window)**

**See also**

*wxGraphicsRenderer:: CreateContextFromNativeWindow* (p. 734)

**wxGraphicsContext::CreatePen**

**wxGraphicsPen CreatePen(const wxPen& pen) const**

Creates a native pen from a `wxPen`.

**wxGraphicsContext::CreateBrush**

**wxGraphicsBrush CreateBrush(const wxBrush& brush) const**

Creates a native brush from a `wxBrush`.

**wxGraphicsContext::CreateRadialGradientBrush**

**wxGraphicsBrush CreateRadialGradientBrush(wxDouble xo, wxDouble yo, wxDouble xc, wxDouble yc, wxDouble radius, const wxColour& oColor, const wxColour& cColor) const**

Creates a native brush, having a radial gradient originating at (xo,yc) with color oColour and ends on a circle around (xc,yc) with radius r and color cColour

**wxGraphicsContext::CreateLinearGradientBrush**

**wxGraphicsBrush CreateLinearGradientBrush(wxDouble x1, wxDouble y1, wxDouble x2, wxDouble y2, const wxColour&c1, const wxColour&c2) const**

Creates a native brush, having a linear gradient, starting at (x1,y1) with color c1 to (x2,y2) with color c2

### **wxGraphicsContext::CreateFont**

**wxGraphicsFont CreateFont(const wxFont& font, const wxColour& col= \*wxBLACK) const**

Creates a native graphics font from a wxFont and a text colour.

### **wxGraphicsContext::CreateMatrix**

**wxGraphicsMatrix CreateMatrix(wxDouble a = 1.0, wxDouble b = 0.0, wxDouble c = 0.0, wxDouble d = 1.0, wxDouble tx = 0.0, wxDouble ty = 0.0) const**

Creates a native affine transformation matrix from the passed in values. The defaults result in an identity matrix.

### **wxGraphicsContext::CreatePath**

**wxGraphicsPath CreatePath() const**

Creates a native graphics path which is initially empty.

### **wxGraphicsContext::Clip**

**void Clip(const wxRegion& region)**

Clips drawings to the region, combined to current clipping region

**void Clip(wxDouble x, wxDouble y, wxDouble w, wxDouble h)**

Clips drawings to the rectangle.

### **wxGraphicsContext::ResetClip**

**void ResetClip()**

Resets the clipping to original shape.

### **wxGraphicsContext::DrawBitmap**

**void DrawBitmap(const wxBitmap& bmp, wxDouble x, wxDouble y, wxDouble w, wxDouble h)**

Draws the bitmap. In case of a mono bitmap, this is treated as a mask and the current brushed is used for filling.

### **wxGraphicsContext::DrawEllipse**

**void DrawEllipse(wxDouble x, wxDouble y, wxDouble w, wxDouble h)**

Draws an ellipse.

**wxGraphicsContext::DrawIcon**

**void DrawIcon(const wxIcon& icon, wxDouble x, wxDouble y, wxDouble w, wxDouble h)**

Draws the icon.

**wxGraphicsContext::DrawLines**

**void DrawLines(size\_t n, const wxPoint2DDouble\* points, int fillStyle = wxODDEVEN\_RULE)**

Draws a polygon.

**wxGraphicsContext::DrawPath**

**void DrawPath(const wxGraphicsPath& path, int fillStyle = wxODDEVEN\_RULE)**

Draws the path by first filling and then stroking.

**wxGraphicsContext::DrawRectangle**

**void DrawRectangle(wxDouble x, wxDouble y, wxDouble w, wxDouble h)**

Draws a rectangle.

**wxGraphicsContext::DrawRoundedRectangle**

**void DrawRoundedRectangle(wxDouble x, wxDouble y, wxDouble w, wxDouble h, wxDouble radius)**

Draws a rounded rectangle.

**wxGraphicsContext::DrawText**

**void DrawText(const wxString& str, wxDouble x, wxDouble y, wxDouble angle)**

**void DrawText(const wxString& str, wxDouble x, wxDouble y)**

Draws a text at the defined position, at the given angle.

**wxGraphicsContext::FillPath**

**void FillPath(const wxGraphicsPath& path, int fillStyle = wxODDEVEN\_RULE)**

Fills the path with the current brush.

**wxGraphicsContext::StrokePath****void StrokePath(const wxGraphicsPath& path)**

Strokes along a path with the current pen.

**wxGraphicsContext::GetNativeContext****void \* GetNativeContext()**

Returns the native context (CGContextRef for Core Graphics, Graphics pointer for GDIPlus and cairo\_t pointer for cairo).

**wxGraphicsContext::GetPartialTextExtents****void GetPartialTextExtents(const wxString& text, wxArrayDouble& widths) const**

Fills the *widths* array with the widths from the beginning of *text* to the corresponding character of *text*.

**wxGraphicsContext::GetTextExtent****void GetTextExtent(const wxString& text, wxDouble\* width, wxDouble\* height, wxDouble\* descent, wxDouble\* externalLeading) const**

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).

**wxGraphicsContext::Rotate****void Rotate(wxDouble angle)**

Rotates the current transformation matrix (radians),

**wxGraphicsContext::Scale****void Scale(wxDouble xScale, wxDouble yScale)**

Scales the current transformation matrix.

**wxGraphicsContext::Translate****void Translate(wxDouble dx, wxDouble dy)**

Translates the current transformation matrix.

**wxGraphicsContext::GetTransform**

**wxGraphicsMatrix GetTransform() const**

Gets the current transformation matrix of this context.

**wxGraphicsContext::SetTransform****void SetTransform(const wxGraphicsMatrix& *matrix*)**

Sets the current transformation matrix of this context

**wxGraphicsContext::ConcatTransform****void ConcatTransform(const wxGraphicsMatrix& *matrix*)**

Concatenates the passed in transform with the current transform of this context

**wxGraphicsContext::SetBrush****void SetBrush(const wxBrush& *brush*)****void SetBrush(const wxGraphicsBrush& *brush*)**

Sets the brush for filling paths.

**wxGraphicsContext::SetFont****void SetFont(const wxFont& *font*, const wxColour& *colour*)****void SetFont(const wxGraphicsFont& *font*)**

Sets the font for drawing text.

**wxGraphicsContext::SetPen****void SetPen(const wxGraphicsPen& *pen*)****void SetPen(const wxPen& *pen*)**

Sets the pen used for stroking.

**wxGraphicsContext::StrokeLine****void StrokeLine(wxDouble *x1*, wxDouble *y1*, wxDouble *x2*, wxDouble *y2*)**

Strokes a single line.

**wxGraphicsContext::StrokeLines****void StrokeLines(size\_t *n*, const wxPoint2DDouble\* *beginPoints*, const wxPoint2DDouble\* *endPoints*)**

**void StrokeLines(size\_t n, const wxPoint2DDouble\* points)**

Stroke disconnected lines from begin to end points, fastest method available for this purpose.

## wxGraphicsFont

### Derived from

*wxGraphicsObject* (p. 729)

A *wxGraphicsFont* is a native representation of a font (including text colour). The contents are specific and private to the respective renderer. Instances are ref counted and can therefore be assigned as usual. The only way to get a valid instance is via a *CreateFont* call on the graphics context or the renderer instance.

### Include files

<wx/graphics.h>

## wxGraphicsMatrix

A *wxGraphicsMatrix* is a native representation of an affine matrix. The contents are specific and private to the respective renderer. Instances are ref counted and can therefore be assigned as usual. The only way to get a valid instance is via a *CreateMatrix* call on the graphics context or the renderer instance.

### Derived from

*wxGraphicsObject* (p. 729)

### Include files

<wx/graphics.h>

### wxGraphicsMatrix::Concat

**void Concat(const wxGraphicsMatrix\* t)**

Concatenates the matrix passed with the current matrix.

**void Concat(const wxGraphicsMatrix& t)**

### wxGraphicsMatrix::Get

**void Get(wxDouble\* a=NULL, wxDouble\* b=NULL, wxDouble\* c=NULL, wxDouble\* d=NULL, wxDouble\* tx=NULL, wxDouble\* ty=NULL) const**

Returns the component values of the matrix via the argument pointers.

### **wxGraphicsMatrix::GetNativeMatrix**

**void \* GetNativeMatrix() const**

Returns the native representation of the matrix. For CoreGraphics this is a CGAffineTransform pointer. For GDIPlus a Matrix Pointer and for Cairo a cairo\_matrix\_t pointer.

### **wxGraphicsMatrix::Invert**

**void Invert()**

Inverts the matrix.

### **wxGraphicsMatrix::IsEqual**

**bool IsEqual(const wxGraphicsMatrix& t) const**

Returns true if the elements of the transformation matrix are equal.

### **wxGraphicsMatrix::IsIdentity**

**bool IsIdentity() const**

Return true if this is the identity matrix.

### **wxGraphicsMatrix::Rotate**

**void Rotate(wxDouble angle)**

Rotates this matrix (radians).

### **wxGraphicsMatrix::Scale**

**void Scale(wxDouble xScale, wxDouble yScale)**

Scales this matrix.

### **wxGraphicsMatrix::Translate**

**void Translate(wxDouble dx, wxDouble dy)**

Translates this matrix.

### **wxGraphicsMatrix::Set**

**void Set(wxDouble a = 1.0, wxDouble b = 0.0, wxDouble c = 0.0, wxDouble d = 1.0, wxDouble tx = 0.0, wxDouble ty = 0.0)**



Sets the matrix to the respective values (default values are the identity matrix)

### **wxGraphicsMatrix::TransformPoint**

**void TransformPoint(wxDouble\* x, wxDouble\* y) const**

Applies this matrix to a point.

### **wxGraphicsMatrix::TransformDistance**

**void TransformDistance(wxDouble\* dx, wxDouble\* dy) const**

Applies this matrix to a distance (ie. performs all transforms except translations)

## **wxGraphicsObject**

This class is the superclass of native graphics objects like pens etc. It allows reference counting. Not instantiated by user code.

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/graphics.h>

### **See also**

*wxGraphicsBrush* (p. 721), *wxGraphicsPen* (p. 732), *wxGraphicsMatrix* (p. 727),  
*wxGraphicsPath* (p. 729)

### **wxGraphicsObject::GetRenderer**

**wxGraphicsRenderer\* GetRenderer() const**

Returns the renderer that was used to create this instance, or NULL if it has not been initialized yet

### **wxGraphicsObject::IsNull**

**bool IsNull() const**

Is this object valid (false) or still empty (true)?

## **wxGraphicsPath**

A *wxGraphicsPath* is a native representation of an geometric path. The contents are

specific and private to the respective renderer. Instances are ref counted and can therefore be assigned as usual. The only way to get a valid instance is via a `CreatePath` call on the graphics context or the renderer instance.

**Derived from**

*wxGraphicsObject* (p. 729)

**Include files**

<wx/graphics.h>

**wxGraphicsPath::MoveToPoint**

**void MoveToPoint(wxDouble x, wxDouble y)**

**void MoveToPoint(const wxPoint2DDouble& p)**

Begins a new subpath at (x,y)

**wxGraphicsPath::AddArc**

**void AddArc(wxDouble x, wxDouble y, wxDouble r, wxDouble startAngle, wxDouble endAngle, bool clockwise)**

Adds an arc of a circle centering at (x,y) with radius (r) from startAngle to endAngle.

**void AddArc(const wxPoint2DDouble& c, wxDouble r, wxDouble startAngle, wxDouble endAngle, bool clockwise)**

**wxGraphicsPath::AddArcToPoint**

**void AddArcToPoint(wxDouble x1, wxDouble y1, wxDouble x2, wxDouble y2, wxDouble r)**

Appends an arc to two tangents connecting (current) to (x1,y1) and (x1,y1) to (x2,y2), also a straight line from (current) to (x1,y1).

**wxGraphicsPath::AddCircle**

**void AddCircle(wxDouble x, wxDouble y, wxDouble r)**

Appends a circle around (x,y) with radius r as a new closed subpath.

**wxGraphicsPath::AddCurveToPoint**

**void AddCurveToPoint(wxDouble cx1, wxDouble cy1, wxDouble cx2, wxDouble cy2, wxDouble x, wxDouble y)**

Adds a cubic Bezier curve from the current point, using two control points and an end

point.

**void AddCurveToPoint(const wxPoint2DDouble& c1, const wxPoint2DDouble& c2, const wxPoint2DDouble& e)**

### **wxGraphicsPath::AddEllipse**

**void AddEllipse(wxDouble x, wxDouble y, wxDouble w, wxDouble h)**

Appends an ellipse fitting into the passed in rectangle.

### **wxGraphicsPath::AddLineToPoint**

**void AddLineToPoint(wxDouble x, wxDouble y)**

Adds a straight line from the current point to (x,y).

**void AddLineToPoint(const wxPoint2DDouble& p)**

### **wxGraphicsPath::AddPath**

**void AddPath(const wxGraphicsPath& path)**

Adds another path.

### **wxGraphicsPath::AddQuadCurveToPoint**

**void AddQuadCurveToPoint(wxDouble cx, wxDouble cy, wxDouble x, wxDouble y)**

Adds a quadratic Bezier curve from the current point, using a control point and an end point.

### **wxGraphicsPath::AddRectangle**

**void AddRectangle(wxDouble x, wxDouble y, wxDouble w, wxDouble h)**

Appends a rectangle as a new closed subpath.

### **wxGraphicsPath::AddRoundedRectangle**

**void AddRoundedRectangle(wxDouble x, wxDouble y, wxDouble w, wxDouble h, wxDouble radius)**

Appends a rounded rectangle as a new closed subpath.

### **wxGraphicsPath::CloseSubpath**

**void CloseSubpath()**

Closes the current sub-path.

**wxGraphicsPath::Contains****bool Contains(const wxPoint2DDouble& c, int fillStyle = wxODDEVEN\_RULE) const****bool Contains(wxDouble x, wxDouble y, int fillStyle = wxODDEVEN\_RULE) const**

Returns true if the point is within the path.

**wxGraphicsPath::GetBox****wxRect2DDouble GetBox() const****void GetBox(wxDouble\* x, wxDouble\* y, wxDouble\* w, wxDouble\* h) const**

Gets the bounding box enclosing all points (possibly including control points).

**wxGraphicsPath::GetCurrentPoint****void GetCurrentPoint(wxDouble\* x, wxDouble\* y) const****wxPoint2DDouble GetCurrentPoint() const**

Gets the last point of the current path, (0,0) if not yet set.

**wxGraphicsPath::Transform****void Transform(const wxGraphicsMatrix& matrix)**

Transforms each point of this path by the matrix.

**wxGraphicsPath::GetNativePath****void \* GetNativePath() const**

Returns the native path (CGPathRef for Core Graphics, Path pointer for GDIPlus and a `cairo_path_t` pointer for cairo).

**wxGraphicsPath::UnGetNativePath****void UnGetNativePath(void\* p) const**

Gives back the native path returned by `GetNativePath()` because there might be some deallocations necessary (eg on cairo the native path returned by `GetNativePath` is newly allocated each time).

**wxGraphicsPen****Derived from***wxGraphicsObject* (p. 729)

A `wxGraphicsPen` is a native representation of a pen. It is used for stroking a path on a graphics context. The contents are specific and private to the respective renderer. Instances are ref counted and can therefore be assigned as usual. The only way to get a valid instance is via a `CreatePen` call on the graphics context or the renderer instance.

**Include files**

<wx/graphics.h>

## **wxGraphicsRenderer**

A `wxGraphicsRenderer` is the instance corresponding to the rendering engine used. There may be multiple instances on a system, if there are different rendering engines present, but there is always one instance per engine, eg there is ONE core graphics renderer instance on OSX. This instance is pointed back to by all objects created by it (`wxGraphicsContext`, `wxGraphicsPath` etc). Therefore you can create additional instances of paths etc. by calling `GetRenderer()` and then using the appropriate `CreateXXX` function.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/graphics.h>

**Data structures**

### **wxGraphicsRenderer::GetDefaultRenderer**

#### **wxGraphicsRenderer\* GetDefaultRenderer()**

Returns the default renderer on this platform. On OS X this is the Core Graphics (a.k.a. Quartz 2D) renderer, on MSW the GDIPlus renderer, and on GTK we currently default to the cairo renderer.

### **wxGraphicsRenderer::CreateContext**

#### **wxGraphicsContext \* CreateContext(const wxWindowDC& dc)**

Creates a `wxGraphicsContext` from a `wxWindowDC` (eg a `wxPaintDC`).

#### **wxGraphicsContext \* CreateContext(wxWindow\* window)**

Creates a `wxGraphicsContext` from a `wxWindow`.

### **wxGraphicsRenderer::CreateContextFromNativeContext**

**wxGraphicsContext \* CreateContextFromNativeContext(void \* context)**

Creates a wxGraphicsContext from a native context. This native context must be eg a CGContextRef for Core Graphics, a Graphics pointer for GDIPlus or a cairo\_t pointer for cairo.

**wxGraphicsRenderer::CreateContextFromNativeWindow**

**wxGraphicsContext \* CreateContextFromNativeWindow(void \* window)**

Creates a wxGraphicsContext from a native window.

**wxGraphicsRenderer::CreatePen**

**wxGraphicsPen CreatePen(const wxPen& pen)**

Creates a native pen from a wxPen.

**wxGraphicsRenderer::CreateBrush**

**wxGraphicsBrush CreateBrush(const wxBrush& brush)**

Creates a native brush from a wxBrush.

**wxGraphicsRenderer::CreateLinearGradientBrush**

**wxGraphicsBrush CreateLinearGradientBrush(wxDouble x1, wxDouble y1, wxDouble x2, wxDouble y2, const wxColour&c1, const wxColour&c2)**

Creates a native brush, having a linear gradient, starting at (x1,y1) with color c1 to (x2,y2) with color c2

**wxGraphicsRenderer::CreateRadialGradientBrush**

**wxGraphicsBrush CreateRadialGradientBrush(wxDouble xo, wxDouble yo, wxDouble xc, wxDouble yc, wxDouble radius, const wxColour& oColour, const wxColour& cColour)**

Creates a native brush, having a radial gradient originating at (xo,yc) with color oColour and ends on a circle around (xc,yc) with radius r and color cColour

**wxGraphicsRenderer::CreateFont**

**wxGraphicsFont CreateFont(const wxFont& font, const wxColour& col = \*wxBLACK)**

Creates a native graphics font from a wxFont and a text colour.

**wxGraphicsRenderer::CreateMatrix**

**wxGraphicsMatrix CreateMatrix(wxDouble a = 1.0, wxDouble b = 0.0, wxDouble c =**

*0.0*, **wxDouble** *d = 1.0*, **wxDouble** *tx = 0.0*, **wxDouble** *ty = 0.0*)

Creates a native affine transformation matrix from the passed in values. The defaults result in an identity matrix.

### **wxGraphicsRenderer::CreatePath**

**wxGraphicsPath** **CreatePath()**

Creates a native graphics path which is initially empty.

## **wxGrid**

**wxGrid** and its related classes are used for displaying and editing tabular data. They provide a rich set of features for display, editing, and interacting with a variety of data sources. For simple applications, and to help you get started, **wxGrid** is the only class you need to refer to directly. It will set up default instances of the other classes and manage them for you. For more complex applications you can derive your own classes for custom grid views, grid data tables, cell editors and renderers. The *wxGrid classes overview* (p. 2143) has examples of simple and more complex applications, explains the relationship between the various grid classes and has a summary of the keyboard shortcuts and mouse functions provided by **wxGrid**.

**wxGrid** has been greatly expanded and redesigned for **wxWidgets** 2.2 onwards. If you have been using the old **wxGrid** class you will probably want to have a look at the *wxGrid classes overview* (p. 2143) to see how things have changed. The new grid classes are reasonably backward-compatible but there are some exceptions. There are also easier ways of doing many things compared to the previous implementation.

### **Derived from**

*wxScrolledWindow* (p. 1414)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### **Include files**

<wx/grid.h>

### **Window styles**

There are presently no specific window styles for **wxGrid**.

### **Event handling**

The event handler for the following functions takes a *wxGridEvent* (p. 782) parameter. The *...\_CMD\_...* variants also take a window identifier.

**EVT\_GRID\_CELL\_LEFT\_CLICK(func)** The user clicked a cell with the left mouse button.  
Processes a  
**wxEVT\_GRID\_CELL\_LEFT\_CLICK**.

- EVT\_GRID\_CELL\_RIGHT\_CLICK(func)** The user clicked a cell with the right mouse button. Processes a `wxEVT_GRID_CELL_RIGHT_CLICK`.
- EVT\_GRID\_CELL\_LEFT\_DCLICK(func)** The user double-clicked a cell with the left mouse button. Processes a `wxEVT_GRID_CELL_LEFT_DCLICK`.
- EVT\_GRID\_CELL\_RIGHT\_DCLICK(func)** The user double-clicked a cell with the right mouse button. Processes a `wxEVT_GRID_CELL_RIGHT_DCLICK`.
- EVT\_GRID\_LABEL\_LEFT\_CLICK(func)** The user clicked a label with the left mouse button. Processes a `wxEVT_GRID_LABEL_LEFT_CLICK`.
- EVT\_GRID\_LABEL\_RIGHT\_CLICK(func)** The user clicked a label with the right mouse button. Processes a `wxEVT_GRID_LABEL_RIGHT_CLICK`.
- EVT\_GRID\_LABEL\_LEFT\_DCLICK(func)** The user double-clicked a label with the left mouse button. Processes a `wxEVT_GRID_LABEL_LEFT_DCLICK`.
- EVT\_GRID\_LABEL\_RIGHT\_DCLICK(func)** The user double-clicked a label with the right mouse button. Processes a `wxEVT_GRID_LABEL_RIGHT_DCLICK`.
- EVT\_GRID\_CELL\_CHANGE(func)** The user changed the data in a cell. Processes a `wxEVT_GRID_CELL_CHANGE`.
- EVT\_GRID\_SELECT\_CELL(func)** The user moved to, and selected a cell. Processes a `wxEVT_GRID_SELECT_CELL`.
- EVT\_GRID\_EDITOR\_HIDDEN(func)** The editor for a cell was hidden. Processes a `wxEVT_GRID_EDITOR_HIDDEN`.
- EVT\_GRID\_EDITOR\_SHOWN(func)** The editor for a cell was shown. Processes a `wxEVT_GRID_EDITOR_SHOWN`.
- EVT\_GRID\_CMD\_CELL\_LEFT\_CLICK(id, func)** The user clicked a cell with the left mouse button; variant taking a window identifier. Processes a `wxEVT_GRID_CELL_LEFT_CLICK`.
- EVT\_GRID\_CMD\_CELL\_RIGHT\_CLICK(id, func)** The user clicked a cell with the right mouse button; variant taking a window identifier. Processes a `wxEVT_GRID_CELL_RIGHT_CLICK`.
- EVT\_GRID\_CMD\_CELL\_LEFT\_DCLICK(id, func)** The user double-clicked a cell with the left mouse button; variant taking a window identifier. Processes a



`wxEVT_GRID_CELL_LEFT_DCLICK`.

**EVT\_GRID\_CMD\_CELL\_RIGHT\_DCLICK(id, func)** The user double-clicked a cell with the right mouse button; variant taking a window identifier. Processes a `wxEVT_GRID_CELL_RIGHT_DCLICK`.

**EVT\_GRID\_CMD\_LABEL\_LEFT\_CLICK(id, func)** The user clicked a label with the left mouse button; variant taking a window identifier. Processes a `wxEVT_GRID_LABEL_LEFT_CLICK`.

**EVT\_GRID\_CMD\_LABEL\_RIGHT\_CLICK(id, func)** The user clicked a label with the right mouse button; variant taking a window identifier. Processes a `wxEVT_GRID_LABEL_RIGHT_CLICK`.

**EVT\_GRID\_CMD\_LABEL\_LEFT\_DCLICK(id, func)** The user double-clicked a label with the left mouse button; variant taking a window identifier. Processes a `wxEVT_GRID_LABEL_LEFT_DCLICK`.

**EVT\_GRID\_CMD\_LABEL\_RIGHT\_DCLICK(id, func)** The user double-clicked a label with the right mouse button; variant taking a window identifier. Processes a `wxEVT_GRID_LABEL_RIGHT_DCLICK`.

**EVT\_GRID\_CMD\_CELL\_CHANGE(id, func)** The user changed the data in a cell; variant taking a window identifier. Processes a `wxEVT_GRID_CELL_CHANGE`.

**EVT\_GRID\_CMD\_SELECT\_CELL(id, func)** The user moved to, and selected a cell; variant taking a window identifier. Processes a `wxEVT_GRID_SELECT_CELL`.

**EVT\_GRID\_CMD\_EDITOR\_HIDDEN(id, func)** The editor for a cell was hidden; variant taking a window identifier. Processes a `wxEVT_GRID_EDITOR_HIDDEN`.

**EVT\_GRID\_CMD\_EDITOR\_SHOWN(id, func)** The editor for a cell was shown; variant taking a window identifier. Processes a `wxEVT_GRID_EDITOR_SHOWN`.

The event handler for the following functions takes a *wxGridSizeEvent* (p. 787) parameter. The `..._CMD_...` variants also take a window identifier.

**EVT\_GRID\_COL\_SIZE(func)** The user resized a column by dragging it. Processes a `wxEVT_GRID_COL_SIZE`.

**EVT\_GRID\_ROW\_SIZE(func)** The user resized a row by dragging it. Processes a `wxEVT_GRID_ROW_SIZE`.

**EVT\_GRID\_CMD\_COL\_SIZE(func)** The user resized a column by dragging it; variant taking a window identifier. Processes a

`wxEVT_GRID_COL_SIZE`.

**EVT\_GRID\_CMD\_ROW\_SIZE(func)** The user resized a row by dragging it; variant taking a window identifier. Processes a `wxEVT_GRID_ROW_SIZE`.

The event handler for the following functions takes a *wxGridRangeSelectEvent* (p. 785) parameter. The `..._CMD_...` variant also takes a window identifier.

**EVT\_GRID\_RANGE\_SELECT(func)** The user selected a group of contiguous cells. Processes a `wxEVT_GRID_RANGE_SELECT`.

**EVT\_GRID\_CMD\_RANGE\_SELECT(id, func)** The user selected a group of contiguous cells; variant taking a window identifier. Processes a `wxEVT_GRID_RANGE_SELECT`.

The event handler for the following functions takes a *wxGridEditorCreatedEvent* (p. 781) parameter. The `..._CMD_...` variant also takes a window identifier.

**EVT\_GRID\_EDITOR\_CREATED(func)** The editor for a cell was created. Processes a `wxEVT_GRID_EDITOR_CREATED`.

**EVT\_GRID\_CMD\_EDITOR\_CREATED(id, func)** The editor for a cell was created; variant taking a window identifier. Processes a `wxEVT_GRID_EDITOR_CREATED`.

#### See also

*wxGrid overview* (p. 2143)

### Constructors and initialization

*wxGrid* (p. 738)

*~wxGrid* (p. 739)

*CreateGrid* (p. 742)

*SetTable* (p. 768)

### Display format

### Selection functions

*wxGrid::ClearSelection* (p. 742)

*wxGrid::IsSelection* (p. 756)

*wxGrid::SelectAll* (p. 759)

*wxGrid::SelectBlock* (p. 759)

*wxGrid::SelectCol* (p. 759)

*wxGrid::SelectRow* (p. 759)

### **wxGrid::wxGrid**

**wxGrid()**

Default constructor

**wxGrid**(**wxWindow\*** *parent*, **wxWindowID** *id*, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = *wxWANTS\_CHARS*, **const wxString&** *name* = *wxPanelNameStr*)

Constructor to create a grid object. Call either *wxGrid::CreateGrid* (p. 742) or *wxGrid::SetTable* (p. 768) directly after this to initialize the grid before using it.

**wxGrid::~wxGrid****~wxGrid()**

Destructor. This will also destroy the associated grid table unless you passed a table object to the grid and specified that the grid should not take ownership of the table (see *wxGrid::SetTable* (p. 768)).

**wxGrid::AppendCols**

**bool AppendCols**(**int** *numCols* = 1, **bool** *updateLabels* = *true*)

Appends one or more new columns to the right of the grid and returns true if successful. The *updateLabels* argument is not used at present.

If you are using a derived grid table class you will need to override *wxGridTableBase::AppendCols* (p. 795). See *wxGrid::InsertCols* (p. 754) for further information.

**wxGrid::AppendRows**

**bool AppendRows**(**int** *numRows* = 1, **bool** *updateLabels* = *true*)

Appends one or more new rows to the bottom of the grid and returns true if successful. The *updateLabels* argument is not used at present.

If you are using a derived grid table class you will need to override *wxGridTableBase::AppendRows* (p. 795). See *wxGrid::InsertRows* (p. 755) for further information.

**wxGrid::AutoSize****void AutoSize()**

Automatically sets the height and width of all rows and columns to fit their contents.

**Note**

*wxGrid* sets up arrays to store individual row and column sizes when non-default sizes are used. The memory requirements for this could become prohibitive if your grid is very large.

**wxGrid::AutoSizeColumn****void AutoSizeColumn(int col, bool setAsMin = true)**

Automatically sizes the column to fit its contents. If setAsMin is true the calculated width will also be set as the minimal width for the column.

**Note**

wxGrid sets up arrays to store individual row and column sizes when non-default sizes are used. The memory requirements for this could become prohibitive if your grid is very large.

**wxGrid::AutoSizeColumns****void AutoSizeColumns(bool setAsMin = true)**

Automatically sizes all columns to fit their contents. If setAsMin is true the calculated widths will also be set as the minimal widths for the columns.

**Note**

wxGrid sets up arrays to store individual row and column sizes when non-default sizes are used. The memory requirements for this could become prohibitive if your grid is very large.

**wxGrid::AutoSizeRow****void AutoSizeRow(int row, bool setAsMin = true)**

Automatically sizes the row to fit its contents. If setAsMin is true the calculated height will also be set as the minimal height for the row.

**Note**

wxGrid sets up arrays to store individual row and column sizes when non-default sizes are used. The memory requirements for this could become prohibitive if your grid is very large.

**wxGrid::AutoSizeRows****void AutoSizeRows(bool setAsMin = true)**

Automatically sizes all rows to fit their contents. If setAsMin is true the calculated heights will also be set as the minimal heights for the rows.

**Note**

wxGrid sets up arrays to store individual row and column sizes when non-default sizes are used. The memory requirements for this could become prohibitive if your grid is very large.

**wxGrid::BeginBatch****void BeginBatch()**

Increments the grid's batch count. When the count is greater than zero repainting of the grid is suppressed. Each call to BeginBatch must be matched by a later call to `wxGrid::EndBatch` (p. 744). Code that does a lot of grid modification can be enclosed

between `BeginBatch` and `EndBatch` calls to avoid screen flicker. The final `EndBatch` will cause the grid to be repainted.

### **`wxGrid::BlockToDeviceRect`**

**`wxRect BlockToDeviceRect(const wxGridCellCoords & topLeft, const wxGridCellCoords & bottomRight)`**

This function returns the rectangle that encloses the block of cells limited by `TopLeft` and `BottomRight` cell in device coords and clipped to the client size of the grid window.

### **`wxGrid::CanDragColMove`**

**`bool CanDragColMove()`**

Returns true if columns can be moved by dragging with the mouse. Columns can be moved by dragging on their labels.

### **`wxGrid::CanDragColSize`**

**`bool CanDragColSize()`**

Returns true if columns can be resized by dragging with the mouse. Columns can be resized by dragging the edges of their labels. If grid line dragging is enabled they can also be resized by dragging the right edge of the column in the grid cell area (see *`wxGrid::EnableDragGridSize`* (p. 744)).

### **`wxGrid::CanDragRowSize`**

**`bool CanDragRowSize()`**

Returns true if rows can be resized by dragging with the mouse. Rows can be resized by dragging the edges of their labels. If grid line dragging is enabled they can also be resized by dragging the lower edge of the row in the grid cell area (see *`wxGrid::EnableDragGridSize`* (p. 744)).

### **`wxGrid::CanDragGridSize`**

**`bool CanDragGridSize()`**

Return true if the dragging of grid lines to resize rows and columns is enabled or false otherwise.

### **`wxGrid::CanEnableCellControl`**

**`bool CanEnableCellControl() const`**

Returns true if the in-place edit control for the current grid cell can be used and false otherwise (e.g. if the current cell is read-only).

**wxGrid::CanHaveAttributes****bool CanHaveAttributes()**

Do we have some place to store attributes in?

**wxGrid::CellToRect****wxRect CellToRect(int row, int col)****wxRect CellToRect(const wxGridCellCoords& coords)**

Return the rectangle corresponding to the grid cell's size and position in logical coordinates.

**wxGrid::ClearGrid****void ClearGrid()**

Clears all data in the underlying grid table and repaints the grid. The table is not deleted by this function. If you are using a derived table class then you need to override *wxGridTableBase::Clear* (p. 795) for this function to have any effect.

**wxGrid::ClearSelection****void ClearSelection()**

Deselects all cells that are currently selected.

**wxGrid::CreateGrid****bool CreateGrid(int numRows, int numCols, wxGrid::wxGridSelectionMode selmode = wxGrid::wxGridSelectCells)**

Creates a grid with the specified initial number of rows and columns. Call this directly after the grid constructor. When you use this function *wxGrid* will create and manage a simple table of string values for you. All of the grid data will be stored in memory.

For applications with more complex data types or relationships, or for dealing with very large datasets, you should derive your own grid table class and pass a table object to the grid with *wxGrid::SetTable* (p. 768).

**wxGrid::DeleteCols****bool DeleteCols(int pos = 0, int numCols = 1, bool updateLabels = true)**

Deletes one or more columns from a grid starting at the specified position and returns true if successful. The *updateLabels* argument is not used at present.

If you are using a derived grid table class you will need to override *wxGridTableBase::DeleteCols* (p. 795). See *wxGrid::InsertCols* (p. 754) for further

information.

### **wxGrid::DeleteRows**

**bool DeleteRows**(int *pos* = 0, int *numRows* = 1, bool *updateLabels* = true)

Deletes one or more rows from a grid starting at the specified position and returns true if successful. The *updateLabels* argument is not used at present.

If you are using a derived grid table class you will need to override *wxGridTableBase::DeleteRows* (p. 795). See *wxGrid::InsertRows* (p. 755) for further information.

### **wxGrid::DisableCellEditControl**

**void DisableCellEditControl**()

Disables in-place editing of grid cells. Equivalent to calling *EnableCellEditControl*(false).

### **wxGrid::DisableDragColMove**

**void DisableDragColMove**()

Disables column moving by dragging with the mouse. Equivalent to passing false to *wxGrid::EnableDragColMove* (p. 744).

### **wxGrid::DisableDragColSize**

**void DisableDragColSize**()

Disables column sizing by dragging with the mouse. Equivalent to passing false to *wxGrid::EnableDragColSize* (p. 744).

### **wxGrid::DisableDragGridSize**

**void DisableDragGridSize**()

Disable mouse dragging of grid lines to resize rows and columns. Equivalent to passing false to *wxGrid::EnableDragGridSize* (p. 744)

### **wxGrid::DisableDragRowSize**

**void DisableDragRowSize**()

Disables row sizing by dragging with the mouse. Equivalent to passing false to *wxGrid::EnableDragRowSize* (p. 744).

### **wxGrid::EnableCellEditControl**

**void EnableCellEditControl**(bool *enable* = true)

Enables or disables in-place editing of grid cell data. The grid will issue either a `wxEVT_GRID_EDITOR_SHOWN` or `wxEVT_GRID_EDITOR_HIDDEN` event.

**wxGrid::EnableDragColSize**

**void EnableDragColSize**(bool *enable* = true)

Enables or disables column sizing by dragging with the mouse.

**wxGrid::EnableDragColMove**

**void EnableDragColMove**(bool *enable* = true)

Enables or disables column moving by dragging with the mouse.

**wxGrid::EnableDragGridSize**

**void EnableDragGridSize**(bool *enable* = true)

Enables or disables row and column resizing by dragging gridlines with the mouse.

**wxGrid::EnableDragRowSize**

**void EnableDragRowSize**(bool *enable* = true)

Enables or disables row sizing by dragging with the mouse.

**wxGrid::EnableEditing**

**void EnableEditing**(bool *edit*)

If the *edit* argument is false this function sets the whole grid as read-only. If the argument is true the grid is set to the default state where cells may be editable. In the default state you can set single grid cells and whole rows and columns to be editable or read-only via `wxGridCellAttribute::SetReadOnly` (p. 770). For single cells you can also use the shortcut function `wxGrid::SetReadOnly` (p. 765).

For more information about controlling grid cell attributes see the `wxGridCellAttr` (p. 769) cell attribute class and the `wxGrid` classes overview (p. 2143).

**wxGrid::EnableGridLines**

**void EnableGridLines**(bool *enable* = true)

Turns the drawing of grid lines on or off.

**wxGrid::EndBatch**

**void EndBatch**()



Decrements the grid's batch count. When the count is greater than zero repainting of the grid is suppressed. Each previous call to `wxGrid::BeginBatch` (p. 740) must be matched by a later call to `EndBatch`. Code that does a lot of grid modification can be enclosed between `BeginBatch` and `EndBatch` calls to avoid screen flicker. The final `EndBatch` will cause the grid to be repainted.

### **wxGrid::Fit**

**void Fit()**

Overridden `wxWindow` method.

### **wxGrid::ForceRefresh**

**void ForceRefresh()**

Causes immediate repainting of the grid. Use this instead of the usual `wxWindow::Refresh`.

### **wxGrid::GetBatchCount**

**int GetBatchCount()**

Returns the number of times that `wxGrid::BeginBatch` (p. 740) has been called without (yet) matching calls to `wxGrid::EndBatch` (p. 744). While the grid's batch count is greater than zero the display will not be updated.

### **wxGrid::GetCellAlignment**

**void GetCellAlignment(int row, int col, int\* horiz, int\* vert)**

Sets the arguments to the horizontal and vertical text alignment values for the grid cell at the specified location.

Horizontal alignment will be one of `wxALIGN_LEFT`, `wxALIGN_CENTRE` or `wxALIGN_RIGHT`.

Vertical alignment will be one of `wxALIGN_TOP`, `wxALIGN_CENTRE` or `wxALIGN_BOTTOM`.

**wxPerl note:** This method only takes the parameters `row` and `col` and returns a 2-element list ( `horiz, vert` ).

### **wxGrid::GetCellBackgroundColour**

**wxColour GetCellBackgroundColour(int row, int col)**

Returns the background colour of the cell at the specified location.

### **wxGrid::GetCellEditor**

**wxGridCellEditor\* GetCellEditor(int row, int col)**

Returns a pointer to the editor for the cell at the specified location.

See *wxGridCellEditor* (p. 776) and the *wxGrid overview* (p. 2143) for more information about cell editors and renderers.

**wxGrid::GetCellFont****wxFont GetCellFont(int row, int col)**

Returns the font for text in the grid cell at the specified location.

**wxGrid::GetCellRenderer****wxGridCellRenderer\* GetCellRenderer(int row, int col)**

Returns a pointer to the renderer for the grid cell at the specified location.

See *wxGridCellRenderer* (p. 791) and the *wxGrid overview* (p. 2143) for more information about cell editors and renderers.

**wxGrid::GetCellTextColour****wxColour GetCellTextColour(int row, int col)**

Returns the text colour for the grid cell at the specified location.

**wxGrid::GetCellValue****wxString GetCellValue(int row, int col)****wxString GetCellValue(const wxGridCellCoords&coords)**

Returns the string contained in the cell at the specified location. For simple applications where a grid object automatically uses a default grid table of string values you use this function together with *wxGrid::SetCellValue* (p. 760) to access cell values.

For more complex applications where you have derived your own grid table class that contains various data types (e.g. numeric, boolean or user-defined custom types) then you only use this function for those cells that contain string values.

See *wxGridTableBase::CanGetValueAs* (p. 794) and the *wxGrid overview* (p. 2143) for more information.

**wxGrid::GetColAt****int GetColAt(int colPos) const**

Returns the column ID of the specified column position.

**wxGrid::GetColLeft****int GetColLeft(int col) const****wxGrid::GetColLabelAlignment****void GetColLabelAlignment(int\* horiz, int\* vert)**

Sets the arguments to the current column label alignment values.

Horizontal alignment will be one of wxALIGN\_LEFT, wxALIGN\_CENTRE or wxALIGN\_RIGHT.

Vertical alignment will be one of wxALIGN\_TOP, wxALIGN\_CENTRE or wxALIGN\_BOTTOM.

**wxPerl note:** This method takes no parameters and returns a 2-element list ( horiz, vert ).

**wxGrid::GetColLabelSize****int GetColLabelSize()**

Returns the current height of the column labels.

**wxGrid::GetColLabelValue****wxString GetColLabelValue(int col)**

Returns the specified column label. The default grid table class provides column labels of the form A,B...Z,AA,AB...ZZ,AAA... If you are using a custom grid table you can override `wxGridTableBase::GetColLabelValue` (p. 795) to provide your own labels.

**wxGrid::GetColMinimalAcceptableWidth****int GetColMinimalAcceptableWidth()**

This returns the value of the lowest column width that can be handled correctly. See member `SetColMinimalAcceptableWidth` (p. 762) for details.

**wxGrid::GetColMinimalWidth****int GetColMinimalWidth(int col) const**

Get the minimal width of the given column/row.

**wxGrid::GetColPos****int GetColPos(int colID) const**

Returns the position of the specified column.

**wxGrid::GetColRight****int GetColRight(int col) const****wxGrid::GetColSize****int GetColSize(int col)**

Returns the width of the specified column.

**wxGrid::GetDefaultCellAlignment****void GetDefaultCellAlignment(int\* horiz, int\* vert)**

Sets the arguments to the current default horizontal and vertical text alignment values.

Horizontal alignment will be one of wxALIGN\_LEFT, wxALIGN\_CENTRE or wxALIGN\_RIGHT.

Vertical alignment will be one of wxALIGN\_TOP, wxALIGN\_CENTRE or wxALIGN\_BOTTOM.

**wxGrid::GetDefaultCellBackgroundColour****wxColour GetDefaultCellBackgroundColour()**

Returns the current default background colour for grid cells.

**wxGrid::GetDefaultCellFont****wxFont GetDefaultCellFont()**

Returns the current default font for grid cell text.

**wxGrid::GetDefaultCellTextColour****wxColour GetDefaultCellTextColour()**

Returns the current default colour for grid cell text.

**wxGrid::GetDefaultColLabelSize****int GetDefaultColLabelSize()**

Returns the default height for column labels.

**wxGrid::GetDefaultColSize****int GetDefaultColSize()**

Returns the current default width for grid columns.

**wxGrid::GetDefaultEditor****wxGridCellEditor\* GetDefaultEditor() const**

Returns a pointer to the current default grid cell editor.

See *wxGridCellEditor* (p. 776) and the *wxGrid overview* (p. 2143) for more information about cell editors and renderers.

**wxGrid::GetDefaultEditorForCell****wxGridCellEditor\* GetDefaultEditorForCell(int row, int col) const****wxGridCellEditor\* GetDefaultEditorForCell(const wxGridCellCoords& c) const****wxGrid::GetDefaultEditorForType****wxGridCellEditor\* GetDefaultEditorForType(const wxString& typeName) const****wxGrid::GetDefaultRenderer****wxGridCellRenderer\* GetDefaultRenderer() const**

Returns a pointer to the current default grid cell renderer.

See *wxGridCellRenderer* (p. 791) and the *wxGrid overview* (p. 2143) for more information about cell editors and renderers.

**wxGrid::GetDefaultRendererForCell****wxGridCellRenderer\* GetDefaultRendererForCell(int row, int col) const****wxGrid::GetDefaultRendererForType****wxGridCellRenderer\* GetDefaultRendererForType(const wxString& typeName)  
const****wxGrid::GetDefaultRowLabelSize****int GetDefaultRowLabelSize()**

Returns the default width for the row labels.

**wxGrid::GetDefaultRowSize****int GetDefaultRowSize()**

Returns the current default height for grid rows.

**wxGrid::GetGridCursorCol**

**int GetGridCursorCol()**

Returns the current grid cell column position.

**wxGrid::GetGridCursorRow****int GetGridCursorRow()**

Returns the current grid cell row position.

**wxGrid::GetGridLineColour****wxColour GetGridLineColour()**

Returns the colour used for grid lines.

**See also**

*GetDefaultGridLinePen()* (p. 750)

**wxGrid::GetDefaultGridLinePen****wxPen GetDefaultGridLinePen()**

Returns the pen used for grid lines. This virtual function may be overridden in derived classes in order to change the appearance of grid lines. Note that currently the pen width must be 1.

**See also**

*GetColGridLinePen()* (p. 751),  
*GetRowGridLinePen()* (p. 750)

**wxGrid::GetRowGridLinePen****wxPen GetRowGridLinePen(int row)**

Returns the pen used for horizontal grid lines. This virtual function may be overridden in derived classes in order to change the appearance of individual grid line for the given row *row*.

Example:

```
// in a grid displaying music notation, use a solid black pen between
// octaves (C0=row 127, C1=row 115 etc.)
wxPen MidiGrid::GetRowGridLinePen(int row)
{
    if ( row%12 == 7 )
        return wxPen(*wxBLACK, 1, wxSOLID);
    else
        return GetDefaultGridLinePen();
}
```

**wxGrid::GetColGridLinePen****wxPen GetColGridLinePen(int col)**

Returns the pen used for vertical grid lines. This virtual function may be overridden in derived classes in order to change the appearance of individual grid lines for the given column *col*.

See *GetRowGridLinePen()* (p. 750) for an example.

**wxGrid::GridLinesEnabled****bool GridLinesEnabled()**

Returns true if drawing of grid lines is turned on, false otherwise.

**wxGrid::GetLabelBackgroundColour****wxColour GetLabelBackgroundColour()**

Returns the colour used for the background of row and column labels.

**wxGrid::GetLabelFont****wxFont GetLabelFont()**

Returns the font used for row and column labels.

**wxGrid::GetLabelTextColour****wxColour GetLabelTextColour()**

Returns the colour used for row and column label text.

**wxGrid::GetNumberCols****int GetNumberCols()**

Returns the total number of grid columns (actually the number of columns in the underlying grid table).

**wxGrid::GetNumberRows****int GetNumberRows()**

Returns the total number of grid rows (actually the number of rows in the underlying grid table).

**wxGrid::GetOrCreateCellAttr****wxGridCellAttr\* GetOrCreateCellAttr(int row, int col) const****wxGrid::GetRowMinimalAcceptableHeight****int GetRowMinimalAcceptableHeight()**

This returns the value of the lowest row width that can be handled correctly. See member *SetRowMinimalAcceptableHeight* (p. 766) for details.

**wxGrid::GetRowMinimalHeight****int GetRowMinimalHeight(int col) const****wxGrid::GetRowLabelAlignment****void GetRowLabelAlignment(int\* horiz, int\* vert)**

Sets the arguments to the current row label alignment values.

Horizontal alignment will be one of wxLEFT, wxCENTRE or wxRIGHT.  
Vertical alignment will be one of wxTOP, wxCENTRE or wxBOTTOM.

**wxPerl note:** This method takes no parameters and returns a 2-element list ( *horiz*, *vert* ).

**wxGrid::GetRowLabelSize****int GetRowLabelSize()**

Returns the current width of the row labels.

**wxGrid::GetRowLabelValue****wxString GetRowLabelValue(int row)**

Returns the specified row label. The default grid table class provides numeric row labels. If you are using a custom grid table you can override *wxGridTableBase::GetRowLabelValue* (p. 795) to provide your own labels.

**wxGrid::GetRowSize****int GetRowSize(int row)**

Returns the height of the specified row.

**wxGrid::GetScrollLineX****int GetScrollLineX() const**



Returns the number of pixels per horizontal scroll increment. The default is 15.

**See also**

*wxGrid::GetScrollLineY* (p. 753), *wxGrid::SetScrollLineX* (p. 767), *wxGrid::SetScrollLineY* (p. 767)

**wxGrid::GetScrollLineY**

**int GetScrollLineY() const**

Returns the number of pixels per vertical scroll increment. The default is 15.

**See also**

*wxGrid::GetScrollLineX* (p. 752), *wxGrid::SetScrollLineX* (p. 767), *wxGrid::SetScrollLineY* (p. 767)

**wxGrid::GetSelectionMode**

**wxGrid::wxGridSelectionMode GetSelectionMode() const**

Returns the current selection mode, see *wxGrid::SetSelectionMode* (p. 767).

**wxGrid::GetSelectedCells**

**wxGridCellCoordsArray GetSelectedCells() const**

Returns an array of singly selected cells.

**wxGrid::GetSelectedCols**

**wxArrayInt GetSelectedCols() const**

Returns an array of selected cols.

**wxGrid::GetSelectedRows**

**wxArrayInt GetSelectedRows() const**

Returns an array of selected rows.

**wxGrid::GetSelectionBackground**

**wxColour GetSelectionBackground() const**

Access or update the selection fore/back colours

**wxGrid::GetSelectionBlockTopLeft**

**wxGridCellCoordsArray GetSelectionBlockTopLeft() const**

Returns an array of the top left corners of blocks of selected cells, see *wxGrid::GetSelectionBlockBottomRight* (p. 754).

### **wxGrid::GetSelectionBlockBottomRight**

**wxGridCellCoordsArray GetSelectionBlockBottomRight() const**

Returns an array of the bottom right corners of blocks of selected cells, see *wxGrid::GetSelectionBlockTopLeft* (p. 753).

### **wxGrid::GetSelectionForeground**

**wxColour GetSelectionForeground() const**

### **wxGrid::GetTable**

**wxGridTableBase \* GetTable() const**

Returns a base pointer to the current table object.

### **wxGrid::GetViewWidth**

**int GetViewWidth()**

Returned number of whole cols visible.

### **wxGrid::HideCellEditControl**

**void HideCellEditControl()**

Hides the in-place cell edit control.

### **wxGrid::InitColWidths**

**void InitColWidths()**

Init the m\_colWidths/Rights arrays

### **wxGrid::InitRowHeights**

**void InitRowHeights()**

NB: *never* access m\_row/col arrays directly because they are created on demand, *always* use accessor functions instead!

Init the m\_rowHeights/Bottoms arrays with default values.

### **wxGrid::InsertCols**

**bool InsertCols(int pos = 0, int numCols = 1, bool updateLabels = true)**

Inserts one or more new columns into a grid with the first new column at the specified position and returns true if successful. The `updateLabels` argument is not used at present.

The sequence of actions begins with the grid object requesting the underlying grid table to insert new columns. If this is successful the table notifies the grid and the grid updates the display. For a default grid (one where you have called `wxGrid::CreateGrid` (p. 742)) this process is automatic. If you are using a custom grid table (specified with `wxGrid::SetTable` (p. 768)) then you must override `wxGridTableBase::InsertCols` (p. 795) in your derived table class.

### **wxGrid::InsertRows**

**bool InsertRows(int pos = 0, int numRows = 1, bool updateLabels = true)**

Inserts one or more new rows into a grid with the first new row at the specified position and returns true if successful. The `updateLabels` argument is not used at present.

The sequence of actions begins with the grid object requesting the underlying grid table to insert new rows. If this is successful the table notifies the grid and the grid updates the display. For a default grid (one where you have called `wxGrid::CreateGrid` (p. 742)) this process is automatic. If you are using a custom grid table (specified with `wxGrid::SetTable` (p. 768)) then you must override `wxGridTableBase::InsertRows` (p. 795) in your derived table class.

### **wxGrid::IsCellEditControlEnabled**

**bool IsCellEditControlEnabled() const**

Returns true if the in-place edit control is currently enabled.

### **wxGrid::IsCurrentCellReadOnly**

**bool IsCurrentCellReadOnly() const**

Returns true if the current cell has been set to read-only (see `wxGrid::SetReadOnly` (p. 765)).

### **wxGrid::IsEditable**

**bool IsEditable()**

Returns false if the whole grid has been set as read-only or true otherwise. See `wxGrid::EnableEditing` (p. 744) for more information about controlling the editing status of grid cells.

### **wxGrid::IsInSelection**

**bool IsInSelection(int row, int col) const**

**bool IsInSelection(const wxGridCellCoords& coords) const**

Is this cell currently selected.

### **wxGrid::IsReadOnly**

**bool IsReadOnly(int row, int col) const**

Returns true if the cell at the specified location can't be edited. See also *wxGrid::IsReadOnly* (p. 756).

### **wxGrid::IsSelection**

**bool IsSelection()**

Returns true if there are currently rows, columns or blocks of cells selected.

### **wxGrid::IsVisible**

**bool IsVisible(int row, int col, bool wholeCellVisible = true)**

**bool IsVisible(const wxGridCellCoords& coords, bool wholeCellVisible = true)**

Returns true if a cell is either wholly visible (the default) or at least partially visible in the grid window.

### **wxGrid::MakeCellVisible**

**void MakeCellVisible(int row, int col)**

**void MakeCellVisible(const wxGridCellCoords& coords)**

Brings the specified cell into the visible grid cell area with minimal scrolling. Does nothing if the cell is already visible.

### **wxGrid::MoveCursorDown**

**bool MoveCursorDown(bool expandSelection)**

Moves the grid cursor down by one row. If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

#### **Keyboard**

This function is called for Down cursor key presses or Shift+Down to expand a selection.

### **wxGrid::MoveCursorLeft**

**bool MoveCursorLeft(bool expandSelection)**

Moves the grid cursor left by one column. If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

#### **Keyboard**

This function is called for Left cursor key presses or Shift+Left to expand a selection.

### **wxGrid::MoveCursorRight**

**bool MoveCursorRight**(bool *expandSelection*)

Moves the grid cursor right by one column. If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

#### **Keyboard**

This function is called for Right cursor key presses or Shift+Right to expand a selection.

### **wxGrid::MoveCursorUp**

**bool MoveCursorUp**(bool *expandSelection*)

Moves the grid cursor up by one row. If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

#### **Keyboard**

This function is called for Up cursor key presses or Shift+Up to expand a selection.

### **wxGrid::MoveCursorDownBlock**

**bool MoveCursorDownBlock**(bool *expandSelection*)

Moves the grid cursor down in the current column such that it skips to the beginning or end of a block of non-empty cells. If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

#### **Keyboard**

This function is called for the Ctrl+Down key combination. Shift+Ctrl+Down expands a selection.

### **wxGrid::MoveCursorLeftBlock**

**bool MoveCursorLeftBlock**(bool *expandSelection*)

Moves the grid cursor left in the current row such that it skips to the beginning or end of a block of non-empty cells. If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

#### **Keyboard**

This function is called for the Ctrl+Left key combination. Shift+Ctrl+left expands a selection.

### **wxGrid::MoveCursorRightBlock**

**bool MoveCursorRightBlock**(bool *expandSelection*)

Moves the grid cursor right in the current row such that it skips to the beginning or end of a

block of non-empty cells. If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

**Keyboard**

This function is called for the Ctrl+Right key combination. Shift+Ctrl+Right expands a selection.

**wxGrid::MoveCursorUpBlock**

**bool MoveCursorUpBlock**(bool *expandSelection*)

Moves the grid cursor up in the current column such that it skips to the beginning or end of a block of non-empty cells. If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

**Keyboard**

This function is called for the Ctrl+Up key combination. Shift+Ctrl+Up expands a selection.

**wxGrid::MovePageDown**

**bool MovePageDown**()

Moves the grid cursor down by some number of rows so that the previous bottom visible row becomes the top visible row.

**Keyboard**

This function is called for PgDn keypresses.

**wxGrid::MovePageUp**

**bool MovePageUp**()

Moves the grid cursor up by some number of rows so that the previous top visible row becomes the bottom visible row.

**Keyboard**

This function is called for PgUp keypresses.

**wxGrid::RegisterDataType**

**void RegisterDataType**(const wxString& *typeName*, wxGridCellRenderer\* *renderer*, wxGridCellEditor\* *editor*)

Methods for a registry for mapping data types to Renderers/Editors

**wxGrid::SaveEditControlValue**

**void SaveEditControlValue**()

Sets the value of the current grid cell to the current in-place edit control value. This is called automatically when the grid cursor moves from the current cell to a new cell. It is

also a good idea to call this function when closing a grid since any edits to the final cell location will not be saved otherwise.

### **wxGrid::SelectAll**

**void SelectAll()**

Selects all cells in the grid.

### **wxGrid::SelectBlock**

**void SelectBlock(int topRow, int leftCol, int bottomRow, int rightCol, bool addToSelected = false)**

**void SelectBlock(const wxGridCellCoords& topLeft, const wxGridCellCoords& bottomRight, bool addToSelected = false)**

Selects a rectangular block of cells. If addToSelected is false then any existing selection will be deselected; if true the column will be added to the existing selection.

### **wxGrid::SelectCol**

**void SelectCol(int col, bool addToSelected = false)**

Selects the specified column. If addToSelected is false then any existing selection will be deselected; if true the column will be added to the existing selection.

### **wxGrid::SelectionToDeviceRect**

**wxRect SelectionToDeviceRect()**

This function returns the rectangle that encloses the selected cells in device coords and clipped to the client size of the grid window.

### **wxGrid::SelectRow**

**void SelectRow(int row, bool addToSelected = false)**

Selects the specified row. If addToSelected is false then any existing selection will be deselected; if true the row will be added to the existing selection.

### **wxGrid::SetCellAlignment**

**void SetCellAlignment(int row, int col, int horiz, int vert)**

**void SetCellAlignment(int align, int row, int col)**

Sets the horizontal and vertical alignment for grid cell text at the specified location.

Horizontal alignment should be one of wxALIGN\_LEFT, wxALIGN\_CENTRE or wxALIGN\_RIGHT.

Vertical alignment should be one of `wxALIGN_TOP`, `wxALIGN_CENTRE` or `wxALIGN_BOTTOM`.

### **`wxGrid::SetCellBackgroundColour`**

**`void SetCellBackgroundColour(int row, int col, const wxColour& colour)`**

### **`wxGrid::SetCellEditor`**

**`void SetCellEditor(int row, int col, wxGridCellEditor* editor)`**

Sets the editor for the grid cell at the specified location. The grid will take ownership of the pointer.

See *wxGridCellEditor* (p. 776) and the *wxGrid overview* (p. 2143) for more information about cell editors and renderers.

### **`wxGrid::SetCellFont`**

**`void SetCellFont(int row, int col, const wxFont& font)`**

Sets the font for text in the grid cell at the specified location.

### **`wxGrid::SetCellRenderer`**

**`void SetCellRenderer(int row, int col, wxGridCellRenderer* renderer)`**

Sets the renderer for the grid cell at the specified location. The grid will take ownership of the pointer.

See *wxGridCellRenderer* (p. 791) and the *wxGrid overview* (p. 2143) for more information about cell editors and renderers.

### **`wxGrid::SetCellTextColour`**

**`void SetCellTextColour(int row, int col, const wxColour& colour)`**

**`void SetCellTextColour(const wxColour& val, int row, int col)`**

**`void SetCellTextColour(const wxColour& colour)`**

Sets the text colour for the grid cell at the specified location.

### **`wxGrid::SetCellValue`**

**`void SetCellValue(int row, int col, const wxString& s)`**

**`void SetCellValue(const wxGridCellCoords& coords, const wxString& s)`**

**`void SetCellValue(const wxString& val, int row, int col)`**



Sets the string value for the cell at the specified location. For simple applications where a grid object automatically uses a default grid table of string values you use this function together with *wxGrid::GetCellValue* (p. 746) to access cell values.

For more complex applications where you have derived your own grid table class that contains various data types (e.g. numeric, boolean or user-defined custom types) then you only use this function for those cells that contain string values.

The last form is for backward compatibility only.

See *wxGridTableBase::CanSetValueAs* (p. 794) and the *wxGrid overview* (p. 2143) for more information.

### **wxGrid::SetColAttr**

**void SetColAttr(int col, wxGridCellAttr\* attr)**

Sets the cell attributes for all cells in the specified column.

For more information about controlling grid cell attributes see the *wxGridCellAttr* (p. 769) cell attribute class and the *wxGrid classes overview* (p. 2143).

### **wxGrid::SetColFormatBool**

**void SetColFormatBool(int col)**

Sets the specified column to display boolean values. *wxGrid* displays boolean values with a checkbox.

### **wxGrid::SetColFormatNumber**

**void SetColFormatNumber(int col)**

Sets the specified column to display integer values.

### **wxGrid::SetColFormatFloat**

**void SetColFormatFloat(int col, int width = -1, int precision = -1)**

Sets the specified column to display floating point values with the given width and precision.

### **wxGrid::SetColFormatCustom**

**void SetColFormatCustom(int col, const wxString& typeName)**

Sets the specified column to display data in a custom format. See the *wxGrid overview* (p. 2143) for more information on working with custom data types.

### **wxGrid::SetColLabelAlignment**

**void SetColLabelAlignment(int *horiz*, int *vert*)**

Sets the horizontal and vertical alignment of column label text.

Horizontal alignment should be one of `wxALIGN_LEFT`, `wxALIGN_CENTRE` or `wxALIGN_RIGHT`.

Vertical alignment should be one of `wxALIGN_TOP`, `wxALIGN_CENTRE` or `wxALIGN_BOTTOM`.

**wxGrid::SetColLabelSize**

**void SetColLabelSize(int *height*)**

Sets the height of the column labels.

If *height* equals to `wxGRID_AUTOSIZE` then height is calculated automatically so that no label is truncated. Note that this could be slow for a large table. This flag is new since wxWidgets version 2.8.8.

**wxGrid::SetColLabelValue**

**void SetColLabelValue(int *col*, const wxString& *value*)**

Set the value for the given column label. If you are using a derived grid table you must override `wxGridTableBase::SetColLabelValue` (p. 795) for this to have any effect.

**wxGrid::SetColMinimalWidth**

**void SetColMinimalWidth(int *col*, int *width*)**

Sets the minimal width for the specified column. This should normally be called when creating the grid because it will not resize a column that is already narrower than the minimal width. The width argument must be higher than the minimal acceptable column width, see `wxGrid::GetColMinimalAcceptableWidth` (p. 747).

**wxGrid::SetColMinimalAcceptableWidth**

**void SetColMinimalAcceptableWidth(int *width*)**

This modifies the minimum column width that can be handled correctly. Specifying a low value here allows smaller grid cells to be dealt with correctly. Specifying a value here which is much smaller than the actual minimum size will incur a performance penalty in the functions which perform grid cell index lookup on the basis of screen coordinates. This should normally be called when creating the grid because it will not resize existing columns with sizes smaller than the value specified here.

**wxGrid::SetColPos**

**void SetColPos(int *colID*, int *newPos*)**

Sets the position of the specified column.

### **wxGrid::SetColSize**

**void SetColSize(int col, int width)**

Sets the width of the specified column.

This function does not refresh the grid. If you are calling it outside of a `BeginBatch / EndBatch` block you can use `wxGrid::ForceRefresh` (p. 745) to see the changes.

Automatically sizes the column to fit its contents. If `setAsMin` is true the calculated width will also be set as the minimal width for the column.

#### **Note**

`wxGrid` sets up arrays to store individual row and column sizes when non-default sizes are used. The memory requirements for this could become prohibitive if your grid is very large.

### **wxGrid::SetDefaultCellAlignment**

**void SetDefaultCellAlignment(int horiz, int vert)**

Sets the default horizontal and vertical alignment for grid cell text.

Horizontal alignment should be one of `wxALIGN_LEFT`, `wxALIGN_CENTRE` or `wxALIGN_RIGHT`.

Vertical alignment should be one of `wxALIGN_TOP`, `wxALIGN_CENTRE` or `wxALIGN_BOTTOM`.

### **wxGrid::SetDefaultCellBackgroundColour**

**void SetDefaultCellBackgroundColour(const wxColour& colour)**

Sets the default background colour for grid cells.

### **wxGrid::SetDefaultCellFont**

**void SetDefaultCellFont(const wxFont& font)**

Sets the default font to be used for grid cell text.

### **wxGrid::SetDefaultCellTextColour**

**void SetDefaultCellTextColour(const wxColour& colour)**

Sets the current default colour for grid cell text.

### **wxGrid::SetDefaultEditor**

**void SetDefaultEditor(wxGridCellEditor\* editor)**

Sets the default editor for grid cells. The grid will take ownership of the pointer.

See *wxGridCellEditor* (p. 776) and the *wxGrid overview* (p. 2143) for more information about cell editors and renderers.

### **wxGrid::SetDefaultRenderer**

**void SetDefaultRenderer(wxGridCellRenderer\* *renderer*)**

Sets the default renderer for grid cells. The grid will take ownership of the pointer.

See *wxGridCellRenderer* (p. 791) and the *wxGrid overview* (p. 2143) for more information about cell editors and renderers.

### **wxGrid::SetDefaultColSize**

**void SetDefaultColSize(int *width*, bool *resizeExistingCols* = false)**

Sets the default width for columns in the grid. This will only affect columns subsequently added to the grid unless *resizeExistingCols* is true.

### **wxGrid::SetDefaultRowSize**

**void SetDefaultRowSize(int *height*, bool *resizeExistingRows* = false)**

Sets the default height for rows in the grid. This will only affect rows subsequently added to the grid unless *resizeExistingRows* is true.

### **wxGrid::SetGridCursor**

**void SetGridCursor(int *row*, int *col*)**

Set the grid cursor to the specified cell. This function calls *wxGrid::MakeCellVisible* (p. 756).

### **wxGrid::SetGridLineColour**

**void SetGridLineColour(const wxColour& *colour*)**

Sets the colour used to draw grid lines.

### **wxGrid::SetLabelBackgroundColour**

**void SetLabelBackgroundColour(const wxColour& *colour*)**

Sets the background colour for row and column labels.

### **wxGrid::SetLabelFont**

**void SetLabelFont(const wxFont& *font*)**

Sets the font for row and column labels.

### **wxGrid::SetLabelTextColour**

**void SetLabelTextColour(const wxColour& colour)**

Sets the colour for row and column label text.

### **wxGrid::SetMargins**

**void SetMargins(int extraWidth, int extraHeight)**

A grid may occupy more space than needed for its rows/columns. This function allows to set how big this extra space is

### **wxGrid::SetOrCalcColumnSizes**

**int SetOrCalcColumnSizes(bool calcOnly, bool setAsMin = true)**

Common part of AutoSizeColumn/Row() and GetBestSize()

### **wxGrid::SetOrCalcRowSizes**

**int SetOrCalcRowSizes(bool calcOnly, bool setAsMin = true)**

### **wxGrid::SetReadOnly**

**void SetReadOnly(int row, int col, bool isReadOnly = true)**

Makes the cell at the specified location read-only or editable. See also *wxGrid::IsReadOnly* (p. 756).

### **wxGrid::SetRowAttr**

**void SetRowAttr(int row, wxGridCellAttr\* attr)**

Sets the cell attributes for all cells in the specified row. See the *wxGridCellAttr* (p. 769) class for more information about controlling cell attributes.

### **wxGrid::SetRowLabelAlignment**

**void SetRowLabelAlignment(int horiz, int vert)**

Sets the horizontal and vertical alignment of row label text.

Horizontal alignment should be one of *wxALIGN\_LEFT*, *wxALIGN\_CENTRE* or *wxALIGN\_RIGHT*.

Vertical alignment should be one of *wxALIGN\_TOP*, *wxALIGN\_CENTRE* or *wxALIGN\_BOTTOM*.

**wxGrid::SetRowLabelSize****void SetRowLabelSize(int width)**

Sets the width of the row labels.

If *width* equals `wxGRID_AUTOSIZE` then width is calculated automatically so that no label is truncated. Note that this could be slow for a large table. This flag is new since wxWidgets version 2.8.8.

**wxGrid::SetRowLabelValue****void SetRowLabelValue(int row, const wxString& value)**

Set the value for the given row label. If you are using a derived grid table you must override `wxGridTableBase::SetRowLabelValue` (p. 795) for this to have any effect.

**wxGrid::SetRowMinimalHeight****void SetRowMinimalHeight(int row, int height)**

Sets the minimal height for the specified row. This should normally be called when creating the grid because it will not resize a row that is already shorter than the minimal height. The height argument must be higher than the minimal acceptable row height, see `wxGrid::GetRowMinimalAcceptableHeight` (p. 752).

**wxGrid::SetRowMinimalAcceptableHeight****void SetRowMinimalAcceptableHeight(int height)**

This modifies the minimum row width that can be handled correctly. Specifying a low value here allows smaller grid cells to be dealt with correctly. Specifying a value here which is much smaller than the actual minimum size will incur a performance penalty in the functions which perform grid cell index lookup on the basis of screen coordinates. This should normally be called when creating the grid because it will not resize existing rows with sizes smaller than the value specified here.

**wxGrid::SetRowSize****void SetRowSize(int row, int height)**

Sets the height of the specified row.

This function does not refresh the grid. If you are calling it outside of a `BeginBatch / EndBatch` block you can use `wxGrid::ForceRefresh` (p. 745) to see the changes.

Automatically sizes the column to fit its contents. If `setAsMin` is true the calculated width will also be set as the minimal width for the column.

**Note**

`wxGrid` sets up arrays to store individual row and column sizes when non-default sizes are used. The memory requirements for this could become prohibitive if your grid is very large.

### **`wxGrid::SetScrollLineX`**

**`void SetScrollLineX(int x)`**

Sets the number of pixels per horizontal scroll increment. The default is 15. Sometimes `wxGrid` has trouble setting the scrollbars correctly due to rounding errors: setting this to 1 can help.

#### **See also**

`wxGrid::GetScrollLineX` (p. 752), `wxGrid::GetScrollLineY` (p. 753), `wxGrid::SetScrollLineY` (p. 767)

### **`wxGrid::SetScrollLineY`**

**`void SetScrollLineY(int y)`**

Sets the number of pixels per vertical scroll increment. The default is 15. Sometimes `wxGrid` has trouble setting the scrollbars correctly due to rounding errors: setting this to 1 can help.

#### **See also**

`wxGrid::GetScrollLineX` (p. 752), `wxGrid::GetScrollLineY` (p. 753), `wxGrid::SetScrollLineX` (p. 767)

### **`wxGrid::SetSelectionBackground`**

**`void SetSelectionBackground(const wxColour& c)`**

### **`wxGrid::SetSelectionForeground`**

**`void SetSelectionForeground(const wxColour& c)`**

### **`wxGrid::SetSelectionMode`**

**`void SetSelectionMode(wxGrid::wxGridSelectionModes selmode)`**

Set the selection behaviour of the grid.

#### **Parameters**

`wxGrid::wxGridSelectCells`

The default mode where individual cells are selected.

`wxGrid::wxGridSelectRows`

Selections will consist of whole rows.

*wxGrid::wxGridSelectColumns*

Selections will consist of whole columns.

### **wxGrid::SetTable**

**bool SetTable(wxGridTableBase\* table, bool takeOwnership = false,  
wxGrid::wxGridSelectionMode selmode = wxGrid::wxGridSelectCells)**

Passes a pointer to a custom grid table to be used by the grid. This should be called after the grid constructor and before using the grid object. If takeOwnership is set to true then the table will be deleted by the wxGrid destructor.

Use this function instead of *wxGrid::CreateGrid* (p. 742) when your application involves complex or non-string data or data sets that are too large to fit wholly in memory.

### **wxGrid::ShowCellEditControl**

**void ShowCellEditControl()**

Displays the in-place cell edit control for the current cell.

### **wxGrid::XToCol**

**int XToCol(int x, bool clipToMinMax = false)**

#### **Parameters**

The x position to evaluate.

*clipToMinMax*

If true, rather than returning wxNOT\_FOUND, it returns either the first or last column depending on whether x is too far to the left or right respectively.

**Return value** The grid column that corresponds to the logical x coordinate.  
Returns wxNOT\_FOUND if there is no column at the x position.

### **wxGrid::XToEdgeOfCol**

**int XToEdgeOfCol(int x)**

Returns the column whose right hand edge is close to the given logical x position. If no column edge is near to this position wxNOT\_FOUND is returned.

### **wxGrid::YToEdgeOfRow**

**int YToEdgeOfRow(int y)**

Returns the row whose bottom edge is close to the given logical y position. If no row edge is near to this position wxNOT\_FOUND is returned.



**wxGrid::YToRow****int YToRow(int y)**

Returns the grid row that corresponds to the logical y coordinate. Returns `wxNOT_FOUND` if there is no row at the y position.

**wxGridCellAttr**

This class can be used to alter the cells' appearance in the grid by changing their colour/font/... from default. An object of this class may be returned by `wxGridTable::GetAttr()`.

**Derived from**

No base class

**Include files**

<wx/grid.h>

**wxGridCellAttr::wxGridCellAttr****wxGridCellAttr()**

Default constructor.

**wxGridCellAttr(const wxColour& colText, const wxColour& colBack, const wxFont& font, int hAlign, int vAlign)**

Constructor specifying some of the often used attributes.

**wxGridCellAttr::Clone****wxGridCellAttr\* Clone() const**

Creates a new copy of this object.

**wxGridCellAttr::IncRef****void IncRef()**

This class is ref counted: it is created with ref count of 1, so calling `DecRef()` once will delete it. Calling `IncRef()` allows to lock it until the matching `DecRef()` is called

**wxGridCellAttr::DecRef****void DecRef()**

**wxGridCellAttr::SetTextColour****void SetTextColour(const wxColour& colText)**

Sets the text colour.

**wxGridCellAttr::SetBackgroundColour****void SetBackgroundColour(const wxColour& colBack)**

Sets the background colour.

**wxGridCellAttr::SetFont****void SetFont(const wxFont& font)**

Sets the font.

**wxGridCellAttr::SetAlignment****void SetAlignment(int hAlign, int vAlign)**

Sets the alignment. *hAlign* can be one of `wxALIGN_LEFT`, `wxALIGN_CENTRE` or `wxALIGN_RIGHT` and *vAlign* can be one of `wxALIGN_TOP`, `wxALIGN_CENTRE` or `wxALIGN_BOTTOM`.

**wxGridCellAttr::SetReadOnly****void SetReadOnly(bool isReadOnly = true)****wxGridCellAttr::SetRenderer****void SetRenderer(wxGridCellRenderer\* renderer)**

takes ownership of the pointer

**wxGridCellAttr::SetEditor****void SetEditor(wxGridCellEditor\* editor)****wxGridCellAttr::HasTextColour****bool HasTextColour() const**

accessors

**wxGridCellAttr::HasBackgroundColour****bool HasBackgroundColour() const**

**wxGridCellAttr::HasFont**

**bool HasFont() const**

**wxGridCellAttr::HasAlignment**

**bool HasAlignment() const**

**wxGridCellAttr::HasRenderer**

**bool HasRenderer() const**

**wxGridCellAttr::HasEditor**

**bool HasEditor() const**

**wxGridCellAttr::GetTextColour**

**const wxColour& GetTextColour() const**

**wxGridCellAttr::GetBackgroundColour**

**const wxColour& GetBackgroundColour() const**

**wxGridCellAttr::GetFont**

**const wxFont& GetFont() const**

**wxGridCellAttr::GetAlignment**

**void GetAlignment(int\* hAlign, int\* vAlign) const**

**wxPerl note:** This method takes no parameters and returns a 2-element list ( *hAlign*, *vAlign* ).

See *SetAlignment* (p. 770) for the returned values.

**wxGridCellAttr::GetRenderer**

**wxGridCellRenderer\* GetRenderer(wxGrid\* grid, int row, int col) const**

**wxGridCellAttr::GetEditor**

**wxGridCellEditor\* GetEditor(wxGrid\* grid, int row, int col) const**

**wxGridCellAttr::IsReadOnly**

**bool IsReadOnly() const**

**wxGridCellAttr::SetDefAttr****void SetDefAttr(wxGridCellAttr\* defAttr)****wxGridBagSizer**

A *wxSizer* (p. 1444) that can lay out items in a virtual grid like a *wxFlexGridSizer* (p. 652) but in this case explicit positioning of the items is allowed using *wxGBPosition* (p. 705), and items can optionally span more than one row and/or column using *wxGBSpan* (p. 708).

**Derived from**

*wxFlexGridSizer* (p. 652)  
*wxGridSizer* (p. 797)  
*wxSizer* (p. 1444)  
*wxObject* (p. 1148)

**Include files**

<wx/gbsizer.h>

**wxGridBagSizer::wxGridBagSizer****wxGridBagSizer(int vgap = 0, int hgap = 0)**

Constructor, with optional parameters to specify the gap between the rows and columns.

**wxGridBagSizer::Add****wxSizerItem\* Add(wxWindow\* window, const wxGBPosition& pos, const wxGBSpan& span = wxDefaultSpan, int flag = 0, int border = 0, wxObject\* userData = NULL)****wxSizerItem\* Add(wxSizer\* sizer, const wxGBPosition& pos, const wxGBSpan& span = wxDefaultSpan, int flag = 0, int border = 0, wxObject\* userData = NULL)****wxSizerItem\* Add(int width, int height, const wxGBPosition& pos, const wxGBSpan& span = wxDefaultSpan, int flag = 0, int border = 0, wxObject\* userData = NULL)****wxSizerItem\* Add(wxGBSizerItem\* item)**

The Add methods return a valid pointer if the item was successfully placed at the given position, NULL if something was already there.

**wxGridBagSizer::CalcMin****wxSize CalcMin()**

Called when the managed size of the sizer is needed or when layout needs done.

### **wxGridBagSizer::CheckForIntersection**

**bool CheckForIntersection(wxGBSizerItem\* item, wxGBSizerItem\* excludeItem = NULL)**

**bool CheckForIntersection(const wxGBPosition& pos, const wxGBSpan& span, wxGBSizerItem\* excludeItem = NULL)**

Look at all items and see if any intersect (or would overlap) the given item. Returns true if so, false if there would be no overlap. If an excludeItem is given then it will not be checked for intersection, for example it may be the item we are checking the position of.

### **wxGridBagSizer::FindItem**

**wxGBSizerItem\* FindItem(wxWindow\* window)**

**wxGBSizerItem\* FindItem(wxSizer\* sizer)**

Find the sizer item for the given window or subsizer, returns NULL if not found. (non-recursive)

### **wxGridBagSizer::FindItemAtPoint**

**wxGBSizerItem\* FindItemAtPoint(const wxPoint& pt)**

Return the sizer item located at the point given in pt, or NULL if there is no item at that point. The (x,y) coordinates in pt correspond to the client coordinates of the window using the sizer for layout. (non-recursive)

### **wxGridBagSizer::FindItemAtPosition**

**wxGBSizerItem\* FindItemAtPosition(const wxGBPosition& pos)**

Return the sizer item for the given grid cell, or NULL if there is no item at that position. (non-recursive)

### **wxGridBagSizer::FindItemWithData**

**wxGBSizerItem\* FindItemWithData(const wxObject\* userData)**

Return the sizer item that has a matching user data (it only compares pointer values) or NULL if not found. (non-recursive)

### **wxGridBagSizer::GetCellSize**

**wxSize GetCellSize(int row, int col) const**

Get the size of the specified cell, including hgap and vgap. Only valid after a Layout.

**wxGridBagSizer::GetEmptyCellSize****wxSize GetEmptyCellSize() const**

Get the size used for cells in the grid with no item.

**wxGridBagSizer::GetItemPosition****wxGBPosition GetItemPosition(wxWindow\* window)****wxGBPosition GetItemPosition(wxSizer\* sizer)****wxGBPosition GetItemPosition(size\_t index)**

Get the grid position of the specified item.

**wxGridBagSizer::GetItemSpan****wxGBSpan GetItemSpan(wxWindow\* window)****wxGBSpan GetItemSpan(wxSizer\* sizer)****wxGBSpan GetItemSpan(size\_t index)**

Get the row/col spanning of the specified item

**wxGridBagSizer::RecalcSizes****void RecalcSizes()**

Called when the managed size of the sizer is needed or when layout needs done.

**wxGridBagSizer::SetEmptyCellSize****void SetEmptyCellSize(const wxSize& sz)**

Set the size used for cells in the grid with no item.

**wxGridBagSizer::SetItemPosition****bool SetItemPosition(wxWindow\* window, const wxGBPosition& pos)****bool SetItemPosition(wxSizer\* sizer, const wxGBPosition& pos)****bool SetItemPosition(size\_t index, const wxGBPosition& pos)**

Set the grid position of the specified item. Returns true on success. If the move is not allowed (because an item is already there) then false is returned.

**wxGridBagSizer::SetItemSpan**

**bool SetItemSpan(wxWindow\* window, const wxGBSpan& span)**

**bool SetItemSpan(wxSizer\* sizer, const wxGBSpan& span)**

**bool SetItemSpan(size\_t index, const wxGBSpan& span)**

Set the row/col spanning of the specified item. Returns true on success. If the move is not allowed (because an item is already there) then false is returned.

## wxGridCellBoolEditor

The editor for boolean data.

### Derived from

*wxGridCellEditor* (p. 776)

### See also

*wxGridCellEditor* (p. 776), *wxGridCellFloatEditor* (p. 778), *wxGridCellNumberEditor* (p. 779), *wxGridCellTextEditor* (p. 780), *wxGridCellChoiceEditor* (p. 776)

### Include files

<wx/grid.h>

## wxGridCellBoolEditor::wxGridCellBoolEditor

**wxGridCellBoolEditor()**

Default constructor.

## wxGridCellBoolEditor::IsTrueValue

**static bool IsTrueValue(const wxString& value)**

Returns `true` if the given *value* is equal to the string representation of the truth value we currently use (see *UseStringValue* (p. 775)).

## wxGridCellBoolEditor::UseStringValue

**static void UseStringValue(const wxString& valueTrue = \_T("1"), const wxString& valueFalse = \_T(""))**

This method allows to customize the values returned by `GetValue()` method for the cell using this editor. By default, the default values of the arguments are used, i.e. "1" is returned if the cell is checked and an empty string otherwise, using this method allows to change this.

## **wxGridCellChoiceEditor**

The editor for string data allowing to choose from a list of strings.

### **Derived from**

*wxGridCellEditor* (p. 776)

### **See also**

*wxGridCellEditor* (p. 776), *wxGridCellFloatEditor* (p. 778), *wxGridCellBoolEditor* (p. 775), *wxGridCellTextEditor* (p. 780), *wxGridCellNumberEditor* (p. 779)

## **wxGridCellChoiceEditor::wxGridCellChoiceEditor**

**wxGridCellChoiceEditor**(*size\_t count = 0*, *const wxString choices[] = NULL*, *bool allowOthers = false*)

**wxGridCellChoiceEditor**(*const wxArrayString& choices*, *bool allowOthers = false*)  
*count*

Number of strings from which the user can choose.

*choices*

An array of strings from which the user can choose.

*allowOthers*

If *allowOthers* if true, the user can type a string not in *choices* array.

## **wxGridCellChoiceEditor::SetParameters**

**void SetParameters**(*const wxString& params*)

Parameters string format is "item1[,item2[...itemN]]"

## **wxGridCellEditor**

This class is responsible for providing and manipulating the in-place edit controls for the grid. Instances of *wxGridCellEditor* (actually, instances of derived classes since it is an abstract class) can be associated with the cell attributes for individual cells, rows, columns, or even for the entire grid.

### **Derived from**

*wxGridCellWorker*

### **See also**



*wxGridCellTextEditor* (p. 780), *wxGridCellFloatEditor* (p. 778), *wxGridCellBoolEditor* (p. 775), *wxGridCellNumberEditor* (p. 779), *wxGridCellChoiceEditor* (p. 776)

**Include files**

<wx/grid.h>

**wxGridCellEditor::wxGridCellEditor**

**wxGridCellEditor()**

**wxGridCellEditor::IsCreated**

**bool IsCreated()**

**wxGridCellEditor::Create**

**void Create(wxWindow\* parent, wxWindowID id, wxEvtHandler\* evtHandler)**

Creates the actual edit control.

**wxGridCellEditor::SetSize**

**void SetSize(const wxRect& rect)**

Size and position the edit control.

**wxGridCellEditor::Show**

**void Show(bool show, wxGridCellAttr\* attr = NULL)**

Show or hide the edit control, use the specified attributes to set colours/fonts for it.

**wxGridCellEditor::PaintBackground**

**void PaintBackground(const wxRect& rectCell, wxGridCellAttr\* attr)**

Draws the part of the cell not occupied by the control: the base class version just fills it with background colour from the attribute.

**wxGridCellEditor::BeginEdit**

**void BeginEdit(int row, int col, wxGrid\* grid)**

Fetch the value from the table and prepare the edit control to begin editing. Set the focus to the edit control.

**wxGridCellEditor::EndEdit**

**bool EndEdit(int row, int col, wxGrid\* grid)**

Complete the editing of the current cell. Returns true if the value has changed. If necessary, the control may be destroyed.

**wxGridCellEditor::Reset**

**void Reset()**

Reset the value in the control back to its starting value.

**wxGridCellEditor::StartingKey**

**void StartingKey(wxKeyEvent& event)**

If the editor is enabled by pressing keys on the grid, this will be called to let the editor do something about that first key if desired.

**wxGridCellEditor::StartingClick**

**void StartingClick()**

If the editor is enabled by clicking on the cell, this method will be called.

**wxGridCellEditor::HandleReturn**

**void HandleReturn(wxKeyEvent& event)**

Some types of controls on some platforms may need some help with the Return key.

**wxGridCellEditor::Destroy**

**void Destroy()**

Final cleanup.

**wxGridCellEditor::Clone**

**wxGridCellEditor\* Clone() const**

Create a new object which is the copy of this one.

**wxGridCellEditor::~wxGridCellEditor**

**~wxGridCellEditor()**

The dtor is private because only DecRef() can delete us.

**wxGridCellFloatEditor**

The editor for floating point numbers data.

**Derived from**

*wxGridCellTextEditor* (p. 780)

*wxGridCellEditor* (p. 776)

**See also**

*wxGridCellEditor* (p. 776), *wxGridCellNumberEditor* (p. 779), *wxGridCellBoolEditor* (p. 775), *wxGridCellTextEditor* (p. 780), *wxGridCellChoiceEditor* (p. 776)

**Include files**

<wx/grid.h>

**wxGridCellFloatEditor::wxGridCellFloatEditor**

**wxGridCellFloatEditor**(int *width* = -1, int *precision* = -1)

*width*

Minimum number of characters to be shown.

*precision*

Number of digits after the decimal dot.

**wxGridCellFloatEditor::SetParameters**

**void SetParameters**(const wxString& *params*)

Parameters string format is "width,precision"

**wxGridCellNumberEditor**

The editor for numeric integer data.

**Derived from**

*wxGridCellTextEditor* (p. 780)

*wxGridCellEditor* (p. 776)

**See also**

*wxGridCellEditor* (p. 776), *wxGridCellFloatEditor* (p. 778), *wxGridCellBoolEditor* (p. 775), *wxGridCellTextEditor* (p. 780), *wxGridCellChoiceEditor* (p. 776)

**Include files**

<wx/grid.h>

**wxGridCellNumberEditor::wxGridCellNumberEditor****wxGridCellNumberEditor**(int *min* = -1, int *max* = -1)

Allows to specify the range for acceptable data; if *min* == *max* == -1, no range checking is done

**wxGridCellNumberEditor::GetString****wxString** GetString() const

String representation of the value.

**wxGridCellNumberEditor::HasRange****bool** HasRange() const

If the return value is true, the editor uses a wxSpinCtrl to get user input, otherwise it uses a wxTextCtrl.

**wxGridCellNumberEditor::SetParameters****void** SetParameters(const wxString& *params*)

Parameters string format is "min,max".

**wxGridCellTextEditor**

The editor for string/text data.

**Derived from**

*wxGridCellEditor* (p. 776)

**See also**

*wxGridCellEditor* (p. 776), *wxGridCellFloatEditor* (p. 778), *wxGridCellBoolEditor* (p. 775), *wxGridCellNumberEditor* (p. 779), *wxGridCellChoiceEditor* (p. 776)

**Include files**

<wx/grid.h>

**wxGridCellTextEditor::wxGridCellTextEditor****wxGridCellTextEditor**()

Default constructor.

### **wxGridCellTextEditor::SetParameters**

**void SetParameters(const wxString& params)**

The parameters string format is "n" where n is a number representing the maximum width.

## **wxGridEditorCreatedEvent**

### **Derived from**

*wxCommandEvent* (p. 250)

*wxEvent* (p. 572)

*wxObject* (p. 1148)

### **Event handling**

The event handler for the following functions takes a *wxGridEditorCreatedEvent* (p. 781) parameter. The ...\_CMD\_... variants also take a window identifier.

**EVT\_GRID\_EDITOR\_CREATED(func)** The editor for a cell was created. Processes a wxEVT\_GRID\_EDITOR\_CREATED.

**EVT\_GRID\_CMD\_EDITOR\_CREATED(id, func)** The editor for a cell was created; variant taking a window identifier. Processes a wxEVT\_GRID\_EDITOR\_CREATED.

### **Include files**

<wx/grid.h>

### **wxGridEditorCreatedEvent::wxGridEditorCreatedEvent**

**wxGridEditorCreatedEvent()**

Default constructor.

**wxGridEditorCreatedEvent(int id, wxEventType type, wxObject\* obj, int row, int col, wxControl\* ctrl)**

### **wxGridEditorCreatedEvent::GetCol**

**int GetCol()**

Returns the column at which the event occurred.

### **wxGridEditorCreatedEvent::GetControl**

**wxControl\* GetControl()**

Returns the edit control.

**wxGridEditorCreatedEvent::GetRow****int GetRow()**

Returns the row at which the event occurred.

**wxGridEditorCreatedEvent::SetCol****void SetCol(int col)**

Sets the column at which the event occurred.

**wxGridEditorCreatedEvent::SetControl****void SetControl(wxControl\* ctrl)**

Sets the edit control.

**wxGridEditorCreatedEvent::SetRow****void SetRow(int row)**

Sets the row at which the event occurred.

**wxGridEvent**

This event class contains information about various grid events.

**Derived from**

*wxNotifyEvent* (p. 1146)  
*wxCommandEvent* (p. 250)  
*wxEvent* (p. 572)  
*wxObject* (p. 1148)

**Include files**

<wx/grid.h>

**Event handling**

The event handler for the following functions takes a *wxGridEvent* (p. 782) parameter. The ...\_CMD\_... variants also take a window identifier.

**EVT\_GRID\_CELL\_LEFT\_CLICK(func)** The user clicked a cell with the left mouse button.  
Processes a  
wxEVT\_GRID\_CELL\_LEFT\_CLICK.

**EVT\_GRID\_CELL\_RIGHT\_CLICK(func)** The user clicked a cell with the right mouse

- button. Processes a  
wxEVT\_GRID\_CELL\_RIGHT\_CLICK.
- EVT\_GRID\_CELL\_LEFT\_DCLICK(func)** The user double-clicked a cell with the left mouse button. Processes a  
wxEVT\_GRID\_CELL\_LEFT\_DCLICK.
- EVT\_GRID\_CELL\_RIGHT\_DCLICK(func)** The user double-clicked a cell with the right mouse button. Processes a  
wxEVT\_GRID\_CELL\_RIGHT\_DCLICK.
- EVT\_GRID\_LABEL\_LEFT\_CLICK(func)** The user clicked a label with the left mouse button. Processes a  
wxEVT\_GRID\_LABEL\_LEFT\_CLICK.
- EVT\_GRID\_LABEL\_RIGHT\_CLICK(func)** The user clicked a label with the right mouse button. Processes a  
wxEVT\_GRID\_LABEL\_RIGHT\_CLICK.
- EVT\_GRID\_LABEL\_LEFT\_DCLICK(func)** The user double-clicked a label with the left mouse button. Processes a  
wxEVT\_GRID\_LABEL\_LEFT\_DCLICK.
- EVT\_GRID\_LABEL\_RIGHT\_DCLICK(func)** The user double-clicked a label with the right mouse button. Processes a  
wxEVT\_GRID\_LABEL\_RIGHT\_DCLICK.
- EVT\_GRID\_CELL\_CHANGE(func)** The user changed the data in a cell. Processes a  
wxEVT\_GRID\_CELL\_CHANGE.
- EVT\_GRID\_SELECT\_CELL(func)** The user moved to, and selected a cell. Processes a  
wxEVT\_GRID\_SELECT\_CELL.
- EVT\_GRID\_EDITOR\_HIDDEN(func)** The editor for a cell was hidden. Processes a  
wxEVT\_GRID\_EDITOR\_HIDDEN.
- EVT\_GRID\_EDITOR\_SHOWN(func)** The editor for a cell was shown. Processes a  
wxEVT\_GRID\_EDITOR\_SHOWN.
- EVT\_GRID\_CMD\_CELL\_LEFT\_CLICK(id, func)** The user clicked a cell with the left mouse button; variant taking a window identifier. Processes a  
wxEVT\_GRID\_CELL\_LEFT\_CLICK.
- EVT\_GRID\_CMD\_CELL\_RIGHT\_CLICK(id, func)** The user clicked a cell with the right mouse button; variant taking a window identifier. Processes a  
wxEVT\_GRID\_CELL\_RIGHT\_CLICK.
- EVT\_GRID\_CMD\_CELL\_LEFT\_DCLICK(id, func)** The user double-clicked a cell with the left mouse button; variant taking a window identifier. Processes a  
wxEVT\_GRID\_CELL\_LEFT\_DCLICK.

**EVT\_GRID\_CMD\_CELL\_RIGHT\_DCLICK(id, func)** The user double-clicked a cell with the right mouse button; variant taking a window identifier. Processes a `wxEVT_GRID_CELL_RIGHT_DCLICK`.

**EVT\_GRID\_CMD\_LABEL\_LEFT\_CLICK(id, func)** The user clicked a label with the left mouse button; variant taking a window identifier. Processes a `wxEVT_GRID_LABEL_LEFT_CLICK`.

**EVT\_GRID\_CMD\_LABEL\_RIGHT\_CLICK(id, func)** The user clicked a label with the right mouse button; variant taking a window identifier. Processes a `wxEVT_GRID_LABEL_RIGHT_CLICK`.

**EVT\_GRID\_CMD\_LABEL\_LEFT\_DCLICK(id, func)** The user double-clicked a label with the left mouse button; variant taking a window identifier. Processes a `wxEVT_GRID_LABEL_LEFT_DCLICK`.

**EVT\_GRID\_CMD\_LABEL\_RIGHT\_DCLICK(id, func)** The user double-clicked a label with the right mouse button; variant taking a window identifier. Processes a `wxEVT_GRID_LABEL_RIGHT_DCLICK`.

**EVT\_GRID\_CMD\_CELL\_CHANGE(id, func)** The user changed the data in a cell; variant taking a window identifier. Processes a `wxEVT_GRID_CELL_CHANGE`.

**EVT\_GRID\_CMD\_SELECT\_CELL(id, func)** The user moved to, and selected a cell; variant taking a window identifier. Processes a `wxEVT_GRID_SELECT_CELL`.

**EVT\_GRID\_CMD\_EDITOR\_HIDDEN(id, func)** The editor for a cell was hidden; variant taking a window identifier. Processes a `wxEVT_GRID_EDITOR_HIDDEN`.

**EVT\_GRID\_CMD\_EDITOR\_SHOWN(id, func)** The editor for a cell was shown; variant taking a window identifier. Processes a `wxEVT_GRID_EDITOR_SHOWN`.

## **wxGridEvent::wxGridEvent**

### **wxGridEvent()**

Default constructor.

**wxGridEvent(int id, wxEventType type, wxObject\* obj, int row = -1, int col = -1, int x = -1, int y = -1, bool sel = true, bool control = false, bool shift = false, bool alt = false, bool meta = false)**

### **Parameters**



**wxGridEvent::AltDown****bool AltDown()**

Returns true if the Alt key was down at the time of the event.

**wxGridEvent::ControlDown****bool ControlDown()**

Returns true if the Control key was down at the time of the event.

**wxGridEvent::GetCol****int GetCol()**

Column at which the event occurred.

**wxGridEvent::GetPosition****wxPoint GetPosition()**

Position in pixels at which the event occurred.

**wxGridEvent::GetRow****int GetRow()**

Row at which the event occurred.

**wxGridEvent::MetaDown****bool MetaDown()**

Returns true if the Meta key was down at the time of the event.

**wxGridEvent::Selecting****bool Selecting()**

Returns true if the user deselected a cell, false if the user deselected a cell.

**wxGridEvent::ShiftDown****bool ShiftDown()**

Returns true if the Shift key was down at the time of the event.

**wxGridRangeSelectEvent**

**Derived from**

*wxNotifyEvent* (p. 1146)  
*wxCommandEvent* (p. 250)  
*wxEvent* (p. 572)  
*wxObject* (p. 1148)

**Event handling**

The event handler for the following functions takes a *wxGridRangeSelectEvent* (p. 785) parameter. The *...\_CMD\_...* variants also take a window identifier.

**EVT\_GRID\_RANGE\_SELECT(func)** The user selected a group of contiguous cells. Processes a *wxEVT\_GRID\_RANGE\_SELECT*.

**EVT\_GRID\_CMD\_RANGE\_SELECT(func)** The user selected a group of contiguous cells; variant taking a window identifier. Processes a *wxEVT\_GRID\_RANGE\_SELECT*.

**Include files**

<wx/grid.h>

**wxGridRangeSelectEvent::wxGridRangeSelectEvent**

**wxGridRangeSelectEvent()**

Default constructor.

**wxGridRangeSelectEvent(int id, wxEventType type, wxObject\* obj, const wxGridCellCoords& topLeft, const wxGridCellCoords& bottomRight, bool sel = true, bool control = false, bool shift = false, bool alt = false, bool meta = false)**

**wxGridRangeSelectEvent::AltDown**

**bool AltDown()**

Returns true if the Alt key was down at the time of the event.

**wxGridRangeSelectEvent::ControlDown**

**bool ControlDown()**

Returns true if the Control key was down at the time of the event.

**wxGridRangeSelectEvent::GetBottomRightCoords**

**wxGridCellCoords GetBottomRightCoords()**

Top left corner of the rectangular area that was (de)selected.

**wxGridRangeSelectEvent::GetBottomRow****int GetBottomRow()**

Bottom row of the rectangular area that was (de)selected.

**wxGridRangeSelectEvent::GetLeftCol****int GetLeftCol()**

Left column of the rectangular area that was (de)selected.

**wxGridRangeSelectEvent::GetRightCol****int GetRightCol()**

Right column of the rectangular area that was (de)selected.

**wxGridRangeSelectEvent::GetTopLeftCoords****wxGridCellCoords GetTopLeftCoords()**

Top left corner of the rectangular area that was (de)selected.

**wxGridRangeSelectEvent::GetTopRow****int GetTopRow()**

Top row of the rectangular area that was (de)selected.

**wxGridRangeSelectEvent::MetaDown****bool MetaDown()**

Returns true if the Meta key was down at the time of the event.

**wxGridRangeSelectEvent::Selecting****bool Selecting()**

Returns true if the area was selected, false otherwise.

**wxGridRangeSelectEvent::ShiftDown****bool ShiftDown()**

Returns true if the Shift key was down at the time of the event.

**wxGridSizeEvent**

This event class contains information about a row/column resize event.

**Derived from**

*wxNotifyEvent* (p. 1146)  
*wxCommandEvent* (p. 250)  
*wxEvent* (p. 572)  
*wxObject* (p. 1148)

**Include files**

<wx/grid.h>

**Event handling**

The event handler for the following functions takes a *wxGridSizeEvent* (p. 787) parameter. The *...\_CMD\_...* variants also take a window identifier.

<b>EVT_GRID_COL_SIZE(func)</b>	The user resized a column by dragging it. Processes a <i>wxEVT_GRID_COL_SIZE</i> .
<b>EVT_GRID_ROW_SIZE(func)</b>	The user resized a row by dragging it. Processes a <i>wxEVT_GRID_ROW_SIZE</i> .
<b>EVT_GRID_CMD_COL_SIZE(func)</b>	The user resized a column by dragging it; variant taking a window identifier. Processes a <i>wxEVT_GRID_COL_SIZE</i> .
<b>EVT_GRID_CMD_ROW_SIZE(func)</b>	The user resized a row by dragging it; variant taking a window identifier. Processes a <i>wxEVT_GRID_ROW_SIZE</i> .

**wxGridSizeEvent::wxGridSizeEvent**

**wxGridSizeEvent()**

Default constructor.

**wxGridSizeEvent(int id, wxEventType type, wxObject\* obj, int rowOrCol = -1, int x = -1, int y = -1, bool control = false, bool shift = false, bool alt = false, bool meta = false)**

**wxGridSizeEvent::AltDown**

**bool AltDown()**

Returns true if the Alt key was down at the time of the event.

**wxGridSizeEvent::ControlDown**

**bool ControlDown()**

Returns true if the Control key was down at the time of the event.

**wxGridSizeEvent::GetPosition****wxPoint GetPosition()**

Position in pixels at which the event occurred.

**wxGridSizeEvent::GetRowOrCol****int GetRowOrCol()**

Row or column at that was resized.

**wxGridSizeEvent::MetaDown****bool MetaDown()**

Returns true if the Meta key was down at the time of the event.

**wxGridSizeEvent::ShiftDown****bool ShiftDown()**

Returns true if the Shift key was down at the time of the event.

**wxGridCellBoolRenderer**

This class may be used to format boolean data in a cell. for string cells.

**Derived from**

*wxGridCellRenderer* (p. 791)

**See also**

*wxGridCellRenderer* (p. 791), *wxGridCellStringRenderer* (p. 792),  
*wxGridCellFloatRenderer* (p. 789), *wxGridCellNumberRenderer* (p. 791)

**Include files**

<wx/grid.h>

**wxGridCellBoolRenderer::wxGridCellBoolRenderer****wxGridCellBoolRenderer()**

Default constructor

**wxGridCellFloatRenderer**

This class may be used to format floating point data in a cell.

**Derived from**

*wxGridCellStringRenderer* (p. 792)

*wxGridCellRenderer* (p. 791)

**See also**

*wxGridCellRenderer* (p. 791), *wxGridCellNumberRenderer* (p. 791),

*wxGridCellStringRenderer* (p. 792), *wxGridCellBoolRenderer* (p. 789)

**Include files**

<wx/grid.h>

**wxGridCellFloatRenderer::wxGridCellFloatRenderer**

**wxGridCellFloatRenderer**(int *width* = -1, int *precision* = -1)

*width*

Minimum number of characters to be shown.

*precision*

Number of digits after the decimal dot.

**wxGridCellFloatRenderer::GetPrecision**

**int GetPrecision() const**

Returns the precision ( see *wxGridCellFloatRenderer* (p. 790) ).

**wxGridCellFloatRenderer::GetWidth**

**int GetWidth() const**

Returns the width ( see *wxGridCellFloatRenderer* (p. 790) ).

**wxGridCellFloatRenderer::SetParameters**

**void SetParameters**(const wxString& *params*)

Parameters string format is "width[,precision]".

**wxGridCellFloatRenderer::SetPrecision**

**void SetPrecision**(int *precision*)

Sets the precision ( see *wxGridCellFloatRenderer* (p. 790) ).

### **wxGridCellFloatRenderer::SetWidth**

**void SetWidth**(int *width*)

Sets the width ( see *wxGridCellFloatRenderer* (p. 790) )

## **wxGridCellNumberRenderer**

This class may be used to format integer data in a cell.

### **Derived from**

*wxGridCellStringRenderer* (p. 792)

*wxGridCellRenderer* (p. 791)

### **See also**

*wxGridCellRenderer* (p. 791), *wxGridCellStringRenderer* (p. 792),  
*wxGridCellFloatRenderer* (p. 789), *wxGridCellBoolRenderer* (p. 789)

### **Include files**

<wx/grid.h>

### **wxGridCellNumberRenderer::wxGridCellNumberRenderer**

**wxGridCellNumberRenderer**()

Default constructor

## **wxGridCellRenderer**

This class is responsible for actually drawing the cell in the grid. You may pass it to the *wxGridCellAttr* (below) to change the format of one given cell or to *wxGrid::SetDefaultRenderer()* to change the view of all cells. This is an abstract class, and you will normally use one of the predefined derived classes or derive your own class from it.

### **Derived from**

*wxGridCellWorker*

### **See also**

*wxGridCellStringRenderer* (p. 792), *wxGridCellNumberRenderer* (p. 791),  
*wxGridCellFloatRenderer* (p. 789), *wxGridCellBoolRenderer* (p. 789)

**Include files**

<wx/grid.h>

**wxGridCellRenderer::Draw**

**void Draw**(wxGrid& *grid*, wxGridCellAttr& *attr*, wxDC& *dc*, const wxRect& *rect*, int *row*, int *col*, bool *isSelected*)

Draw the given cell on the provided DC inside the given rectangle using the style specified by the attribute and the default or selected state corresponding to the *isSelected* value.

This pure virtual function has a default implementation which will prepare the DC using the given attribute: it will draw the rectangle with the background colour from *attr* and set the text colour and font.

**wxGridCellRenderer::GetBestSize**

**wxSize GetBestSize**(wxGrid& *grid*, wxGridCellAttr& *attr*, wxDC& *dc*, int *row*, int *col*)

Get the preferred size of the cell for its contents.

**wxGridCellRenderer::Clone**

**wxGridCellRenderer\* Clone**() const

**wxGridCellStringRenderer**

This class may be used to format string data in a cell; it is the default for string cells.

**Derived from**

*wxGridCellRenderer* (p. 791)

**See also**

*wxGridCellRenderer* (p. 791), *wxGridCellNumberRenderer* (p. 791),  
*wxGridCellFloatRenderer* (p. 789), *wxGridCellBoolRenderer* (p. 789)

**Include files**

<wx/grid.h>

**wxGridCellStringRenderer::wxGridCellStringRenderer**

**wxGridCellStringRenderer**()



Default constructor

## **wxGridTableBase**

Grid table classes.

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/grid.h>

### **wxGridTableBase::wxGridTableBase**

**wxGridTableBase()**

### **wxGridTableBase::~~wxGridTableBase**

**~wxGridTableBase()**

### **wxGridTableBase::GetNumberRows**

**int GetNumberRows()**

You must override these functions in a derived table class.

### **wxGridTableBase::GetNumberCols**

**int GetNumberCols()**

### **wxGridTableBase::IsEmptyCell**

**bool IsEmptyCell(int row, int col)**

### **wxGridTableBase::GetValue**

**wxString GetValue(int row, int col)**

### **wxGridTableBase::SetValue**

**void SetValue(int row, int col, const wxString& value)**

### **wxGridTableBase::GetTypeNames**

**wxString GetTypeNames(int row, int col)**

Data type determination and value access.

**wxGridTableBase::CanGetValueAs**

**bool CanGetValueAs(int row, int col, const wxString& typeName)**

**wxGridTableBase::CanSetValueAs**

**bool CanSetValueAs(int row, int col, const wxString& typeName)**

**wxGridTableBase::GetValueAsLong**

**long GetValueAsLong(int row, int col)**

**wxGridTableBase::GetValueAsDouble**

**double GetValueAsDouble(int row, int col)**

**wxGridTableBase::GetValueAsBool**

**bool GetValueAsBool(int row, int col)**

**wxGridTableBase::SetValueAsLong**

**void SetValueAsLong(int row, int col, long value)**

**wxGridTableBase::SetValueAsDouble**

**void SetValueAsDouble(int row, int col, double value)**

**wxGridTableBase::SetValueAsBool**

**void SetValueAsBool(int row, int col, bool value)**

**wxGridTableBase::GetValueAsCustom**

**void\* GetValueAsCustom(int row, int col, const wxString& typeName)**

For user defined types

**wxGridTableBase::SetValueAsCustom**

**void SetValueAsCustom(int row, int col, const wxString& typeName, void\* value)**

**wxGridTableBase::SetView**

**void SetView(wxGrid\* grid)**

Overriding these is optional

**wxGridTableBase::GetView**

**wxGrid \* GetView() const**

**wxGridTableBase::Clear**

**void Clear()**

**wxGridTableBase::InsertRows**

**bool InsertRows(size\_t pos = 0, size\_t numRows = 1)**

**wxGridTableBase::AppendRows**

**bool AppendRows(size\_t numRows = 1)**

**wxGridTableBase::DeleteRows**

**bool DeleteRows(size\_t pos = 0, size\_t numRows = 1)**

**wxGridTableBase::InsertCols**

**bool InsertCols(size\_t pos = 0, size\_t numCols = 1)**

**wxGridTableBase::AppendCols**

**bool AppendCols(size\_t numCols = 1)**

**wxGridTableBase::DeleteCols**

**bool DeleteCols(size\_t pos = 0, size\_t numCols = 1)**

**wxGridTableBase::GetRowLabelValue**

**wxString GetRowLabelValue(int row)**

**wxGridTableBase::GetColLabelValue**

**wxString GetColLabelValue(int col)**

**wxGridTableBase::SetRowLabelValue**

**void SetRowLabelValue(int WXUNUSED(row), const wxString&)**

**wxGridTableBase::SetColLabelValue**

**void SetColLabelValue(int *WXUNUSED*(col), const wxString&)**

**wxGridTableBase::SetAttrProvider**

**void SetAttrProvider(wxGridCellAttrProvider\* attrProvider)**

Attribute handling give us the attr provider to use - we take ownership of the pointer

**wxGridTableBase::GetAttrProvider**

**wxGridCellAttrProvider\* GetAttrProvider() const**

get the currently used attr provider (may be NULL)

**wxGridTableBase::CanHaveAttributes**

**bool CanHaveAttributes()**

Does this table allow attributes? Default implementation creates a wxGridCellAttrProvider if necessary.

**wxGridTableBase::UpdateAttrRows**

**void UpdateAttrRows(size\_t pos, int numRows)**

change row/col number in attribute if needed

**wxGridTableBase::UpdateAttrCols**

**void UpdateAttrCols(size\_t pos, int numCols)**

**wxGridTableBase::GetAttr**

**wxGridCellAttr\* GetAttr(int row, int col)**

by default forwarded to wxGridCellAttrProvider if any. May be overridden to handle attributes directly in the table.

**wxGridTableBase::SetAttr**

**void SetAttr(wxGridCellAttr\* attr, int row, int col)**

these functions take ownership of the pointer

**wxGridTableBase::SetRowAttr**

**void SetRowAttr(wxGridCellAttr\* attr, int row)**

**wxGridTableBase::SetColAttr**

**void SetColAttr(wxGridCellAttr\* attr, int col)**

## **wxGridSizer**

A grid sizer is a sizer which lays out its children in a two-dimensional table with all table fields having the same size, i.e. the width of each field is the width of the widest child, the height of each field is the height of the tallest child.

### **Derived from**

*wxSizer* (p. 1444)

*wxObject* (p. 1148)

### **Include files**

<wx/sizer.h>

### **See also**

*wxSizer* (p. 1444), *Sizer overview* (p. 2098)

## **wxGridSizer::wxGridSizer**

**wxGridSizer(int rows, int cols, int vgap, int hgap)**

**wxGridSizer(int cols, int vgap = 0, int hgap = 0)**

Constructor for a *wxGridSizer*. *rows* and *cols* determine the number of columns and rows in the sizer - if either of the parameters is zero, it will be calculated to form the total number of children in the sizer, thus making the sizer grow dynamically. *vgap* and *hgap* define extra space between all children.

## **wxGridSizer::GetCols**

**int GetCols()**

Returns the number of columns in the sizer.

## **wxGridSizer::GetHGap**

**int GetHGap()**

Returns the horizontal gap (in pixels) between cells in the sizer.

## **wxGridSizer::GetRows**

**int GetRows()**

Returns the number of rows in the sizer.

**wxGridSizer::GetVGap****int GetVGap()**

Returns the vertical gap (in pixels) between the cells in the sizer.

**wxGridSizer::SetCols****void SetCols(int cols)**

Sets the number of columns in the sizer.

**wxGridSizer::SetHGap****void SetHGap(int gap)**

Sets the horizontal gap (in pixels) between cells in the sizer.

**wxGridSizer::SetRows****void SetRows(int rows)**

Sets the number of rows in the sizer.

**wxGridSizer::SetVGap****void SetVGap(int gap)**

Sets the vertical gap (in pixels) between the cells in the sizer.

**wxHashMap**

This is a simple, type-safe, and reasonably efficient hash map class, whose interface is a subset of the interface of STL containers. In particular, the interface is modeled after `std::map`, and the various, non-standard, `std::hash_map`.

**Example**

```
class MyClass { /* ... */ };

// declare a hash map with string keys and int values
WX_DECLARE_STRING_HASH_MAP( int, MyHash5 );
// same, with int keys and MyClass* values
WX_DECLARE_HASH_MAP( int, MyClass*, wxIntegerHash,
wxIntegerEqual, MyHash1 );
// same, with wxString keys and int values
WX_DECLARE_STRING_HASH_MAP( int, MyHash3 );
// same, with wxString keys and values
WX_DECLARE_STRING_HASH_MAP( wxString, MyHash2 );

MyHash1 h1;
```

```
MyHash2 h2;

// store and retrieve values
hl[1] = new MyClass( 1 );
hl[100000000] = NULL;
hl[50000] = new MyClass( 2 );
h2["Bill"] = "ABC";
wxString tmp = h2["Bill"];
// since element with key "Joe" is not present, this will return
// the default value, which is an empty string in the case of
wxString
MyClass tmp2 = h2["Joe"];

// iterate over all the elements in the class
MyHash2::iterator it;
for( it = h2.begin(); it != h2.end(); ++it )
{
    wxString key = it->first, value = it->second;
    // do something useful with key and value
}
```

### Declaring new hash table types

```
WX_DECLARE_STRING_HASH_MAP( VALUE_T,          // type of the values
                           CLASSNAME ); // name of the class
```

Declares a hash map class named CLASSNAME, with wxString keys and VALUE\_T values.

```
WX_DECLARE_VOIDPTR_HASH_MAP( VALUE_T,         // type of the values
                             CLASSNAME ); // name of the class
```

Declares a hash map class named CLASSNAME, with void\* keys and VALUE\_T values.

```
WX_DECLARE_HASH_MAP( KEY_T,          // type of the keys
                    VALUE_T,        // type of the values
                    HASH_T,         // hasher
                    KEY_EQ_T,       // key equality predicate
                    CLASSNAME ); // name of the class
```

The HASH\_T and KEY\_EQ\_T are the types used for the hashing function and key comparison. wxWidgets provides three predefined hashing functions: wxIntegerHash for integer types ( int, long, short, and their unsigned counterparts ), wxStringHash for strings ( wxString, wxChar\*, char\* ), and wxPointerHash for any kind of pointer. Similarly three equality predicates: wxIntegerEqual, wxStringEqual, wxPointerEqual are provided.

Using this you could declare a hash map mapping int values to wxString like this:

```
WX_DECLARE_HASH_MAP( int,
                    wxString,
                    wxIntegerHash,
                    wxIntegerEqual,
                    MyHash );
```

```
// using an user-defined class for keys
class MyKey { /* ... */ };

// hashing function
class MyKeyHash
{
public:
    MyKeyHash() { }

    unsigned long operator()( const MyKey& k ) const
    { /* compute the hash */ }

    MyKeyHash& operator=(const MyKeyHash&) { return *this; }
};

// comparison operator
class MyKeyEqual
{
public:
    MyKeyEqual() { }
    bool operator()( const MyKey& a, const MyKey& b ) const
    { /* compare for equality */ }

    MyKeyEqual& operator=(const MyKeyEqual&) { return *this; }
};

WX_DECLARE_HASH_MAP( MyKey,          // type of the keys
                     SOME_TYPE,    // any type you like
                     MyKeyHash,    // hasher
                     MyKeyEqual,    // key equality predicate
                     CLASSNAME); // name of the class
```

In the documentation below you should replace `wxHashMap` with the name you used in the class declaration.

<code>wxHashMap::key_type</code>	Type of the hash keys
<code>wxHashMap::mapped_type</code>	Type of the values stored in the hash map
<code>wxHashMap::value_type</code>	Equivalent to <code>struct { key_type first; mapped_type second };</code>
<code>wxHashMap::iterator</code>	Used to enumerate all the elements in a hash map; it is similar to a <code>value_type*</code>
<code>wxHashMap::const_iterator</code>	Used to enumerate all the elements in a constant hash map; it is similar to a <code>const value_type*</code>
<code>wxHashMap::size_type</code>	Used for sizes
<code>wxHashMap::Insert_Result</code>	The return value for <code>insert()</code> (p. 802)

### Iterators



An iterator is similar to a pointer, and so you can use the usual pointer operations: `++it` ( and `it++` ) to move to the next element, `*it` to access the element pointed to, `it->first( it->second )` to access the key ( value ) of the element pointed to. Hash maps provide forward only iterators, this means that you can't use `--it`, `it + 3`, `it1 - it2`.

**Include files**

`<wx/hashmap.h>`

**wxHashMap::wxHashMap**

**wxHashMap(size\_type size = 10)**

The size parameter is just a hint, the table will resize automatically to preserve performance.

**wxHashMap(const wxHashMap& map)**

Copy constructor.

**wxHashMap::begin**

**const\_iterator begin() const**

**iterator begin()**

Returns an iterator pointing at the first element of the hash map. Please remember that hash maps do not guarantee ordering.

**wxHashMap::clear**

**void clear()**

Removes all elements from the hash map.

**wxHashMap::count**

**size\_type count(const key\_type& key) const**

Counts the number of elements with the given key present in the map. This function returns only 0 or 1.

**wxHashMap::empty**

**bool empty() const**

Returns true if the hash map does not contain any elements, false otherwise.

**wxHashMap::end****const\_iterator end() const****iterator end()**

Returns an iterator pointing at the one-after-the-last element of the hash map. Please remember that hash maps do not guarantee ordering.

**wxHashMap::erase****size\_type erase(const key\_type& key)**

Erases the element with the given key, and returns the number of elements erased (either 0 or 1).

**void erase(iterator it)****void erase(const\_iterator it)**

Erases the element pointed to by the iterator. After the deletion the iterator is no longer valid and must not be used.

**wxHashMap::find****iterator find(const key\_type& key)****const\_iterator find(const key\_type& key) const**

If an element with the given key is present, the functions returns an iterator pointing at that element, otherwise an invalid iterator is returned (i.e. `hashmap.find( non_existent_key ) == hashmap.end()`).

**wxHashMap::insert****Insert\_Result insert(const value\_type& v)**

Inserts the given value in the hash map. The return value is equivalent to a `std::pair<wxHashMap::iterator, bool>`; the iterator points to the inserted element, the boolean value is `true` if `v` was actually inserted.

**wxHashMap::operator[]****mapped\_type& operator[](const key\_type& key)**

Use the key as an array subscript. The only difference is that if the given key is not present in the hash map, an element with the default `value_type()` is inserted in the table.

**wxHashMap::size****size\_type size() const**

Returns the number of elements in the map.

## wxHashSet

This is a simple, type-safe, and reasonably efficient hash set class, whose interface is a subset of the interface of STL containers. In particular, the interface is modeled after `std::set`, and the various, non-standard, `std::hash_map`.

### Example

```
class MyClass { /* ... */ };

// same, with MyClass* keys (only uses pointer equality!)
WX_DECLARE_HASH_SET( MyClass*, wxPointerHash, wxPointerEqual,
MySet1 );
// same, with int keys
WX_DECLARE_HASH_SET( int, wxIntegerHash, wxIntegerEqual,
MySet2 );
// declare a hash set with string keys
WX_DECLARE_HASH_SET( wxString, wxStringHash, wxStringEqual,
MySet3 );

MySet1 h1;
MySet2 h1;
MySet3 h3;

// store and retrieve values
h1.insert( new MyClass( 1 ) );

h3.insert( "foo" );
h3.insert( "bar" );
h3.insert( "baz" );

int size = h3.size(); // now is three
bool has_foo = h3.find( "foo" ) != h3.end();

h3.insert( "bar" ); // still has size three

// iterate over all the elements in the class
MySet3::iterator it;
for( it = h3.begin(); it != h3.end(); ++it )
{
    wxString key = *it;
    // do something useful with key
}
```

### Declaring new hash set types

```
WX_DECLARE_HASH_SET( KEY_T,          // type of the keys
                    HASH_T,          // hasher
                    KEY_EQ_T,        // key equality predicate
                    CLASSNAME); // name of the class
```

The `HASH_T` and `KEY_EQ_T` are the types used for the hashing function and key

comparison. `wxWidgets` provides three predefined hashing functions: `wxIntegerHash` for integer types ( `int`, `long`, `short`, and their unsigned counterparts ), `wxStringHash` for strings ( `wxString`, `wxChar*`, `char*` ), and `wxPointerHash` for any kind of pointer. Similarly three equality predicates: `wxIntegerEqual`, `wxStringEqual`, `wxPointerEqual` are provided.

Using this you could declare a hash set using `int` values like this:

```
WX_DECLARE_HASH_SET( int,
                    wxIntegerHash,
                    wxIntegerEqual,
                    MySet );

// using an user-defined class for keys
class MyKey { /* ... */ };

// hashing function
class MyKeyHash
{
public:
    MyKeyHash() { }

    unsigned long operator()( const MyKey& k ) const
    { /* compute the hash */ }

    MyKeyHash& operator=(const MyKeyHash&) { return *this; }
};

// comparison operator
class MyKeyEqual
{
public:
    MyKeyEqual() { }
    bool operator()( const MyKey& a, const MyKey& b ) const
    { /* compare for equality */ }

    MyKeyEqual& operator=(const MyKeyEqual&) { return *this; }
};

WX_DECLARE_HASH_SET( MyKey,          // type of the keys
                    MyKeyHash,      // hasher
                    MyKeyEqual,     // key equality predicate
                    CLASSNAME);    // name of the class
```

In the documentation below you should replace `wxHashSet` with the name you used in the class declaration.

<code>wxHashSet::key_type</code>	Type of the hash keys
<code>wxHashSet::mapped_type</code>	Type of hash keys
<code>wxHashSet::value_type</code>	Type of hash keys

<code>wxHashSet::iterator</code>	Used to enumerate all the elements in a hash set; it is similar to a <code>value_type*</code>
<code>wxHashSet::const_iterator</code>	Used to enumerate all the elements in a constant hash set; it is similar to a <code>const value_type*</code>
<code>wxHashSet::size_type</code>	Used for sizes
<code>wxHashSet::Insert_Result</code>	The return value for <code>insert()</code> (p. 806)

### Iterators

An iterator is similar to a pointer, and so you can use the usual pointer operations: `++it` ( and `it++` ) to move to the next element, `*it` to access the element pointed to, `*it` to access the value of the element pointed to. Hash sets provide forward only iterators, this means that you can't use `--it`, `it + 3`, `it1 - it2`.

### Include files

`<wx/hashset.h>`

### `wxHashSet::wxHashSet`

`wxHashSet(size_type size = 10)`

The size parameter is just a hint, the table will resize automatically to preserve performance.

`wxHashSet(const wxHashSet& set)`

Copy constructor.

### `wxHashSet::begin`

`const_iterator begin() const`

`iterator begin()`

Returns an iterator pointing at the first element of the hash set. Please remember that hash sets do not guarantee ordering.

### `wxHashSet::clear`

`void clear()`

Removes all elements from the hash set.

### `wxHashSet::count`

**size\_type count(const key\_type& key) const**

Counts the number of elements with the given key present in the set. This function returns only 0 or 1.

**wxHashSet::empty****bool empty() const**

Returns true if the hash set does not contain any elements, false otherwise.

**wxHashSet::end****const\_iterator end() const****iterator end()**

Returns an iterator pointing at the one-after-the-last element of the hash set. Please remember that hash sets do not guarantee ordering.

**wxHashSet::erase****size\_type erase(const key\_type& key)**

Erases the element with the given key, and returns the number of elements erased (either 0 or 1).

**void erase(iterator it)****void erase(const\_iterator it)**

Erases the element pointed to by the iterator. After the deletion the iterator is no longer valid and must not be used.

**wxHashSet::find****iterator find(const key\_type& key)****const\_iterator find(const key\_type& key) const**

If an element with the given key is present, the functions returns an iterator pointing at that element, otherwise an invalid iterator is returned (i.e. `hashset.find( non_existent_key ) == hashset.end()`).

**wxHashSet::insert****Insert\_Result insert(const value\_type& v)**

Inserts the given value in the hash set. The return value is equivalent to a `std::pair<wxHashMap::iterator, bool>`; the iterator points to the inserted element, the boolean value is `true` if `v` was actually inserted.

**wxHashSet::size****size\_type size() const**

Returns the number of elements in the set.

**wxHashTable**

**Please note** that this class is retained for backward compatibility reasons; you should use *wxHashMap* (p. 798).

This class provides hash table functionality for *wxWidgets*, and for an application if it wishes. Data can be hashed on an integer or string key.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/hash.h>

**Example**

Below is an example of using a hash table.

```
wxHashTable table(wxKEY_STRING);

wxPoint *point = new wxPoint(100, 200);
table.Put("point 1", point);

....

wxPoint *found_point = (wxPoint *)table.Get("point 1");
```

A hash table is implemented as an array of pointers to lists. When no data has been stored, the hash table takes only a little more space than this array (default size is 1000). When a data item is added, an integer is constructed from the integer or string key that is within the bounds of the array. If the array element is NULL, a new (keyed) list is created for the element. Then the data object is appended to the list, storing the key in case other data objects need to be stored in the list also (when a 'collision' occurs).

Retrieval involves recalculating the array index from the key, and searching along the keyed list for the data object whose stored key matches the passed key. Obviously this is quicker when there are fewer collisions, so hashing will become inefficient if the number of items to be stored greatly exceeds the size of the hash table.

**See also**

*wxList* (p. 966)

**wxHashTable::wxHashTable****wxHashTable(unsigned int *key\_type*, int *size* = 1000)**

Constructor. *key\_type* is one of wxKEY\_INTEGER, or wxKEY\_STRING, and indicates what sort of keying is required. *size* is optional.

**wxHashTable::~~wxHashTable****~wxHashTable()**

Destroys the hash table.

**wxHashTable::BeginFind****void BeginFind()**

The counterpart of *Next*. If the application wishes to iterate through all the data in the hash table, it can call *BeginFind* and then loop on *Next*.

**wxHashTable::Clear****void Clear()**

Clears the hash table of all nodes (but as usual, doesn't delete user data).

**wxHashTable::Delete****wxObject \* Delete(long *key*)****wxObject \* Delete(const wxString& *key*)**

Deletes entry in hash table and returns the user's data (if found).

**wxHashTable::DeleteContents****void DeleteContents(bool *flag*)**

If set to true data stored in hash table will be deleted when hash table object is destroyed.

**wxHashTable::Get****wxObject \* Get(long *key*)****wxObject \* Get(const 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(const wxString& string)**

Makes an integer key out of a string. An application may wish to make a key explicitly (for instance when combining two data values to form a key).

**wxHashTable::Next****wxHashTable::Node \* Next()**

If the application wishes to iterate through all the data in the hash table, it can call *BeginFind* and then loop on *Next*. This function returns a **wxHashTable::Node** pointer (or NULL if there are no more nodes). The return value is functionally equivalent to **wxNode** but might not be implemented as a **wxNode**. The user will probably only wish to use the **GetData** method to retrieve the data; the node may also be deleted.

**wxHashTable::Put****void Put(long key, wxObject \*object)****void Put(const 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.

**wxHashTable::GetCount****size\_t GetCount() const**

Returns the number of elements in the hash table.

**wxHelpController**

This is a family of classes by which applications may invoke a help viewer to provide on-line help.

A help controller allows an application to display help, at the contents or at a particular topic, and shut the help program down on termination. This avoids proliferation of many instances of the help viewer whenever the user requests a different topic via the application's menus or buttons.

Typically, an application will create a help controller instance when it starts, and immediately call **Initialize** to associate a filename with it. The help viewer will only get run, however, just before the first call to display something.

Most help controller classes actually derive from **wxHelpControllerBase** and have names of the form **wxXXXHelpController** or **wxHelpControllerXXX**. An appropriate class is aliased to the name **wxHelpController** for each platform, as follows:

- On desktop Windows, **wxCHMHelpController** is used (MS HTML Help).

- On Windows CE, `wxWinceHelpController` is used.
- On all other platforms, `wxHtmlHelpController` is used if `wxHTML` is compiled into `wxWidgets`; otherwise `wxExtHelpController` is used (for invoking an external browser).

The remaining help controller classes need to be named explicitly by an application that wishes to make use of them.

There are currently the following help controller classes defined:

- `wxWinHelpController`, for controlling Windows Help.
- `wxCHMHelpController`, for controlling MS HTML Help. To use this, you need to set `wxUSE_MS_HTML_HELP` to 1 in `setup.h` and have `htmlhelp.h` header from Microsoft's HTML Help kit (you don't need VC++ specific `htmlhelp.lib` because `wxWidgets` loads necessary DLL at runtime and so it works with all compilers).
- `wxBestHelpController`, for controlling MS HTML Help or, if Microsoft's runtime is not available, *wxHtmlHelpController* (p. 838). You need to provide **both** CHM and HTB versions of the help file. For 32bit Windows only.
- `wxExtHelpController`, for controlling external browsers under Unix. The default browser is Netscape Navigator. The 'help' sample shows its use.
- `wxWinceHelpController`, for controlling a simple `.htm` help controller for Windows CE applications.
- *wxHtmlHelpController* (p. 838), a sophisticated help controller using `wxHTML` (p. 2179), in a similar style to the Microsoft HTML Help viewer and using some of the same files. Although it has an API compatible with other help controllers, it has more advanced features, so it is recommended that you use the specific API for this class instead. Note that if you use `.zip` or `.htb` formats for your books, you must add this line to your application initialization:  
`wxFileSystem::AddHandler(new wxArchiveFSHandler);` or nothing will be shown in your help window.

### Derived from

`wxHelpControllerBase`  
*wxObject* (p. 1148)

### Include files

`<wx/help.h>` (`wxWidgets` chooses the appropriate help controller class)  
`<wx/helpbase.h>` (`wxHelpControllerBase` class)  
`<wx/helpwin.h>` (Windows Help controller)  
`<wx/msw/helpchm.h>` (MS HTML Help controller)  
`<wx/generic/helpext.h>` (external HTML browser controller)  
`<wx/html/helpctrl.h>` (`wxHTML` based help controller: `wxHtmlHelpController`)

### See also

*wxHtmlHelpController* (p. 838), *wxHTML* (p. 2179)

### **wxHelpController::wxHelpController**

**wxHelpController(wxWindow\* parentWindow = NULL)**

Constructs a help instance object, but does not invoke the help viewer.

If you provide a window, it will be used by some help controller classes, such as *wxCHMHelpController*, *wxWinHelpController* and *wxHtmlHelpController*, as the parent for the help window instead of the value of *wxApp::GetTopWindow* (p. 48). You can also change the parent window later with *wxHelpController::SetParentWindow* (p. 814).

### **wxHelpController::~~wxHelpController**

**~wxHelpController()**

Destroys the help instance, closing down the viewer if it is running.

### **wxHelpController::Initialize**

**virtual bool Initialize(const wxString& file)**

**virtual bool Initialize(const wxString& file, int server)**

Initializes the help instance with a help filename, and optionally a server socket number if using *wxHelp* (now obsolete). Does not invoke the help viewer. This must be called directly after the help instance object is created and before any attempts to communicate with the viewer.

You may omit the file extension and a suitable one will be chosen. For *wxHtmlHelpController*, the extensions *zip*, *htb* and *hhp* will be appended while searching for a suitable file. For *WinHelp*, the *hlp* extension is appended.

### **wxHelpController::DisplayBlock**

**virtual bool DisplayBlock(long blockNo)**

If the help viewer is not running, runs it and displays the file at the given block number.

*WinHelp*: Refers to the context number.

*MS HTML Help*: Refers to the context number.

*External HTML help*: the same as for *wxHelpController::DisplaySection* (p. 812).

*wxHtmlHelpController*: *sectionNo* is an identifier as specified in the *.hhc* file. See *Help files format* (p. 2181).

This function is for backward compatibility only, and applications should use

*wxHelpController* (p. 812) instead.

### **wxHelpController::DisplayContents**

**virtual bool DisplayContents()**

If the help viewer is not running, runs it and displays the contents.

### **wxHelpController::DisplayContextPopup**

**virtual bool DisplayContextPopup(int contextId)**

Displays the section as a popup window using a context id.

Returns false if unsuccessful or not implemented.

### **wxHelpController::DisplaySection**

**virtual bool DisplaySection(const wxString& section)**

If the help viewer is not running, runs it and displays the given section.

The interpretation of *section* differs between help viewers. For most viewers, this call is equivalent to *KeywordSearch*. For MS HTML Help, wxHTML help and external HTML help, if *section* has a .htm or .html extension, that HTML file will be displayed; otherwise a keyword search is done.

**virtual bool DisplaySection(int sectionNo)**

If the help viewer is not running, runs it and displays the given section.

*WinHelp*, *MS HTML Help* *sectionNo* is a context id.

*External HTML help*: *wxExtHelpController* implements *sectionNo* as an id in a map file, which is of the form:

```
0  wx.html           ; Index
1  wx34.html#classref ; Class reference
2  wx204.html        ; Function reference
```

*wxHtmlHelpController*: *sectionNo* is an identifier as specified in the .hhc file. See *Help files format* (p. 2181).

See also the help sample for notes on how to specify section numbers for various help file formats.

### **wxHelpController::DisplayTextPopup**

**virtual bool DisplayTextPopup(const wxString& text, const wxPoint& pos)**

Displays the text in a popup window, if possible.

Returns false if unsuccessful or not implemented.

### **wxHelpController::GetFrameParameters**

**virtual wxFrame \* GetFrameParameters(const wxSize \* size = NULL, const wxPoint \* pos = NULL, bool \*newFrameEachTime = NULL)**

wxHtmlHelpController returns the frame, size and position.

For all other help controllers, this function does nothing and just returns NULL.

#### **Parameters**

*viewer*

This defaults to "netscape" for wxExtHelpController.

*flags*

This defaults to wxHELP\_NETSCAPE for wxExtHelpController, indicating that the viewer is a variant of Netscape Navigator.

### **wxHelpController::GetParentWindow**

**virtual wxWindow\* GetParentWindow() const**

Returns the window to be used as the parent for the help window. This window is used by wxCHMHelpController, wxWinHelpController and wxHtmlHelpController.

### **wxHelpController::KeywordSearch**

**virtual bool KeywordSearch(const wxString& keyWord, wxHelpSearchMode mode = wxHELP\_SEARCH\_ALL)**

If the help viewer is not running, runs it, and searches for sections matching the given keyword. If one match is found, the file is displayed at this section. The optional parameter allows the search the index (wxHELP\_SEARCH\_INDEX) but this currently only supported by the wxHtmlHelpController.

*WinHelp, MS HTML Help:* If more than one match is found, the first topic is displayed.

*External HTML help, simple wxHTML help:* If more than one match is found, a choice of topics is displayed.

*wxHtmlHelpController:* see *wxHtmlHelpController::KeywordSearch* (p. 842).

### **wxHelpController::LoadFile**

**virtual bool LoadFile(const wxString& file = "")**

If the help viewer is not running, runs it and loads the given file. If the filename is not supplied or is empty, the file specified in **Initialize** is used. If the viewer is already

displaying the specified file, it will not be reloaded. This member function may be used before each display call in case the user has opened another file.

`wxHtmlHelpController` ignores this call.

### **`wxHelpController::OnQuit`**

#### **`virtual bool OnQuit()`**

Overridable member called when this application's viewer is quit by the user.

This does not work for all help controllers.

### **`wxHelpController::SetFrameParameters`**

#### **`virtual void SetFrameParameters(const wxString & title, const wxSize & size, const wxPoint & pos = wxDefaultPosition, bool newFrameEachTime = false)`**

For `wxHtmlHelpController`, the title is set (again with %s indicating the page title) and also the size and position of the frame if the frame is already open. *newFrameEachTime* is ignored.

For all other help controllers this function has no effect.

### **`wxHelpController::SetParentWindow`**

#### **`virtual void SetParentWindow(wxWindow* parentWindow)`**

Sets the window to be used as the parent for the help window. This is used by `wxCHMHelpController`, `wxWinHelpController` and `wxHtmlHelpController`.

### **`wxHelpController::SetViewer`**

#### **`virtual void SetViewer(const wxString& viewer, long flags)`**

Sets detailed viewer information. So far this is only relevant to `wxExtHelpController`.

Some examples of usage:

```
m_help.SetViewer("kdehelp");  
m_help.SetViewer("gnome-help-browser");  
m_help.SetViewer("netscape", wxHELP_NETSCAPE);
```

### **`wxHelpController::Quit`**

#### **`virtual bool Quit()`**

If the viewer is running, quits it by disconnecting.

For Windows Help, the viewer will only close if no other application is using it.

## wxHelpControllerHelpProvider

`wxHelpControllerHelpProvider` is an implementation of `wxHelpProvider` which supports both context identifiers and plain text help strings. If the help text is an integer, it is passed to `wxHelpController::DisplayContextPopup`. Otherwise, it shows the string in a tooltip as per `wxSimpleHelpProvider`. If you use this with a `wxCHMHelpController` instance on windows, it will use the native style of tip window instead of `wxTipWindow` (p. 1691).

You can use the convenience function **`wxContextId`** to convert an integer context id to a string for passing to `wxWindow::SetHelpText` (p. 1841).

### Derived from

`wxSimpleHelpProvider` (p. 1432)  
`wxHelpProvider` (p. 817)

### Include files

<wx/cshelp.h>

### See also

`wxHelpProvider` (p. 817), `wxSimpleHelpProvider` (p. 1432), `wxContextHelp` (p. 282), `wxWindow::SetHelpText` (p. 1841), `wxWindow::GetHelpTextAtPoint` (p. 1814)

## `wxHelpControllerHelpProvider::wxHelpControllerHelpProvider`

**`wxHelpControllerHelpProvider(wxHelpControllerBase* hc = NULL)`**

Note that the instance doesn't own the help controller. The help controller should be deleted separately.

## `wxHelpControllerHelpProvider::SetHelpController`

**`void SetHelpController(wxHelpControllerBase* hc)`**

Sets the help controller associated with this help provider.

## `wxHelpControllerHelpProvider::GetHelpController`

**`wxHelpControllerBase* GetHelpController() const`**

Returns the help controller associated with this help provider.

## wxHelpEvent

A help event is sent when the user has requested context-sensitive help. This can either be caused by the application requesting context-sensitive help mode via `wxContextHelp` (p. 282), or (on MS Windows) by the system generating a `WM_HELP` message when the

user pressed F1 or clicked on the query button in a dialog caption.

A help event is sent to the window that the user clicked on, and is propagated up the window hierarchy until the event is processed or there are no more event handlers. The application should call `wxEvt::GetId` to check the identity of the clicked-on window, and then either show some suitable help or call `wxEvt::Skip` if the identifier is unrecognised. Calling `Skip` is important because it allows `wxWidgets` to generate further events for ancestors of the clicked-on window. Otherwise it would be impossible to show help for container windows, since processing would stop after the first window found.

### Derived from

*wxCommandEvent* (p. 250)

*wxEvt* (p. 572)

*wxObject* (p. 1148)

### Include files

<wx/event.h>

### Event table macros

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

<b>EVT_HELP(id, func)</b>	Process a <code>wxEVT_HELP</code> event.
<b>EVT_HELP_RANGE(id1, id2, func)</b>	Process a <code>wxEVT_HELP</code> event for a range of ids.

### See also

*wxContextHelp* (p. 282), *wxDialog* (p. 496), *Event handling overview* (p. 2077)

## wxHelpEvent::wxHelpEvent

**wxHelpEvent(WXTYPE eventType = 0, wxWindowID id = 0, const wxPoint& point)**

Constructor.

## wxHelpEvent::GetOrigin

**wxHelpEvent::Origin GetOrigin() const**

Returns the origin of the help event which is one of the following values:

<b>Origin_Unknown</b>	Unrecognized event source.
<b>Origin_Keyboard</b>	Event generated by F1 key press.
<b>Origin_HelpButton</b>	Event generated by <i>wxContextHelp</i> (p. 282) or using the "?" title bar button under MS Windows.



The application may handle events generated using the keyboard or mouse differently, e.g. by using *wxGetMousePosition()* (p. 1957) for the mouse events.

**See also**

*wxHelpEvent::SetOrigin* (p. 817)

**wxHelpEvent::GetPosition**

**const wxPoint& GetPosition() const**

Returns the left-click position of the mouse, in screen coordinates. This allows the application to position the help appropriately.

**wxHelpEvent::SetOrigin**

**void SetOrigin(wxHelpEvent::Origin origin)**

Set the help event origin, only used internally by wxWidgets normally.

**See also**

*wxHelpEvent::GetOrigin* (p. 816)

**wxHelpEvent::SetPosition**

**void SetPosition(const wxPoint& pt)**

Sets the left-click position of the mouse, in screen coordinates.

**wxHelpProvider**

wxHelpProvider is an abstract class used by a program implementing context-sensitive help to show the help text for the given window.

The current help provider must be explicitly set by the application using *wxHelpProvider::Set()*.

**Derived from**

No base class

**Include files**

<wx/cshelp.h>

**See also**

*wxContextHelp* (p. 282), *wxContextHelpButton* (p. 284), *wxSimpleHelpProvider* (p. 1432), *wxHelpControllerHelpProvider* (p. 815), *wxWindow::SetHelpText* (p. 1841), *wxWindow::GetHelpTextAtPoint* (p. 1814)

**wxHelpProvider::~~wxHelpProvider****~wxHelpProvider()**

Virtual destructor for any base class.

**wxHelpProvider::AddHelp****virtual void AddHelp(wxWindowBase\* window, const wxString& text)**

Associates the text with the given window or id. Although all help providers have these functions to allow making *wxWindow::SetHelpText* (p. 1841) work, not all of them implement the functions.

**virtual void AddHelp(wxWindowID id, const wxString& text)**

This version associates the given text with all windows with this id. May be used to set the same help string for all Cancel buttons in the application, for example.

**wxHelpProvider::Get****static wxHelpProvider\* Get()**

Returns pointer to help provider instance.

Unlike some other classes, the help provider is not created on demand. This must be explicitly done by the application using *Set* (p. 818).

**wxHelpProvider::GetHelp****virtual wxString GetHelp(const wxWindowBase\* window)**

Gets the help string for this window. Its interpretation is dependent on the help provider except that empty string always means that no help is associated with the window.

**wxHelpProvider::RemoveHelp****virtual void RemoveHelp(wxWindowBase\* window)**

Removes the association between the window pointer and the help text, if it was previously set using *AddHelp* (p. 818). This is called by the *wxWindow* destructor. Without this, the table of help strings will fill up and when window pointers are reused, the wrong help string will be found.

Note that this method may be called even for windows that don't have any associated help text. If that happens, its implementation should simply do nothing.

**wxHelpProvider::Set**

**static wxHelpProvider\* Set(wxHelpProvider\* helpProvider)**

Get/set the current, application-wide help provider. Returns the previous one.

### **wxHelpProvider::ShowHelpAtPoint**

**virtual bool ShowHelpAtPoint(wxWindowBase\* window, const wxPoint&point, wxHelpEvent::Origin origin)**

This function may be overridden to show help for the window when it should depend on the position inside the window, By default this method forwards to *ShowHelp* (p. 819), so it is enough to only implement the latter if the help doesn't depend on the position.

Returns `true` if help was shown, or `false` if no help was available for this window.

#### **Parameters**

*window*

Window to show help text for.

*point*

Coordinates of the mouse at the moment of help event emission.

*origin*

Help event origin, see *wxHelpEvent::GetOrigin* (p. 816).

This function is new since wxWidgets version 2.7.0

### **wxHelpProvider::ShowHelp**

**virtual bool ShowHelp(wxWindowBase\* window)**

Shows help for the given window. Override this function if the help doesn't depend on the exact position inside the window, otherwise you need to override *ShowHelpAtPoint* (p. 819).

Returns `true` if help was shown, or `false` if no help was available for this window.

## **wxHtmlCell**

Internal data structure. It represents fragments of parsed HTML page, the so-called **cell** - a word, picture, table, horizontal line and so on. It is used by *wxHtmlWindow* (p. 872) and *wxHtmlWinParser* (p. 883) to represent HTML page in memory.

You can divide cells into two groups : *visible* cells with non-zero width and height and *helper* cells (usually with zero width and height) that perform special actions such as color or font change.

#### **Derived from**

*wxObject* (p. 1148)

#### **Include files**

<wx/html/htmlcell.h>

#### **See Also**

*Cells Overview* (p. 2182), *wxHtmlContainerCell* (p. 826)

### **wxHtmlCell::wxHtmlCell**

#### **wxHtmlCell()**

Constructor.

### **wxHtmlCell::AdjustPagebreak**

#### **virtual bool AdjustPagebreak(int \* pagebreak)**

This method is used to adjust pagebreak position. The parameter is variable that contains y-coordinate of page break (= horizontal line that should not be crossed by words, images etc.). If this cell cannot be divided into two pieces (each one on another page) then it moves the pagebreak few pixels up.

Returns true if pagebreak was modified, false otherwise

```
Usage: while (container->AdjustPagebreak(&p)) {}
```

### **wxHtmlCell::Draw**

#### **virtual void Draw(wxDC& dc, int x, int y, int view\_y1, int view\_y2)**

Renders the cell.

#### **Parameters**

*dc*

Device context to which the cell is to be drawn

*x,y*

Coordinates of parent's upper left corner (origin). You must add this to *m\_PosX*, *m\_PosY* when passing coordinates to *dc*'s methods Example : *dc -> DrawText("hello", x + m\_PosX, y + m\_PosY)*

*view\_y1*

y-coord of the first line visible in window. This is used to optimize rendering speed

*view\_y2*

y-coord of the last line visible in window. This is used to optimize rendering speed

### **wxHtmlCell::DrawInvisible**

**virtual void DrawInvisible(wxDC& *dc*, int *x*, int *y*)**

This method is called instead of *Draw* (p. 820) when the cell is certainly out of the screen (and thus invisible). This is not nonsense - some tags (like *wxHtmlColourCell* (p. 825) or font setter) must be drawn even if they are invisible!

#### **Parameters**

*dc*

Device context to which the cell is to be drawn

*x,y*

Coordinates of parent's upper left corner. You must add this to *m\_PosX*, *m\_PosY* when passing coordinates to *dc*'s methods Example : *dc -> DrawText( "hello" , x + m\_PosX, y + m\_PosY)*

### **wxHtmlCell::Find**

**virtual const wxHtmlCell\* Find(int *condition*, const void\* *param*)**

Returns pointer to itself if this cell matches condition (or if any of the cells following in the list matches), NULL otherwise. (In other words if you call top-level container's *Find* it will return pointer to the first cell that matches the condition)

It is recommended way how to obtain pointer to particular cell or to cell of some type (e.g. *wxHtmlAnchorCell* reacts on *wxHTML\_COND\_ISANCHOR* condition)

#### **Parameters**

*condition*

Unique integer identifier of condition

*param*

Optional parameters

#### **Defined conditions**

**wxHTML\_COND\_ISANCHOR**

Finds particular anchor. *param* is pointer to *wxString* with name of the anchor.

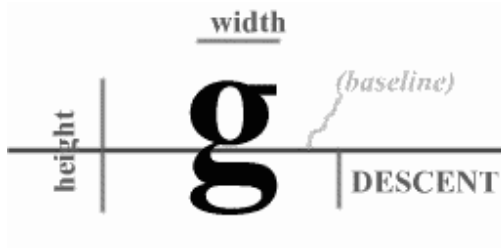
**wxHTML\_COND\_USER**

User-defined conditions start from this number.

### **wxHtmlCell::GetDescent**

**int GetDescent() const**

Returns descent value of the cell (m\_Descent member). See explanation:

**wxHtmlCell::GetFirstChild****wxHtmlCell\* GetFirstChild()**

Returns pointer to the first cell in the list. You can then use child's *GetNext* (p. 823) method to obtain pointer to the next cell in list.

**Note:** This shouldn't be used by the end user. If you need some way of finding particular cell in the list, try *Find* (p. 821) method instead.

**wxHtmlCell::GetHeight****int GetHeight() const**

Returns height of the cell (m\_Height member).

**wxHtmlCell::GetId****virtual wxString GetId() const**

Returns unique cell identifier if there is any, empty string otherwise.

**wxHtmlCell::GetLink****virtual wxHtmlLinkInfo\* GetLink(int x = 0, int y = 0) const**

Returns hypertext link if associated with this cell or NULL otherwise. See *wxHtmlLinkInfo* (p. 851). (Note: this makes sense only for visible tags).

**Parameters**

*x,y*

Coordinates of position where the user pressed mouse button. These coordinates are used e.g. by COLORMAP. Values are relative to the upper left corner of THIS cell (i.e. from 0 to m\_Width or m\_Height)

**wxHtmlCell::GetMouseCursor**

**virtual wxCursor GetMouseCursor(wxHtmlWindowInterface\* window)**

Returns cursor to show when mouse pointer is over the cell.

**Parameters**

*window*

interface to the parent HTML window

This function is new since wxWidgets version 2.7.0 (before `GetCursor` method served a similar purpose, which is now deprecated).

**wxHtmlCell::GetNext**

**wxHtmlCell\* GetNext() const**

Returns pointer to the next cell in list (see `htmlcell.h` if you're interested in details).

**wxHtmlCell::GetParent**

**wxHtmlContainerCell\* GetParent() const**

Returns pointer to parent container.

**wxHtmlCell::GetPosX**

**int GetPosX() const**

Returns X position within parent (the value is relative to parent's upper left corner). The returned value is meaningful only if parent's *Layout* (p. 823) was called before!

**wxHtmlCell::GetPosY**

**int GetPosY() const**

Returns Y position within parent (the value is relative to parent's upper left corner). The returned value is meaningful only if parent's *Layout* (p. 823) was called before!

**wxHtmlCell::GetWidth**

**int GetWidth() const**

Returns width of the cell (`m_Width` member).

**wxHtmlCell::Layout**

**virtual void Layout(int w)**

This method performs two actions:

1. adjusts the cell's width according to the fact that maximal possible width is *w*. (this has sense when working with horizontal lines, tables etc.)
2. prepares layout (=fill-in *m\_PosX*, *m\_PosY* (and sometimes *m\_Height*) members) based on actual width *w*

It must be called before displaying cells structure because *m\_PosX* and *m\_PosY* are undefined (or invalid) before calling *Layout*.

### **wxHtmlCell::ProcessMouseClicked**

**virtual bool ProcessMouseClicked(wxHtmlWindowInterface\* *window*, const wxPoint& *pos*, const wxMouseEvent& *event*)**

This function is simple event handler. Each time the user clicks mouse button over a cell within *wxHtmlWindow* (p. 872) this method of that cell is called. Default behavior is to call *wxHtmlWindow::LoadPage* (p. 875).

#### **Note**

If you need more "advanced" event handling you should use *wxHtmlBinderCell* instead.

#### **Parameters**

*window*

interface to the parent HTML window

*pos*

coordinates of mouse click (this is relative to cell's origin)

*event*

mouse event that triggered the call

#### **Return value**

*true* if a link was clicked, *false* otherwise.

This function is new since wxWidgets version 2.7.0 (before *OnMouseClicked* method served a similar purpose).

### **wxHtmlCell::SetId**

**void SetId(const wxString& *id*)**

Sets unique cell identifier. Default value is no identifier, i.e. empty string.

### **wxHtmlCell::SetLink**

**void SetLink(const wxHtmlLinkInfo& *link*)**



Sets the hypertext link associated with this cell. (Default value is *wxHtmlLinkInfo* (p. 851)("", "")) (no link))

### **wxHtmlCell::SetNext**

**void SetNext(wxHtmlCell \*cell)**

Sets the next cell in the list. This shouldn't be called by user - it is to be used only by *wxHtmlContainerCell::InsertCell* (p. 827).

### **wxHtmlCell::SetParent**

**void SetParent(wxHtmlContainerCell \*p)**

Sets parent container of this cell. This is called from *wxHtmlContainerCell::InsertCell* (p. 827).

### **wxHtmlCell::SetPos**

**void SetPos(int x, int y)**

Sets the cell's position within parent container.

## **wxHtmlColourCell**

This cell changes the colour of either the background or the foreground.

### **Derived from**

*wxHtmlCell* (p. 819)

### **Include files**

<wx/html/htmlcell.h>

### **wxHtmlColourCell::wxHtmlColourCell**

**wxHtmlColourCell(wxColour clr, int flags = wxHTML\_CLR\_FOREGROUND)**

Constructor.

### **Parameters**

*clr*

The color

*flags*

Can be one of following:

<b>wxHTML_CLR_FOREGROUND</b>	change color of text
<b>wxHTML_CLR_BACKGROUND</b>	change background color

## wxHtmlContainerCell

The wxHtmlContainerCell class is an implementation of a cell that may contain more cells in it. It is heavily used in the wxHTML layout algorithm.

### Derived from

*wxHtmlCell* (p. 819)

### Include files

<wx/html/htmlcell.h>

### See Also

*Cells Overview* (p. 2182)

### wxHtmlContainerCell::wxHtmlContainerCell

**wxHtmlContainerCell**(wxHtmlContainerCell \**parent*)

Constructor. *parent* is pointer to parent container or NULL.

### wxHtmlContainerCell::GetAlignHor

**int** GetAlignHor() const

Returns container's horizontal alignment.

### wxHtmlContainerCell::GetAlignVer

**int** GetAlignVer() const

Returns container's vertical alignment.

### wxHtmlContainerCell::GetBackgroundColour

**wxColour** GetBackgroundColour()

Returns the background colour of the container or wxNullColour if no background colour is set.

**wxHtmlContainerCell::GetIndent****int GetIndent(int *ind*) const**

Returns the indentation. *ind* is one of the **wxHTML\_INDENT\_\*** constants.

**Note:** You must call *GetIndentUnits* (p. 827) with same *ind* parameter in order to correctly interpret the returned integer value. It is NOT always in pixels!

**wxHtmlContainerCell::GetIndentUnits****int GetIndentUnits(int *ind*) const**

Returns the units of indentation for *ind* where *ind* is one of the **wxHTML\_INDENT\_\*** constants.

**wxHtmlContainerCell::InsertCell****void InsertCell(wxHtmlCell \**cell*)**

Inserts new cell into the container.

**wxHtmlContainerCell::SetAlign****void SetAlign(const wxHtmlTag& *tag*)**

Sets the container's alignment (both horizontal and vertical) according to the values stored in *tag*. (Tags **ALIGN** parameter is extracted.) In fact it is only a front-end to *SetAlignHor* (p. 827) and *SetAlignVer* (p. 828).

**wxHtmlContainerCell::SetAlignHor****void SetAlignHor(int *a*)**

Sets the container's *horizontal alignment*. During *Layout* (p. 823) each line is aligned according to *a*/value.

**Parameters***a*

new horizontal alignment. May be one of these values:

<b>wxHTML_ALIGN_LEFT</b>	lines are left-aligned (default)
<b>wxHTML_ALIGN_JUSTIFY</b>	lines are justified
<b>wxHTML_ALIGN_CENTER</b>	lines are centered
<b>wxHTML_ALIGN_RIGHT</b>	lines are right-aligned

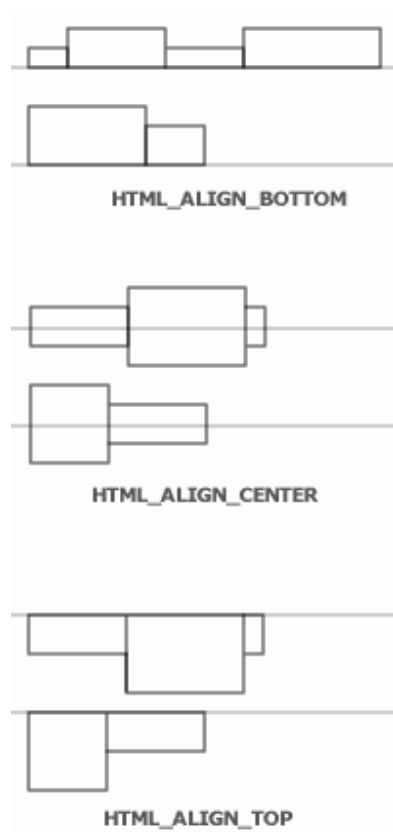
**wxHtmlContainerCell::SetAlignVer****void SetAlignVer(int *al*)**

Sets the container's *vertical alignment*. This is per-line alignment!

**Parameters***al*

new vertical alignment. May be one of these values:

<b>wxHTML_ALIGN_BOTTOM</b>	cells are over the line (default)
<b>wxHTML_ALIGN_CENTER</b>	cells are centered on line
<b>wxHTML_ALIGN_TOP</b>	cells are under the line

**wxHtmlContainerCell::SetBackgroundColour****void SetBackgroundColour(const wxColour& *clr*)**

Sets the background colour for this container.

**wxHtmlContainerCell::SetBorder**

**void SetBorder(const wxColour& *clr1*, const wxColour& *clr2*)**

Sets the border (frame) colours. A border is a rectangle around the container.

**Parameters**

*clr1*

Colour of top and left lines

*clr2*

Colour of bottom and right lines

**wxHtmlContainerCell::SetIndent**

**void SetIndent(int *i*, int *what*, int *units* = wxHTML\_UNITS\_PIXELS)**

Sets the indentation (free space between borders of container and subcells).

**Parameters**

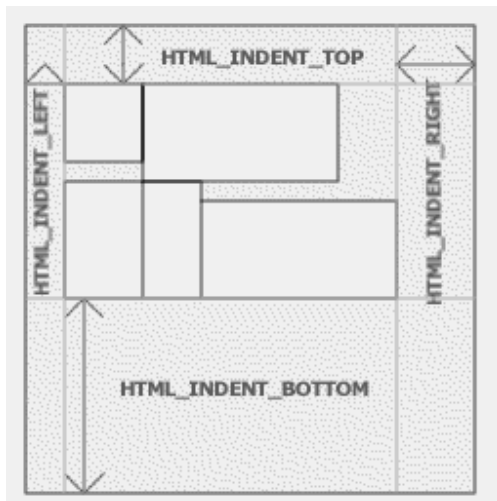
*i*

Indentation value.

*what*

Determines which of the four borders we're setting. It is OR combination of following constants:

<b>wxHTML_INDENT_TOP</b>	top border
<b>wxHTML_INDENT_BOTTOM</b>	bottom
<b>wxHTML_INDENT_LEFT</b>	left
<b>wxHTML_INDENT_RIGHT</b>	right
<b>wxHTML_INDENT_HORIZONTAL</b>	left and right
<b>wxHTML_INDENT_VERTICAL</b>	top and bottom
<b>wxHTML_INDENT_ALL</b>	all 4 borders



*units*

Units of *i*. This parameter affects interpretation of *i* value.

**wxHTML\_UNITS\_PIXELS** *i* is number of pixels

**wxHTML\_UNITS\_PERCENT** *i* is interpreted as percents of width of parent container

### **wxHtmlContainerCell::SetMinHeight**

**void SetMinHeight(int h, int align = wxHTML\_ALIGN\_TOP)**

Sets minimal height of the container.

When container's *Layout* (p. 823) is called, *m\_Height* is set depending on layout of subcells to the height of area covered by layed-out subcells. Calling this method guarantees you that the height of container is never smaller than *h* - even if the subcells cover much smaller area.

#### **Parameters**

*h*

The minimal height.

*align*

If height of the container is lower than the minimum height, empty space must be inserted somewhere in order to ensure minimal height. This parameter is one of **wxHTML\_ALIGN\_TOP**, **wxHTML\_ALIGN\_BOTTOM**, **wxHTML\_ALIGN\_CENTER**. It refers to the *contents*, not to the empty place.

### **wxHtmlContainerCell::SetWidthFloat**

**void SetWidthFloat(int *w*, int *units*)**

**void SetWidthFloat(const wxHtmlTag& *tag*, double *pixel\_scale* = 1.0)**

Sets floating width adjustment.

The normal behaviour of container is that its width is the same as the width of parent container (and thus you can have only one sub-container per line). You can change this by setting FWA.

*pixel\_scale* is number of real pixels that equals to 1 HTML pixel.

#### Parameters

*w*

Width of the container. If the value is negative it means complement to full width of parent container (e.g. `SetWidthFloat(-50, wxHTML_UNITS_PIXELS)` sets the width of container to parent's width minus 50 pixels. This is useful when creating tables - you can call `SetWidthFloat(50)` and `SetWidthFloat(-50)`)

*units*

Units of *w* This parameter affects the interpretation of *w* value.

**wxHTML\_UNITS\_PIXELS** *w* is number of pixels

**wxHTML\_UNITS\_PERCENT** *w* is interpreted as percents of width of parent container

*tag*

In the second version of method, *w* and *units* info is extracted from *tag*'s `WIDTH` parameter.

**wxPython note:** The second form of this method is named `SetWidthFloatFromTag` in wxPython.

## wxHtmlDCRenderer

This class can render HTML document into a specified area of a DC. You can use it in your own printing code, although use of *wxHtmlEasyPrinting* (p. 833) or *wxHtmlPrintout* (p. 863) is strongly recommended.

#### Derived from

*wxObject* (p. 1148)

#### Include files

<wx/html/htmprint.h>

**wxHtmlDCRenderer::wxHtmlDCRenderer****wxHtmlDCRenderer()**

Constructor.

**wxHtmlDCRenderer::SetDC****void SetDC(wxDC\* dc, double pixel\_scale = 1.0)**

Assign DC instance to the renderer.

*pixel\_scale* can be used when rendering to high-resolution DCs (e.g. printer) to adjust size of pixel metrics. (Many dimensions in HTML are given in pixels -- e.g. image sizes. 300x300 image would be only one inch wide on typical printer. With *pixel\_scale* = 3.0 it would be 3 inches.)

**wxHtmlDCRenderer::SetFont****void SetFont(const wxString& normal\_face, const wxString& fixed\_face, const int \*sizes = NULL)**Sets fonts. See *wxHtmlWindow::SetFont* (p. 879) for detailed description.See also *SetSize* (p. 832).**wxHtmlDCRenderer::SetSize****void SetSize(int width, int height)**

Set size of output rectangle, in pixels. Note that you **can't** change width of the rectangle between calls to *Render* (p. 833)! (You can freely change height.)

**wxHtmlDCRenderer::SetHtmlText****void SetHtmlText(const wxString& html, const wxString& basepath = wxEmptyString, bool isdir = true)**Assign text to the renderer. *Render* (p. 833) then draws the text onto DC.**Parameters***html*HTML text. This is *not* a filename.*basepath*

base directory (html string would be stored there if it was in file). It is used to determine path for loading images, for example.

*isdir*



false if basepath is filename, true if it is directory name (see *wxFileSystem* (p. 633) for detailed explanation)

### **wxHtmlDCRenderer::Render**

**int Render**(int *x*, int *y*, int *from* = 0, int *dont\_render* = false)

Renders HTML text to the DC.

#### **Parameters**

*x,y*

position of upper-left corner of printing rectangle (see *SetSize* (p. 832))

*from*

y-coordinate of the very first visible cell

*dont\_render*

if true then this method only returns y coordinate of the next page and does not output anything

Returned value is y coordinate of first cell than didn't fit onto page. Use this value as *from* in next call to *Render* in order to print multipages document.

#### **Caution!**

The Following three methods **must** always be called before any call to *Render* (preferably in this order):

- *SetDC* (p. 832)
- *SetSize* (p. 832)
- *SetHtmlText* (p. 832)

**Render() changes the DC's user scale and does NOT restore it.**

### **wxHtmlDCRenderer::GetTotalHeight**

**int GetTotalHeight**()

Returns the height of the HTML text. This is important if area height (see *SetSize* (p. 832)) is smaller than total height and thus the page cannot fit into it. In that case you're supposed to call *Render* (p. 833) as long as its return value is smaller than *GetTotalHeight*'s.

## **wxHtmlEasyPrinting**

This class provides very simple interface to printing architecture. It allows you to print HTML documents using only a few commands.

**Note**

Do not create this class on the stack only. You should create an instance on app startup and use this instance for all printing operations. The reason is that this class stores various settings in it.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/html/htmprint.h>

**wxHtmlEasyPrinting::wxHtmlEasyPrinting**

**wxHtmlEasyPrinting(const wxString& name = "Printing", wxWindow\* parentWindow = NULL)**

Constructor.

**Parameters**

*name*

Name of the printing object. Used by preview frames and setup dialogs.

*parentWindow*

pointer to the window that will own the preview frame and setup dialogs. May be NULL.

**wxHtmlEasyPrinting::GetParentWindow**

**wxWindow\* GetParentWindow() const**

Gets the parent window for dialogs.

**wxHtmlEasyPrinting::GetPrintData**

**wxPrintData\* GetPrintData()**

Returns pointer to *wxPrintData* (p. 1200) instance used by this class. You can set its parameters (via SetXXXX methods).

**wxHtmlEasyPrinting::GetPageSetupData**

**wxPageSetupDialogData\* GetPageSetupData()**

Returns a pointer to *wxPageSetupDialogData* (p. 1159) instance used by this class. You can set its parameters (via SetXXXX methods).

**wxHtmlEasyPrinting::PreviewFile****bool PreviewFile(const wxString& *htmlfile*)**

Preview HTML file.

Returns false in case of error -- call *wxPrinter::GetLastError* (p. 1212) to get detailed information about the kind of the error.

**wxHtmlEasyPrinting::PreviewText****bool PreviewText(const wxString& *htmltext*, const wxString& *basepath* = *wxEmptyString*)**

Preview HTML text (not file!).

Returns false in case of error -- call *wxPrinter::GetLastError* (p. 1212) to get detailed information about the kind of the error.

**Parameters***htmltext*

HTML text.

*basepath*

base directory (html string would be stored there if it was in file). It is used to determine path for loading images, for example.

**wxHtmlEasyPrinting::PrintFile****bool PrintFile(const wxString& *htmlfile*)**

Print HTML file.

Returns false in case of error -- call *wxPrinter::GetLastError* (p. 1212) to get detailed information about the kind of the error.

**wxHtmlEasyPrinting::PrintText****bool PrintText(const wxString& *htmltext*, const wxString& *basepath* = *wxEmptyString*)**

Print HTML text (not file!).

Returns false in case of error -- call *wxPrinter::GetLastError* (p. 1212) to get detailed information about the kind of the error.

**Parameters***htmltext*

HTML text.

*basepath*

base directory (html string would be stored there if it was in file). It is used to determine path for loading images, for example.

### **wxHtmlEasyPrinting::PageSetup**

**void PageSetup()**

Display page setup dialog and allows the user to modify settings.

### **wxHtmlEasyPrinting::SetFont**

**void SetFont(const wxString& normal\_face, const wxString& fixed\_face, const int \*sizes = NULL)**

Sets fonts. See *wxHtmlWindow::SetFont* (p. 879) for detailed description.

### **wxHtmlEasyPrinting::SetHeader**

**void SetHeader(const wxString& header, int pg = wxPAGE\_ALL)**

Set page header. The following macros can be used inside it:

- @DATE@ is replaced by the current date in default format
- @PAGENUM@ is replaced by page number
- @PAGESCNT@ is replaced by total number of pages
- @TIME@ is replaced by the current time in default format
- @TITLE@ is replaced with the title of the document

#### **Parameters**

*header*

HTML text to be used as header.

*pg*

one of wxPAGE\_ODD, wxPAGE\_EVEN and wxPAGE\_ALL constants.

### **wxHtmlEasyPrinting::SetFooter**

**void SetFooter(const wxString& footer, int pg = wxPAGE\_ALL)**

Set page footer. The following macros can be used inside it:

- @DATE@ is replaced by the current date in default format

- @PAGENUM@ is replaced by page number
- @PAGESCNT@ is replaced by total number of pages
- @TIME@ is replaced by the current time in default format
- @TITLE@ is replaced with the title of the document

**Parameters**

*footer*

HTML text to be used as footer.

*pg*

one of wxPAGE\_ODD, wxPAGE\_EVEN and wxPAGE\_ALL constants.

**wxHtmlEasyPrinting::SetParentWindow**

**void SetParentWindow(wxWindow\* window)**

Sets the parent window for dialogs.

**wxHtmlFilter**

This class is the parent class of input filters for *wxHtmlWindow* (p. 872). It allows you to read and display files of different file formats.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/html/htmlfilt.h>

**See Also**

*Overview* (p. 2182)

**wxHtmlFilter::wxHtmlFilter**

**wxHtmlFilter()**

Constructor.

**wxHtmlFilter::CanRead**

**bool CanRead(const wxFSFile& file)**

Returns true if this filter is capable of reading file *file*.

Example:

```
bool MyFilter::CanRead(const wxFSFile& file)
{
    return (file.GetMimeType() == "application/x-ugh");
}
```

### **wxHtmlFilter::ReadFile**

**wxString ReadFile(const wxFSFile& file)**

Reads the file and returns string with HTML document.

Example:

```
wxString MyImgFilter::ReadFile(const wxFSFile& file)
{
    return "<html><body><img src=\"\" +
        file.GetLocation() +
        \"\"></body></html>";
}
```

## **wxHtmlHelpController**

This help controller provides an easy way of displaying HTML help in your application (see *test* sample). The help system is based on **books** (see *AddBook* (p. 840)). A book is a logical section of documentation (for example "User's Guide" or "Programmer's Guide" or "C++ Reference" or "wxWidgets Reference"). The help controller can handle as many books as you want.

Although this class has an API compatible with other wxWidgets help controllers as documented by *wxHelpController* (p. 809), it is recommended that you use the enhanced capabilities of wxHtmlHelpController's API.

wxHTML uses Microsoft's HTML Help Workshop project files (.hhp, .hhk, .hhc) as its native format. The file format is described *here* (p. 2181). Have a look at docs/html/ directory where sample project files are stored.

You can use Tex2RTF to produce these files when generating HTML, if you set **htmlWorkshopFiles** to **true** in your tex2rtf.ini file. The commercial tool HelpBlocks (www.helpblocks.com) can also create these files.

### **Note**

It is strongly recommended to use preprocessed **.hhp.cached** version of projects. It can be either created on-the-fly (see *SetTempDir* (p. 842)) or you can use **hhp2cached** utility from *utils/hhp2cached* to create it and distribute the cached version together with helpfiles. See *samples/html/help* sample for demonstration of its use.

### **See also**

*Information about wxBestHelpController (p. 809), wxHtmlHelpFrame (p. 846), wxHtmlHelpDialog (p. 844), wxHtmlHelpWindow (p. 847), wxHtmlModalHelp (p. 851)*

**Derived from**

wxHelpControllerBase

**Include files**

<wx/html/helpctrl.h>

**wxHtmlHelpController::wxHtmlHelpController**

**wxHtmlHelpController**(int *style* = wxHF\_DEFAULT\_STYLE, **wxWindow\*** *parentWindow* = NULL)

Constructor.

**Parameters**

*style* is a combination of these flags:

<b>wxHF_TOOLBAR</b>	The help window has a toolbar.
<b>wxHF_FLAT_TOOLBAR</b>	The help window has a toolbar with flat buttons (aka coolbar).
<b>wxHF_CONTENTS</b>	The help window has a contents panel.
<b>wxHF_INDEX</b>	The help window has an index panel.
<b>wxHF_SEARCH</b>	The help window has a search panel.
<b>wxHF_BOOKMARKS</b>	The help window has bookmarks controls.
<b>wxHF_OPEN_FILES</b>	Allows user to open arbitrary HTML document.
<b>wxHF_PRINT</b>	The toolbar contains "print" button.
<b>wxHF_MERGE_BOOKS</b>	The contents pane does not show book nodes. All books are merged together and appear as single book to the user.
<b>wxHF_ICONS_BOOK</b>	All nodes in contents pane have a book icon. This is how Microsoft's HTML help viewer behaves.
<b>wxHF_ICONS_FOLDER</b>	Book nodes in contents pane have a book icon, book's sections have a folder icon. This is the default.
<b>wxHF_ICONS_BOOK_CHAPTER</b>	Both book nodes and nodes of top-level sections of a book (i.e. chapters) have a book

	icon, all other sections (sections, subsections, ...) have a folder icon.
<b>wxHF_EMBEDDED</b>	Specifies that the help controller controls an embedded window of class <i>wxHtmlHelpWindow</i> (p. 847) that should not be destroyed when the controller is destroyed.
<b>wxHF_DIALOG</b>	Specifies that the help controller should create a dialog containing the help window.
<b>wxHF_FRAME</b>	Specifies that the help controller should create a frame containing the help window. This is the default if neither <b>wxHF_DIALOG</b> nor <b>wxHF_EMBEDDED</b> is specified.
<b>wxHF_MODAL</b>	Specifies that the help controller should create a modal dialog containing the help window (used with the <b>wxHF_DIALOG</b> style).
<b>wxHF_DEFAULT_STYLE</b>	<div>wxHF_TOOLBAR   wxHF_CONTENTS   wxHF_INDEX   wxHF_SEARCH   wxHF_BOOKMARKS   wxHF_PRINT</div>

*parentWindow* is an optional window to be used as the parent for the help window.

### **wxHtmlHelpController::AddBook**

**bool AddBook(const wxString& *bookFile*, bool *showWaitMsg*)**

**bool AddBook(const wxString& *bookUrl*, bool *showWaitMsg*)**

Adds book (*.hhp file* (p. 2181) - HTML Help Workshop project file) into the list of loaded books. This must be called at least once before displaying any help.

*bookFile* or *bookUrl* may be either *.hhp* file or ZIP archive that contains arbitrary number of *.hhp* files in top-level directory. This ZIP archive must have *.zip* or *.htb* extension (the latter stands for "HTML book"). In other words, `AddBook(wxFileName("help.zip"))` is possible and is the recommended way.

#### **Parameters**

*showWaitMsg*

If true then a decoration-less window with progress message is displayed.

*bookFile*

Help book filename. It is recommended to use this prototype instead of the one taking URL, because it is less error-prone.

*bookUrl*

Help book URL (note that syntax of filename and URL is different on most platforms)



**Note**

Don't forget to install the archive wxFileSystem handler  
`withwxFileSystem::AddHandler(new wxArchiveFSHandler);` before calling this method on a .zip or .htb file!

**wxHtmlHelpController::CreateHelpDialog**

**virtual wxHtmlHelpDialog\* CreateHelpDialog(wxHtmlHelpData \* data)**

This protected virtual method may be overridden so that when specifying the `wxHF_DIALOG` style, the controller uses a different dialog.

**wxHtmlHelpController::CreateHelpFrame**

**virtual wxHtmlHelpFrame\* CreateHelpFrame(wxHtmlHelpData \* data)**

This protected virtual method may be overridden so that the controller uses a different frame.

**wxHtmlHelpController::Display**

**void Display(const wxString& x)**

Displays page x. This is THE important function - it is used to display the help in application.

You can specify the page in many ways:

- as direct filename of HTML document
- as chapter name (from contents) or as a book name
- as some word from index
- even as any word (will be searched)

Looking for the page runs in these steps:

1. try to locate file named x (if x is for example "doc/howto.htm")
2. try to open starting page of book named x
3. try to find x in contents (if x is for example "How To ...")
4. try to find x in index (if x is for example "How To ...")
5. switch to Search panel and start searching

**void Display(const int id)**

This alternative form is used to search help contents by numeric IDs.

**wxPython note:** The second form of this method is named `DisplayId` in wxPython.

### **wxHtmlHelpController::DisplayContents**

**void DisplayContents()**

Displays help window and focuses contents panel.

### **wxHtmlHelpController::DisplayIndex**

**void DisplayIndex()**

Displays help window and focuses index panel.

### **wxHtmlHelpController::KeywordSearch**

**bool KeywordSearch(const wxString& keyword, wxHelpSearchMode mode = wxHELP\_SEARCH\_ALL)**

Displays help window, focuses search panel and starts searching. Returns true if the keyword was found. Optionally it searches through the index (mode = wxHELP\_SEARCH\_INDEX), default the content (mode = wxHELP\_SEARCH\_ALL).

**Important:** KeywordSearch searches only pages listed in .hnc file(s). You should list all pages in the contents file.

### **wxHtmlHelpController::ReadCustomization**

**void ReadCustomization(wxConfigBase\* cfg, wxString path = wxEmptyString)**

Reads the controller's setting (position of window, etc.)

### **wxHtmlHelpController::SetTempDir**

**void SetTempDir(const wxString& path)**

Sets the path for storing temporary files - cached binary versions of index and contents files. These binary forms are much faster to read. Default value is empty string (empty string means that no cached data are stored). Note that these files are *not* deleted when program exits.

Once created these cached files will be used in all subsequent executions of your application. If cached files become older than corresponding .hnc file (e.g. if you regenerate documentation) it will be refreshed.

### **wxHtmlHelpController::SetTitleFormat**

**void SetTitleFormat(const wxString& format)**

Sets format of title of the frame. Must contain exactly one "%s" (for title of displayed HTML page).

**wxHtmlHelpController::UseConfig****void UseConfig(wxConfigBase\* config, const wxString& rootpath = wxEmptyString)**

Associates *config* object with the controller.

If there is associated config object, wxHtmlHelpController automatically reads and writes settings (including wxHtmlWindow's settings) when needed.

The only thing you must do is create wxConfig object and call UseConfig.

If you do not use *UseConfig*, wxHtmlHelpController will use default wxConfig object if available (for details see *wxConfigBase::Get* (p. 271) and *wxConfigBase::Set* (p. 275)).

**wxHtmlHelpController::WriteCustomization****void WriteCustomization(wxConfigBase\* cfg, wxString path = wxEmptyString)**

Stores controllers setting (position of window etc.)

**wxHtmlHelpData**

This class is used by *wxHtmlHelpController* (p. 838) and *wxHtmlHelpFrame* (p. 846) to access HTML help items. It is internal class and should not be used directly - except for the case you're writing your own HTML help controller.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/html/helpdata.h>

**wxHtmlHelpData::wxHtmlHelpData****wxHtmlHelpData()**

Constructor.

**wxHtmlHelpData::AddBook****bool AddBook(const wxString& book\_url)**

Adds new book. *book* is URL (not filename!) of HTML help project (hhp) or ZIP file that contains arbitrary number of .hhp projects (this zip file can have either .zip or .htb extension, htb stands for "html book"). Returns success.

**wxHtmlHelpData::FindPageById**

**wxString FindPageById(int id)**

Returns page's URL based on integer ID stored in project.

**wxHtmlHelpData::FindPageByName**

**wxString FindPageByName(const wxString& page)**

Returns page's URL based on its (file)name.

**wxHtmlHelpData::GetBookRecArray**

**const wxHtmlBookRecArray& GetBookRecArray()**

Returns array with help books info.

**wxHtmlHelpData::GetContentsArray**

**const wxHtmlHelpDataItems& GetContentsArray()**

Returns reference to array with contents entries.

**wxHtmlHelpData::GetIndexArray**

**const wxHtmlHelpDataItems& GetIndexArray()**

Returns reference to array with index entries.

**wxHtmlHelpData::SetTempDir**

**void SetTempDir(const wxString& path)**

Sets temporary directory where binary cached versions of MS HTML Workshop files will be stored. (This is turned off by default and you can enable this feature by setting non-empty temp dir.)

## **wxHtmlHelpDialog**

This class is used by *wxHtmlHelpController* (p. 838) to display help. It is an internal class and should not be used directly - except for the case when you're writing your own HTML help controller.

**Derived from**

*wxFrame* (p. 682)

**Include files**

<wx/html/helpdlg.h>

**wxHtmlHelpDialog::wxHtmlHelpDialog****wxHtmlHelpDialog(wxHtmlHelpData\* data = NULL)****wxHtmlHelpDialog(wxWindow\* parent, int wxWindowID, const wxString& title = wxEmptyString, int style = wxHF\_DEFAULT\_STYLE, wxHtmlHelpData\* data = NULL)**

Constructor. For the values of *style*, please see the documentation for *wxHtmlHelpController* (p. 838).

**wxHtmlHelpDialog::AddToolBarButtons****virtual void AddToolBarButtons(wxToolBar \*toolBar, int style)**

You may override this virtual method to add more buttons to the help window's toolbar. *toolBar* is a pointer to the toolbar and *style* is the style flag as passed to the *Create* method.

*wxToolBar::Realize* is called immediately after returning from this function.

**wxHtmlHelpDialog::Create****bool Create(wxWindow\* parent, wxWindowID id, const wxString& title = wxEmptyString, int style = wxHF\_DEFAULT\_STYLE)**

Creates the dialog. See *the constructor* (p. 845) for a description of the parameters.

**wxHtmlHelpDialog::GetController****wxHtmlHelpController\* GetController() const**

Returns the help controller associated with the dialog.

**wxHtmlHelpDialog::ReadCustomization****void ReadCustomization(wxConfigBase\* cfg, const wxString& path = wxEmptyString)**

Reads the user's settings for this dialog see *wxHtmlHelpController::ReadCustomization* (p. 842))

**wxHtmlHelpDialog::SetController****void SetController(wxHtmlHelpController\* controller)**

Sets the help controller associated with the dialog.

**wxHtmlHelpDialog::SetTitleFormat****void SetTitleFormat(const wxString& format)**

Sets the dialog's title format. *format* must contain exactly one "%s" (it will be replaced by the page title).

### **wxHtmlHelpDialog::WriteCustomization**

**void WriteCustomization**(wxConfigBase\* *cfg*, const wxString& *path* = wxEmptyString)

Saves the user's settings for this dialog (see *wxHtmlHelpController::WriteCustomization* (p. 843)).

## **wxHtmlHelpFrame**

This class is used by *wxHtmlHelpController* (p. 838) to display help. It is an internal class and should not be used directly - except for the case when you're writing your own HTML help controller.

### **Derived from**

*wxFrame* (p. 682)

### **Include files**

<wx/html/helpfrm.h>

### **wxHtmlHelpFrame::wxHtmlHelpFrame**

**wxHtmlHelpFrame**(wxHtmlHelpData\* *data* = NULL)

**wxHtmlHelpFrame**(wxWindow\* *parent*, int *wxWindowID*, const wxString& *title* = wxEmptyString, int *style* = wxHF\_DEFAULT\_STYLE, wxHtmlHelpData\* *data* = NULL)

Constructor. For the values of *style*, please see the documentation for *wxHtmlHelpController* (p. 838).

### **wxHtmlHelpFrame::AddToolBarButtons**

**virtual void AddToolBarButtons**(wxToolBar\* *toolBar*, int *style*)

You may override this virtual method to add more buttons to the help window's toolbar. *toolBar* is a pointer to the toolbar and *style* is the style flag as passed to the *Create* method.

*wxToolBar::Realize* is called immediately after returning from this function.

### **wxHtmlHelpFrame::Create**

**bool Create**(wxWindow\* *parent*, wxWindowID *id*, const wxString& *title* = wxEmptyString, int *style* = wxHF\_DEFAULT\_STYLE)

Creates the frame. See *the constructor* (p. 846) for a description of the parameters.

### **wxHtmlHelpFrame::GetController**

**wxHtmlHelpController\* GetController() const**

Returns the help controller associated with the frame.

### **wxHtmlHelpFrame::ReadCustomization**

**void ReadCustomization(wxConfigBase\* cfg, const wxString& path = wxEmptyString)**

Reads the user's settings for this frame see *wxHtmlHelpController::ReadCustomization* (p. 842))

### **wxHtmlHelpFrame::SetController**

**void SetController(wxHtmlHelpController\* controller)**

Sets the help controller associated with the frame.

### **wxHtmlHelpFrame::SetTitleFormat**

**void SetTitleFormat(const wxString& format)**

Sets the frame's title format. *format* must contain exactly one "%s" (it will be replaced by the page title).

### **wxHtmlHelpFrame::WriteCustomization**

**void WriteCustomization(wxConfigBase\* cfg, const wxString& path = wxEmptyString)**

Saves the user's settings for this frame (see *wxHtmlHelpController::WriteCustomization* (p. 843)).

## **wxHtmlHelpWindow**

This class is used by *wxHtmlHelpController* (p. 838) to display help within a frame or dialog, but you can use it yourself to create an embedded HTML help window.

For example:

```
// m_embeddedHelpWindow is a wxHtmlHelpWindow
// m_embeddedHtmlHelp is a wxHtmlHelpController

// Create embedded HTML Help window
m_embeddedHelpWindow = new wxHtmlHelpWindow;
m_embeddedHtmlHelp.UseConfig(config, rootPath); // Set your own
```

```
config object here
    m_embeddedHtmlHelp.SetHelpWindow(m_embeddedHelpWindow);
    m_embeddedHelpWindow->Create(this,
        wxID_ANY, wxDefaultPosition, GetClientSize(),
        wxTAB_TRAVERSAL|wxNO_BORDER, wxHF_DEFAULT_STYLE);
    m_embeddedHtmlHelp.AddBook(wxFileName(_T("doc.zip")));
```

You should pass the style `wxHF_EMBEDDED` to the style parameter of `wxHtmlHelpController` to allow the embedded window to be destroyed independently of the help controller.

### Derived from

`wxWindow` (p. 1795)

### Include files

`<wx/html/helpwnd.h>`

## **wxHtmlHelpWindow::wxHtmlHelpWindow**

**wxHtmlHelpWindow(wxHtmlHelpData\* data = NULL)**

**wxHtmlHelpWindow(wxWindow\* parent, int wxWindowID, const wxPoint& pos = wxDefaultPosition, const wxSize& pos = wxDefaultSize, int style = wxTAB\_TRAVERSAL|wxTAB\_NO\_BORDER, int helpStyle = wxHF\_DEFAULT\_STYLE, wxHtmlHelpData\* data = NULL)**

Constructor.

Constructor. For the values of *helpStyle*, please see the documentation for *wxHtmlHelpController* (p. 838).

## **wxHtmlHelpWindow::Create**

**bool Create(wxWindow\* parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& pos = wxDefaultSize, int style = wxTAB\_TRAVERSAL|wxTAB\_NO\_BORDER, int helpStyle = wxHF\_DEFAULT\_STYLE, wxHtmlHelpData\* data = NULL)**

Creates the help window. See *the constructor* (p. 848) for a description of the parameters.

## **wxHtmlHelpWindow::CreateContents**

**void CreateContents()**

Creates contents panel. (May take some time.)

Protected.



**wxHtmlHelpWindow::CreateIndex****void CreateIndex()**

Creates index panel. (May take some time.)

Protected.

**wxHtmlHelpWindow::CreateSearch****void CreateSearch()**

Creates search panel.

**wxHtmlHelpWindow::Display****bool Display(const wxString& x)****bool Display(const int id)**

Displays page x. If not found it will give the user the choice of searching books. Looking for the page runs in these steps:

1. try to locate file named x (if x is for example "doc/howto.htm")
2. try to open starting page of book x
3. try to find x in contents (if x is for example "How To ...")
4. try to find x in index (if x is for example "How To ...")

The second form takes numeric ID as the parameter. (uses extension to MS format, <param name="ID" value=id>)

**wxPython note:** The second form of this method is named DisplayId in wxPython.

**wxHtmlHelpWindow::DisplayContents****bool DisplayContents()**

Displays contents panel.

**wxHtmlHelpWindow::DisplayIndex****bool DisplayIndex()**

Displays index panel.

**wxHtmlHelpWindow::GetData****wxHtmlHelpData\* GetData()**

Returns the `wxHtmlHelpData` object, which is usually a pointer to the controller's data.

### **wxHtmlHelpWindow::KeywordSearch**

**bool KeywordSearch(const wxString& keyword, wxHelpSearchMode mode = wxHELP\_SEARCH\_ALL)**

Search for given keyword. Optionally it searches through the index (mode = wxHELP\_SEARCH\_INDEX), default the content (mode = wxHELP\_SEARCH\_ALL).

### **wxHtmlHelpWindow::ReadCustomization**

**void ReadCustomization(wxConfigBase\* cfg, const wxString& path = wxEmptyString)**

Reads the user's settings for this window (see `wxHtmlHelpController::ReadCustomization` (p. 842))

### **wxHtmlHelpWindow::RefreshLists**

**void RefreshLists()**

Refresh all panels. This is necessary if a new book was added.

Protected.

### **wxHtmlHelpWindow::SetTitleFormat**

**void SetTitleFormat(const wxString& format)**

Sets the frame's title format. *format* must contain exactly one "%s" (it will be replaced by the page title).

### **wxHtmlHelpWindow::UseConfig**

**void UseConfig(wxConfigBase\* config, const wxString& rootpath = wxEmptyString)**

Associates a `wxConfig` object with the help window. It is recommended that you use `wxHtmlHelpController::UseConfig` (p. 843) instead.

### **wxHtmlHelpWindow::WriteCustomization**

**void WriteCustomization(wxConfigBase\* cfg, const wxString& path = wxEmptyString)**

Saves the user's settings for this window (see `wxHtmlHelpController::WriteCustomization` (p. 843)).

### **wxHtmlHelpWindow::AddToolBarButtons**

**virtual void AddToolBarButtons(wxToolBar \*toolBar, int style)**

You may override this virtual method to add more buttons to the help window's toolbar. *toolBar* is a pointer to the toolbar and *style* is the style flag as passed to the `Create` method.

`wxToolBar::Realize` is called immediately after returning from this function.

See *samples/html/helpview* for an example.

## wxHtmlModalHelp

This class uses *wxHtmlHelpController* (p. 838) to display help in a modal dialog. This is useful on platforms such as `wxMac` where if you display help from a modal dialog, the help window must itself be a modal dialog.

Create objects of this class on the stack, for example:

```
// The help can be browsed during the lifetime of this object; when
the user quits
// the help, program execution will continue.
wxHtmlModalHelp help(parent, wxT("help"), wxT("My topic"));
```

### Derived from

None

### Include files

<wx/html/helpctrl.h>

## wxHtmlModalHelp::wxHtmlModalHelp

**wxHtmlModalHelp(wxWindow\* parent, const wxString& helpFile, const wxString& topic = wxEmptyString, int style = wxHF\_DEFAULT\_STYLE | wxHF\_DIALOG | wxHF\_MODAL)**

### Parameters

*parent* is the parent of the dialog.

*helpFile* is the HTML help file to show.

*topic* is an optional topic. If this is empty, the help contents will be shown.

*style* is a combination of the flags described in the *wxHtmlHelpController* (p. 838) documentation.

## wxHtmlLinkInfo

This class stores all necessary information about hypertext links (as represented by <A> tag in HTML documents). In current implementation it stores URL and target frame name. *Note that frames are not currently supported by wxHTML!*

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/html/htmlcell.h>

**wxHtmlLinkInfo::wxHtmlLinkInfo**

**wxHtmlLinkInfo()**

Default ctor.

**wxHtmlLinkInfo(const wxString& href, const wxString& target = wxEmptyString)**

Construct hypertext link from HREF (aka URL) and TARGET (name of target frame).

**wxHtmlLinkInfo::GetEvent**

**const wxMouseEvent \* GetEvent()**

Return pointer to event that generated OnLinkClicked event. Valid only within *wxHtmlWindow::OnLinkClicked* (p. 876), NULL otherwise.

**wxHtmlLinkInfo::GetHtmlCell**

**const wxHtmlCell \* GetHtmlCell()**

Return pointer to the cell that was clicked. Valid only within *wxHtmlWindow::OnLinkClicked* (p. 876), NULL otherwise.

**wxHtmlLinkInfo::GetHref**

**wxString GetHref()**

Return *HREF* value of the <A> tag.

**wxHtmlLinkInfo::GetTarget**

**wxString GetTarget()**

Return *TARGET* value of the <A> tag (this value is used to specify in which frame should be the page pointed by *Href* (p. 852) opened).

## wxHtmlListBox

wxHtmlListBox is an implementation of *wxVListBox* (p. 1783) which shows HTML content in the listbox rows. This is still an abstract base class and you will need to derive your own class from it (see *htlbox* sample for the example) but you will only need to override a single *OnGetItem()* (p. 855) function.

### Derived from

*wxVListBox* (p. 1783)  
*wxVScrolledWindow* (p. 1790)  
*wxPanel* (p. 1170)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/htmlbox.h>

### See also

*wxSimpleHtmlListBox* (p. 855)

### Event handling

To process input from a *wxHtmlListBox*, use these event handler macros to direct input to member functions that take a *wxHtmlCellEvent* (p. 882) argument or a *wxHtmlLinkEvent* (p. 881).

**EVT\_HTML\_CELL\_CLICKED(id, func)** A *wxHtmlCell* (p. 819) was clicked.

**EVT\_HTML\_CELL\_HOVER(id, func)** The mouse passed over a *wxHtmlCell* (p. 819).

**EVT\_HTML\_LINK\_CLICKED(id, func)** A *wxHtmlCell* (p. 819) which contains an hyperlink was clicked.

### wxHtmlListBox::wxHtmlListBox

**wxHtmlListBox(wxWindow\* parent, wxWindowID id = wxID\_ANY, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0, const wxString& name = wxHtmlListBoxNameStr)**

Normal constructor which calls *Create()* (p. 854) internally.

**wxHtmlListBox()**

Default constructor, you must call *Create()* (p. 854) later.

### wxHtmlListBox::~~wxHtmlListBox

**~wxHtmlListBox()**

Destructor cleans up whatever resources we use.

**wxHtmlListBox::Create**

**bool Create**(wxWindow\* *parent*, wxWindowID *id* = wxID\_ANY, const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size* = wxDefaultSize, long *style* = 0, const wxString& *name* = wxHtmlListBoxNameStr)

Creates the control and optionally sets the initial number of items in it (it may also be set or changed later with *SetItemCount()* (p. 1789)).

There are no special styles defined for wxHtmlListBox, in particular the wxListBox styles (with the exception of wxLB\_MULTIPLE) can not be used here.

Returns *true* on success or *false* if the control couldn't be created

**wxHtmlListBox::GetFileSystem**

**wxFileSystem& GetFileSystem()**

**const wxFileSystem& GetFileSystem() const**

Returns the *wxFileSystem* (p. 633) used by the HTML parser of this object. The file system object is used to resolve the paths in HTML fragments displayed in the control and you should use *wxFileSystem::ChangePathTo* (p. 634) if you use relative paths for the images or other resources embedded in your HTML.

**wxHtmlListBox::GetSelectedTextBgColour**

**wxColour GetSelectedTextBgColour(const wxColour& colBg) const**

This virtual function may be overridden to change the appearance of the background of the selected cells in the same way as *GetSelectedTextColour* (p. 854).

It should be rarely, if ever, used because *SetSelectionBackground* (p. 1790) allows to change the selection background for all cells at once and doing anything more fancy is probably going to look strangely.

**See also**

*GetSelectedTextColour* (p. 854)

**wxHtmlListBox::GetSelectedTextColour**

**wxColour GetSelectedTextColour(const wxColour& colFg) const**

This virtual function may be overridden to customize the appearance of the selected cells. It is used to determine how the colour *colFg* is going to look inside selection. By default all original colours are completely ignored and the standard, system-dependent, selection colour is used but the program may wish to override this to achieve some custom

appearance.

### See also

*GetSelectedTextBgColour* (p. 854),  
*SetSelectionBackground* (p. 1790),  
*wxSystemSettings::GetColour* (p. 1594)

## **wxHtmlListBox::OnGetItem**

### **wxString OnGetItem(size\_t n) const**

This method must be implemented in the derived class and should return the body (i.e. without `<html>` nor `<body>` tags) of the HTML fragment for the given item.

Note that this function should always return a text fragment for the *n* item which renders with the same height both when it is selected and when it's not: i.e. if you call, inside your `OnGetItem()` implementation, `IsSelected(n)` to make the items appear differently when they are selected, then you should make sure that the returned HTML fragment will render with the same height or else you'll see some artifacts when the user selects an item.

## **wxHtmlListBox::OnGetItemMarkup**

### **wxString OnGetItemMarkup(size\_t n) const**

This function may be overridden to decorate HTML returned by `OnGetItem()` (p. 855).

## **wxHtmlListBox::OnLinkClicked**

### **virtual void OnLinkClicked(size\_t n, const wxHtmlLinkInfo& link)**

Called when the user clicks on hypertext link. Does nothing by default. Overloading this method is deprecated; intercept the event instead.

### Parameters

*n*

Index of the item containing the link.

*link*

Description of the link.

### See also

See also *wxHtmlLinkInfo* (p. 851).

## **wxSimpleHtmlListBox**

`wxSimpleHtmlListBox` is an implementation of `wxHtmlListBox` (p. 853) which shows HTML content in the listbox rows.

Unlike `wxHtmlListBox` (p. 853), this is not an abstract class and thus it has the advantage that you can use it without deriving your own class from it. However, it also has the disadvantage that this is not a virtual control and thus it's not well-suited for those cases where you need to show a huge number of items: every time you add/insert a string, it will be stored internally and thus will take memory.

The interface exposed by `wxSimpleHtmlListBox` fully implements the `wxControlWithItems` (p. 286) interface, thus you should refer to `wxControlWithItems` (p. 286)'s documentation for the API reference for adding/removing/retrieving items in the listbox. Also note that the `wxVListBox::SetItemCount` (p. 1789) function is `protected` in `wxSimpleHtmlListBox`'s context so that you cannot call it directly, `wxSimpleHtmlListBox` will do it for you.

Note: in case you need to append a lot of items to the control at once, make sure to use the `Append(const wxArrayString &)` (p. 287) function.

Thus the only difference between a `wxListBox` (p. 974) and a `wxSimpleHtmlListBox` is that the latter stores strings which can contain HTML fragments (see the list of *tags supported by wxHTML* (p. 2186)).

Note that the HTML strings you fetch to `wxSimpleHtmlListBox` should not contain the `<html>` or `<body>` tags.

### Derived from

`wxHtmlListBox` (p. 853), `wxControlWithItems` (p. 286)  
`wxVListBox` (p. 1783)  
`wxVScrolledWindow` (p. 1790)  
`wxPanel` (p. 1170)  
`wxWindow` (p. 1795)  
`wxEvtHandler` (p. 576)  
`wxObject` (p. 1148)

### Include files

`<wx/htmlbox.h>`

### Window styles

**wxHLB\_DEFAULT\_STYLE** The default style: `wxSUNKEN_BORDER`

**wxHLB\_MULTIPLE** Multiple-selection list: the user can toggle multiple items on and off.

See also *window styles overview* (p. 2089).

### Event handling

A `wxSimpleHtmlListBox` emits the same events used by `wxListBox` (p. 974) and by `wxHtmlListBox` (p. 853).

The event handlers for the following events take a `wxCommandEvent` (p. 250):



<b>EVT_LISTBOX(id, func)</b>	Process a <code>wxEVT_COMMAND_LISTBOX_SELECTED</code> event, when an item on the list is selected.
<b>EVT_LISTBOX_DCLICK(id, func)</b>	Process a <code>wxEVT_COMMAND_LISTBOX_DOUBLECLICKED</code> event, when the listbox is double-clicked.

The event handlers for the following events take a *wxHtmlCellEvent* (p. 882) or a *wxHtmlLinkEvent* (p. 881):

<b>EVT_HTML_CELL_CLICKED(id, func)</b>	A <i>wxHtmlCell</i> (p. 819) was clicked.
<b>EVT_HTML_CELL_HOVER(id, func)</b>	The mouse passed over a <i>wxHtmlCell</i> (p. 819).
<b>EVT_HTML_LINK_CLICKED(id, func)</b>	A <i>wxHtmlCell</i> (p. 819) which contains an hyperlink was clicked.

### **wxSimpleHtmlListBox::wxSimpleHtmlListBox**

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

**wxHtmlListBox**(*wxWindow\** parent, *wxWindowID* id, **const wxPoint&** pos, **const wxSize&** size, **const wxArrayString&** choices, **long** style = *wxHLB\_DEFAULT\_STYLE*, **const wxValidator&** validator = *wxDefaultValidator*, **const wxString&** name = *"simpleHtmlListBox"*)

Constructor, creating and showing the HTML list box.

#### **Parameters**

*parent*

Parent window. Must not be *NULL*.

*id*

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

*pos*

Window position.

*size*

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

*n*

Number of strings with which to initialise the control.

*choices*

An array of strings with which to initialise the control.

*style*

Window style. See `wxHLB_*` flags.

*validator*

Window validator.

*name*

Window name.

### See also

*wxSimpleHtmlListBox::Create* (p. 858)

**wxSimpleHtmlListBox()**

Default constructor, you must call *Create()* (p. 858) later.

### **wxSimpleHtmlListBox::~wxSimpleHtmlListBox**

**~wxSimpleHtmlListBox()**

Frees the array of stored items and relative client data.

### **wxSimpleHtmlListBox::Create**

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

```
bool Create(wxWindow* parent, wxWindowID id, const wxPoint& pos, const  
wxSize& size, const wxString& choices, long style = wxHLB_DEFAULT_STYLE,  
const wxValidator& validator = wxDefaultValidator, const wxString& name =  
"simpleHtmlListBox")
```

Creates the HTML listbox for two-step construction. See *wxSimpleHtmlListBox::wxSimpleHtmlListBox* (p. 857) for further details.

## **wxHtmlParser**

Classes derived from this handle the **generic** parsing of HTML documents: it scans the document and divide it into blocks of tags (where one block consists of beginning and ending tag and of text between these two tags).

It is independent from `wxHtmlWindow` and can be used as stand-alone parser (Julian Smart's idea of speech-only HTML viewer or `wget`-like utility - see `InetGet` sample for example).

It uses system of tag handlers to parse the HTML document. Tag handlers are not statically shared by all instances but are created for each `wxHtmlParser` instance. The reason is that the handler may contain document-specific temporary data used during parsing (e.g. complicated structures like tables).

Typically the user calls only the *Parse* (p. 861) method.

### **Derived from**

`wxObject`

### **Include files**

`<wx/html/htmlpars.h>`

### **See also**

*Cells Overview* (p. 2182), *Tag Handlers Overview* (p. 2183), *wxHtmlTag* (p. 865)

## **wxHtmlParser::wxHtmlParser**

**wxHtmlParser()**

Constructor.

## **wxHtmlParser::AddTag**

**void AddTag(const wxHtmlTag& tag)**

This may (and may not) be overwritten in derived class.

This method is called each time new tag is about to be added. *tag* contains information about the tag. (See *wxHtmlTag* (p. 865) for details.)

Default (`wxHtmlParser`) behaviour is this: First it finds a handler capable of handling this tag and then it calls handler's `HandleTag` method.

## **wxHtmlParser::AddTagHandler**

**virtual void AddTagHandler(wxHtmlTagHandler \*handler)**

Adds handler to the internal list (& hash table) of handlers. This method should not be called directly by user but rather by derived class' constructor.

This adds the handler to this **instance** of `wxHtmlParser`, not to all objects of this class! (Static front-end to `AddTagHandler` is provided by `wxHtmlWinParser`).

All handlers are deleted on object deletion.

### **wxHtmlParser::AddText**

**virtual void AddWord(const char\* txt)**

Must be overwritten in derived class.

This method is called by *DoParsing* (p. 860) each time a part of text is parsed. *txt* is NOT only one word, it is substring of input. It is not formatted or preprocessed (so white spaces are unmodified).

### **wxHtmlParser::DoParsing**

**void DoParsing(int begin\_pos, int end\_pos)**

**void DoParsing()**

Parses the *m\_Source* from *begin\_pos* to *end\_pos*-1. (in *noparams* version it parses whole *m\_Source*)

### **wxHtmlParser::DoneParser**

**virtual void DoneParser()**

This must be called after *DoParsing()*.

### **wxHtmlParser::GetFS**

**wxFileSystem\* GetFS() const**

Returns pointer to the file system. Because each tag handler has reference to it is parent parser it can easily request the file by calling

```
wxFSFile *f = m_Parser -> GetFS() -> OpenFile("image.jpg");
```

### **wxHtmlParser::GetProduct**

**virtual wxObject\* GetProduct()**

Returns product of parsing. Returned value is result of parsing of the document. The type of this result depends on internal representation in derived parser (but it must be derived from *wxObject!*).

See *wxHtmlWinParser* for details.

### **wxHtmlParser::GetSource**

**wxString\* GetSource()**

Returns pointer to the source being parsed.

**wxHtmlParser::InitParser****virtual void InitParser(const wxString& source)**

Setups the parser for parsing the *source* string. (Should be overridden in derived class)

**wxHtmlParser::OpenURL****virtual wxFSFile\* OpenURL(wxHtmlURLType type, const wxString& url)**

Opens given URL and returns `wxFSFile` object that can be used to read data from it. This method may return NULL in one of two cases: either the URL doesn't point to any valid resource or the URL is blocked by overridden implementation of *OpenURL* in derived class.

**Parameters***type*

Indicates type of the resource. Is one of:

<b>wxHTML_URL_PAGE</b>	Opening a HTML page.
<b>wxHTML_URL_IMAGE</b>	Opening an image.
<b>wxHTML_URL_OTHER</b>	Opening a resource that doesn't fall into any other category.

*url*

URL being opened.

**Notes**

Always use this method in tag handlers instead of `GetFS() -> OpenFile()` because it can block the URL and is thus more secure.

Default behaviour is to call `wxHtmlWindow::OnOpeningURL` (p. 877) of the associated `wxHtmlWindow` object (which may decide to block the URL or redirect it to another one), if there's any, and always open the URL if the parser is not used with `wxHtmlWindow`.

Returned `wxFSFile` object is not guaranteed to point to *url*, it might have been redirected!

**wxHtmlParser::Parse****wxObject\* Parse(const wxString& source)**

Proceeds parsing of the document. This is end-user method. You can simply call it when you need to obtain parsed output (which is parser-specific)

The method does these things:

1. calls *InitParser(source)* (p. 861)
2. calls *DoParsing* (p. 860)
3. calls *GetProduct* (p. 860)
4. calls *DoneParser* (p. 860)
5. returns value returned by *GetProduct*

You shouldn't use *InitParser*, *DoParsing*, *GetProduct* or *DoneParser* directly.

### **wxHtmlParser::PushTagHandler**

**void PushTagHandler(wxHtmlTagHandler\* handler, const wxString& tags)**

Forces the handler to handle additional tags (not returned by *GetSupportedTags* (p. 869)). The handler should already be added to this parser.

#### **Parameters**

*handler*

the handler

*tags*

List of tags (in same format as *GetSupportedTags*'s return value). The parser will redirect these tags to *handler* (until call to *PopTagHandler* (p. 863)).

#### **Example**

Imagine you want to parse following pseudo-html structure:

```
<myitems>
  <param name="one" value="1">
  <param name="two" value="2">
</myitems>

<execute>
  <param program="text.exe">
</execute>
```

It is obvious that you cannot use only one tag handler for *<param>* tag. Instead you must use context-sensitive handlers for *<param>* inside *<myitems>* and *<param>* inside *<execute>*.

This is the preferred solution:

```
TAG_HANDLER_BEGIN(MYITEM, "MYITEMS")
TAG_HANDLER_PROC(tag)
{
    // ...something...

    m_Parser -> PushTagHandler(this, "PARAM");
}
```

```
        ParseInner(tag);
        m_Parser -> PopTagHandler();

        // ...something...
    }
    TAG_HANDLER_END(MYITEM)
```

### **wxHtmlParser::PopTagHandler**

#### **void PopTagHandler()**

Restores parser's state before last call to *PushTagHandler* (p. 862).

### **wxHtmlParser::SetFS**

#### **void SetFS(wxFileSystem \*fs)**

Sets the virtual file system that will be used to request additional files. (For example <IMG> tag handler requests wxFSFile with the image data.)

### **wxHtmlParser::StopParsing**

#### **void StopParsing()**

Call this function to interrupt parsing from a tag handler. No more tags will be parsed afterward. This function may only be called from *wxHtmlParser::Parse* (p. 861) or any function called by it (i.e. from tag handlers).

## **wxHtmlPrintout**

This class serves as printout class for HTML documents.

#### **Derived from**

*wxPrintout* (p. 1214)

#### **Include files**

<wx/html/htmprint.h>

### **wxHtmlPrintout::wxHtmlPrintout**

**wxHtmlPrintout(const wxString& title = "Printout")**

Constructor.

### **wxHtmlPrintout::AddFilter**

**static void AddFilter(wxHtmlFilter\* filter)**

Adds a filter to the static list of filters for wxHtmlPrintout. See *wxHtmlFilter* (p. 837) for further information.

**wxHtmlPrintout::SetFont**

**void SetFont(const wxString& normal\_face, const wxString& fixed\_face, const int \*sizes = NULL)**

Sets fonts. See *wxHtmlWindow::SetFont* (p. 879) for detailed description.

**wxHtmlPrintout::SetFooter**

**void SetFooter(const wxString& footer, int pg = wxPAGE\_ALL)**

Set page footer. The following macros can be used inside it:

- @DATE@ is replaced by the current date in default format
- @PAGENUM@ is replaced by page number
- @PAGESCNT@ is replaced by total number of pages
- @TIME@ is replaced by the current time in default format
- @TITLE@ is replaced with the title of the document

**Parameters**

*footer*

HTML text to be used as footer.

*pg*

one of wxPAGE\_ODD, wxPAGE\_EVEN and wxPAGE\_ALL constants.

**wxHtmlPrintout::SetHeader**

**void SetHeader(const wxString& header, int pg = wxPAGE\_ALL)**

Set page header. The following macros can be used inside it:

- @DATE@ is replaced by the current date in default format
- @PAGENUM@ is replaced by page number
- @PAGESCNT@ is replaced by total number of pages
- @TIME@ is replaced by the current time in default format
- @TITLE@ is replaced with the title of the document



**Parameters**

*header*

HTML text to be used as header.

*pg*

one of wxPAGE\_ODD, wxPAGE\_EVEN and wxPAGE\_ALL constants.

**wxHtmlPrintout::SetHtmlFile**

**void SetHtmlFile(const wxString& *htmlfile*)**

Prepare the class for printing this HTML **file**. The file may be located on any virtual file system or it may be normal file.

**wxHtmlPrintout::SetHtmlText**

**void SetHtmlText(const wxString& *html*, const wxString& *basepath* = wxEmptyString, bool *isdir* = true)**

Prepare the class for printing this HTML text.

**Parameters**

*html*

HTML text. (NOT file!)

*basepath*

base directory (html string would be stored there if it was in file). It is used to determine path for loading images, for example.

*isdir*

false if basepath is filename, true if it is directory name (see *wxFileSystem* (p. 633) for detailed explanation)

**wxHtmlPrintout::SetMargins**

**void SetMargins(float *top* = 25.2, float *bottom* = 25.2, float *left* = 25.2, float *right* = 25.2, float *spaces* = 5)**

Sets margins in millimeters. Defaults to 1 inch for margins and 0.5cm for space between text and header and/or footer

**wxHtmlTag**

This class represents a single HTML tag. It is used by *tag handlers* (p. 2183).

**Derived from**

wxObject

**Include files**

<wx/html/htmltag.h>

**wxHtmlTag::wxHtmlTag**

**wxHtmlTag(wxHtmlTag \*parent, const wxString& source, int pos, int end\_pos, wxHtmlTagsCache\* cache, wxHtmlEntitiesParser \*entParser)**

Constructor. You will probably never have to construct a wxHtmlTag object yourself. Feel free to ignore the constructor parameters. Have a look at src/html/htmlpars.cpp if you're interested in creating it.

**wxHtmlTag::GetAllParams**

**const wxString& GetAllParams() const**

Returns a string containing all parameters.

Example : tag contains <FONT SIZE=+2 COLOR="#000000">. Call to tag.GetAllParams() would return SIZE=+2 COLOR="#000000".

**wxHtmlTag::GetBeginPos**

**int GetBeginPos() const**

Returns beginning position of the text *between* this tag and paired ending tag. See explanation (returned position is marked with '|'):

```
bla bla bla <MYTAG> bla bla internal text</MYTAG> bla bla
                  |
```

**wxHtmlTag::GetEndPos1**

**int GetEndPos1() const**

Returns ending position of the text *between* this tag and paired ending tag. See explanation (returned position is marked with '|'):

```
bla bla bla <MYTAG> bla bla internal text</MYTAG> bla bla
                                     |
```

**wxHtmlTag::GetEndPos2**

**int GetEndPos2() const**

Returns ending position 2 of the text *between* this tag and paired ending tag. See explanation (returned position is marked with '|'):

```
bla bla bla <MYTAG> bla bla internal text</MYTAG> bla bla
                                     |
```

### **wxHtmlTag::GetName**

#### **wxString GetName() const**

Returns tag's name. The name is always in uppercase and it doesn't contain '<' or '/' characters. (So the name of <FONT SIZE=+2> tag is "FONT" and name of </table> is "TABLE")

### **wxHtmlTag::GetParam**

#### **wxString GetParam(const wxString& par, bool with\_commas = false) const**

Returns the value of the parameter. You should check whether the parameter exists or not (use *HasParam* (p. 868)) first.

#### **Parameters**

*par*

The parameter's name.

*with\_commas*

true if you want to get commas as well. See example.

#### **Example**

```
...
/* you have wxHtmlTag variable tag which is equal to
   HTML tag <FONT SIZE=+2 COLOR="#0000FF"> */
dummy = tag.GetParam("SIZE");
// dummy == "+2"
dummy = tag.GetParam("COLOR");
// dummy == "#0000FF"
dummy = tag.GetParam("COLOR", true);
// dummy == "\"#0000FF\"" -- see the difference!!
```

### **wxHtmlTag::GetParamAsColour**

#### **bool GetParamAsColour(const wxString& par, wxColour \*clr) const**

Interprets tag parameter *par* as colour specification and saves its value into wxColour variable pointed by *clr*.

Returns true on success and false if *par* is not colour specification or if the tag has no such parameter.

**wxHtmlTag::GetParamAsInt****bool GetParamAsInt(const wxString& *par*, int \**value*) const**

Interprets tag parameter *par* as an integer and saves its value into int variable pointed by *value*.

Returns true on success and false if *par* is not an integer or if the tag has no such parameter.

**wxHtmlTag::HasEnding****bool HasEnding() const**

Returns true if this tag is paired with ending tag, false otherwise.

See the example of HTML document:

```
<html><body>
Hello<p>
How are you?
<p align=center>This is centered...</p>
Oops<br>Oooops!
</body></html>
```

In this example tags HTML and BODY have ending tags, first P and BR doesn't have ending tag while the second P has. The third P tag (which is ending itself) of course doesn't have ending tag.

**wxHtmlTag::HasParam****bool HasParam(const wxString& *par*) const**

Returns true if the tag has a parameter of the given name. Example : <FONT SIZE=+2 COLOR=" #FF00FF" > has two parameters named "SIZE" and "COLOR".

**Parameters***par*

the parameter you're looking for.

**wxHtmlTag::ScanParam****wxString ScanParam(const wxString& *par*, const wxChar \**format*, void \**value*) const**

This method scans the given parameter. Usage is exactly the same as sscanf's usage except that you don't pass a string but a parameter name as the first argument and you can only retrieve one value (i.e. you can use only one "%" element in *format*).

**Parameters***par*

The name of the tag you want to query

*format*

scanf()-like format string.

*value*

pointer to a variable to store the value in

## **wxHtmlTagHandler**

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/html/htmlpars.h>

**See Also**

*Overview* (p. 2183), *wxHtmlTag* (p. 865)

### **wxHtmlTagHandler::m\_Parser**

**wxHtmlParser\* m\_Parser**

This attribute is used to access parent parser. It is protected so that it can't be accessed by user but can be accessed from derived classes.

### **wxHtmlTagHandler::wxHtmlTagHandler**

**wxHtmlTagHandler()**

Constructor.

### **wxHtmlTagHandler::GetSupportedTags**

**virtual wxString GetSupportedTags()**

Returns list of supported tags. The list is in uppercase and tags are delimited by ','.

Example : "I , B , FONT , P "

### **wxHtmlTagHandler::HandleTag**

**virtual bool HandleTag(const wxHtmlTag& tag)**

This is the core method of each handler. It is called each time one of supported tags is

detected. *tag* contains all necessary info (see *wxHtmlTag* (p. 865) for details).

### Return value

true if *ParseInner* (p. 870) was called, false otherwise.

### Example

```
bool MyHandler::HandleTag(const wxHtmlTag& tag)
{
    ...
    // change state of parser (e.g. set bold face)
    ParseInner(tag);
    ...
    // restore original state of parser
}
```

You shouldn't call *ParseInner* if the tag is not paired with an ending one.

### **wxHtmlTagHandler::ParseInner**

**void ParseInner(const wxHtmlTag& tag)**

This method calls parser's *DoParsing* (p. 860) method for the string between this tag and the paired ending tag:

```
...<A HREF="x.htm">Hello, world!</A>...
```

In this example, a call to *ParseInner* (with *tag* pointing to A tag) will parse 'Hello, world!'.

### **wxHtmlTagHandler::SetParser**

**virtual void SetParser(wxHtmlParser \*parser)**

Assigns *parser* to this handler. Each **instance** of handler is guaranteed to be called only from the parser.

## **wxHtmlTagsModule**

This class provides easy way of filling *wxHtmlWinParser*'s table of tag handlers. It is used almost exclusively together with the set of *TAGS\_MODULE\_\** macros (p. 2183)

### Derived from

*wxModule* (p. 1116)

### Include files

<wx/html/winpars.h>

### See Also

*Tag Handlers* (p. 2183), *wxHtmlTagHandler* (p. 869), *wxHtmlWinTagHandler* (p. 888),

### **wxHtmlTagsModule::FillHandlersTable**

**virtual void FillHandlersTable(wxHtmlWinParser \*parser)**

You must override this method. In most common case its body consists only of lines of the following type:

```
parser -> AddTagHandler(new MyHandler);
```

I recommend using the **TAGS\_MODULE\_\*** macros.

#### **Parameters**

*parser*

Pointer to the parser that requested tables filling.

### **wxHtmlWidgetCell**

wxHtmlWidgetCell is a class that provides a connection between HTML cells and widgets (an object derived from wxWindow). You can use it to display things like forms, input boxes etc. in an HTML window.

wxHtmlWidgetCell takes care of resizing and moving window.

#### **Derived from**

*wxHtmlCell* (p. 819)

#### **Include files**

<wx/html/htmlcell.h>

### **wxHtmlWidgetCell::wxHtmlWidgetCell**

**wxHtmlWidgetCell(wxWindow\* wnd, int w = 0)**

Constructor.

#### **Parameters**

*wnd*

Connected window. It is parent window **must** be the wxHtmlWindow object within which it is displayed!

*w*

Floating width. If non-zero width of *wnd* window is adjusted so that it is always *w*

percents of parent container's width. (For example `w = 100` means that the window will always have same width as parent container)

## wxHtmlWindow

`wxHtmlWindow` is probably the only class you will directly use unless you want to do something special (like adding new tag handlers or MIME filters).

The purpose of this class is to display HTML pages (either local file or downloaded via HTTP protocol) in a window. The width of the window is constant - given in the constructor - and virtual height is changed dynamically depending on page size. Once the window is created you can set its content by calling `SetPage(text)` (p. 880), `LoadPage(filename)` (p. 875) or `LoadFile` (p. 875).

### Note

`wxHtmlWindow` uses the `wxImage` (p. 906) class for displaying images. Don't forget to initialize all image formats you need before loading any page! (See `wxInitAllImageHandlers` (p. 1911) and `wxImage::AddHandler` (p. 910).)

### Derived from

`wxScrolledWindow` (p. 1414)

### Include files

`<wx/html/htmlwin.h>`

### Window styles

**wxHW\_SCROLLBAR\_NEVER**      Never display scrollbars, not even when the page is larger than the window.

**wxHW\_SCROLLBAR\_AUTO**      Display scrollbars only if page's size exceeds window's size.

**wxHW\_NO\_SELECTION**      Don't allow the user to select text.

### Event handling

To process input from a `wxHtmlWindow`, use these event handler macros to direct input to member functions that take a `wxHtmlCellEvent` (p. 882) argument or a `wxHtmlLinkEvent` (p. 881).

**EVT\_HTML\_CELL\_CLICKED(id, func)** A `wxHtmlCell` (p. 819) was clicked.

**EVT\_HTML\_CELL\_HOVER(id, func)** The mouse passed over a `wxHtmlCell` (p. 819).

**EVT\_HTML\_LINK\_CLICKED(id, func)** A `wxHtmlCell` (p. 819) which contains an hyperlink was clicked.

### See also

`wxHtmlLinkEvent` (p. 881), `wxHtmlCellEvent` (p. 882)



**wxHtmlWindow::wxHtmlWindow****wxHtmlWindow()**

Default constructor.

**wxHtmlWindow**(**wxWindow** \*parent, **wxWindowID** id = -1, **const wxPoint&** pos = wxDefaultPosition, **const wxSize&** size = wxDefaultSize, **long** style = wxHW\_DEFAULT\_STYLE, **const wxString&** name = "htmlWindow")

Constructor. The parameters are the same as for the *wxScrolledWindow* (p. 1414) constructor.

**Parameters**

*style*

Window style. See *wxHtmlWindow* (p. 872).

**wxHtmlWindow::AddFilter**

**static void AddFilter**(**wxHtmlFilter** \*filter)

Adds *input filter* (p. 2182) to the static list of available filters. These filters are present by default:

- text/html MIME type
- image/\* MIME types
- Plain Text filter (this filter is used if no other filter matches)

**wxHtmlWindow::AppendToPage**

**bool AppendToPage**(**const wxString&** source)

Appends HTML fragment to currently displayed text and refreshes the window.

**Parameters**

*source*

HTML code fragment

**Return value**

false if an error occurred, true otherwise.

**wxHtmlWindow::GetInternalRepresentation**

**wxHtmlContainerCell\*** GetInternalRepresentation() **const**

Returns pointer to the top-level container.

See also: *Cells Overview* (p. 2182), *Printing Overview* (p. 2180)

### **wxHtmlWindow::GetOpenedAnchor**

#### **wxString GetOpenedAnchor()**

Returns anchor within currently opened page (see *GetOpenedPage* (p. 874)). If no page is opened or if the displayed page wasn't produced by call to *LoadPage*, empty string is returned.

### **wxHtmlWindow::GetOpenedPage**

#### **wxString GetOpenedPage()**

Returns full location of the opened page. If no page is opened or if the displayed page wasn't produced by call to *LoadPage*, empty string is returned.

### **wxHtmlWindow::GetOpenedPageTitle**

#### **wxString GetOpenedPageTitle()**

Returns title of the opened page or *wxEmptyString* if current page does not contain `<TITLE>` tag.

### **wxHtmlWindow::GetRelatedFrame**

#### **wxFrame\* GetRelatedFrame() const**

Returns the related frame.

### **wxHtmlWindow::HistoryBack**

#### **bool HistoryBack()**

Moves back to the previous page. (each page displayed using *LoadPage* (p. 875) is stored in history list.)

### **wxHtmlWindow::HistoryCanBack**

#### **bool HistoryCanBack()**

Returns true if it is possible to go back in the history (i.e. *HistoryBack()* won't fail).

### **wxHtmlWindow::HistoryCanForward**

#### **bool HistoryCanForward()**

Returns true if it is possible to go forward in the history (i.e. *HistoryBack()* won't fail).

### **wxHtmlWindow::HistoryClear**

**void HistoryClear()**

Clears history.

**wxHtmlWindow::HistoryForward****bool HistoryForward()**

Moves to next page in history.

**wxHtmlWindow::LoadFile****virtual bool LoadFile(const wxString& filename)**

Loads HTML page from file and displays it.

**Return value**

false if an error occurred, true otherwise

**See also**

*LoadPage* (p. 875)

**wxHtmlWindow::LoadPage****virtual bool LoadPage(const wxString& location)**

Unlike *SetPage* this function first loads HTML page from *location* and then displays it. See example:

```
htmlwin->LoadPage("help/myproject/index.htm");
```

**Parameters**

*location*

The address of document. See *wxFileSystem* (p. 633) for details on address format and behaviour of "opener".

**Return value**

false if an error occurred, true otherwise

**See also**

*LoadFile* (p. 875)

**wxHtmlWindow::OnCellClicked****virtual bool OnCellClicked(wxHtmlCell \*cell, wxCoord x, wxCoord y, const wxMouseEvent& event)**

This method is called when a mouse button is clicked inside `wxHtmlWindow`.

The default behaviour is to emit a `wxHtmlCellEvent` (p. 882) and, if the event was not processed or skipped, call `OnLinkClicked` (p. 876) if the cell contains an hypertext link.

Overloading this method is deprecated; intercept the event instead.

#### Parameters

*cell*

The cell inside which the mouse was clicked, always a simple (i.e. non-container) cell

*x, y*

The logical coordinates of the click point

*event*

The mouse event containing other information about the click

#### Return value

`true` if a link was clicked, `false` otherwise.

#### `wxHtmlWindow::OnCellMouseHover`

**virtual void OnCellMouseHover(`wxHtmlCell *cell`, `wxCoord x`, `wxCoord y`)**

This method is called when a mouse moves over an HTML cell. Default behaviour is to emit a `wxHtmlCellEvent` (p. 882). Overloading this method is deprecated; intercept the event instead.

#### Parameters

*cell*

The cell inside which the mouse is currently, always a simple (i.e. non-container) cell

*x, y*

The logical coordinates of the click point

#### `wxHtmlWindow::OnLinkClicked`

**virtual void OnLinkClicked(const `wxHtmlLinkInfo& link`)**

Called when user clicks on hypertext link. Default behaviour is to emit a `wxHtmlLinkEvent` (p. 881) and, if the event was not processed or skipped, call `LoadPage` (p. 875) and do nothing else. Overloading this method is deprecated; intercept the event instead.

Also see `wxHtmlLinkInfo` (p. 851).

**wxHtmlWindow::OnOpeningURL**

**virtual wxHtmlOpeningStatus OnOpeningURL(wxHtmlURLType type, const wxString& url, wxString \*redirect)**

Called when an URL is being opened (either when the user clicks on a link or an image is loaded). The URL will be opened only if OnOpeningURL returns `wxHTML_OPEN`. This method is called by `wxHtmlParser::OpenURL` (p. 861). You can override OnOpeningURL to selectively block some URLs (e.g. for security reasons) or to redirect them elsewhere. Default behaviour is to always return `wxHTML_OPEN`.

**Parameters**

*type*

Indicates type of the resource. Is one of	<b>wxHTML_URL_PAGE</b>	Opening a HTML page.
	<b>wxHTML_URL_IMAGE</b>	Opening an image.
	<b>wxHTML_URL_OTHER</b>	Opening a resource that doesn't fall into any other category.

*url*

URL being opened.

*redirect*

Pointer to wxString variable that must be filled with an URL if OnOpeningURL returns `wxHTML_REDIRECT`.

<b>Return value</b>	<b>wxHTML_OPEN</b>	Open the URL.
	<b>wxHTML_BLOCK</b>	Deny access to the URL, <code>wxHtmlParser::OpenURL</code> (p. 861) will return NULL.
	<b>wxHTML_REDIRECT</b>	Don't open <i>url</i> , redirect to another URL. OnOpeningURL must fill <i>*redirect</i> with the new URL. OnOpeningURL will be called again on returned URL.

**wxHtmlWindow::OnSetTitle**

**virtual void OnSetTitle(const wxString& title)**

Called on parsing `<TITLE>` tag.

**wxHtmlWindow::ReadCustomization**

**virtual void ReadCustomization(wxConfigBase \*cfg, wxString path = wxEmptyString)**

This reads custom settings from wxConfig. It uses the path 'path' if given, otherwise it saves info into currently selected path. The values are stored in sub-path `wxHtmlWindow`

Read values: all things set by `SetFont`s, `SetBorders`.

### Parameters

*cfg*

wxConfig from which you want to read the configuration.

*path*

Optional path in config tree. If not given current path is used.

### **wxHtmlWindow::SelectAll**

**void SelectAll()**

Selects all text in the window.

### See also

*SelectLine* (p. 878), *SelectWord* (p. 878)

### **wxHtmlWindow::SelectionToText**

**wxString SelectionToText()**

Returns current selection as plain text. Returns empty string if no text is currently selected.

### **wxHtmlWindow::SelectLine**

**void SelectLine(const wxPoint& pos)**

Selects the line of text that *pos* points at. Note that *pos* is relative to the top of displayed page, not to window's origin, use *CalcUnscrolledPosition* (p. 1417) to convert physical coordinate.

### See also

*SelectAll* (p. 878), *SelectWord* (p. 878)

### **wxHtmlWindow::SelectWord**

**void SelectWord(const wxPoint& pos)**

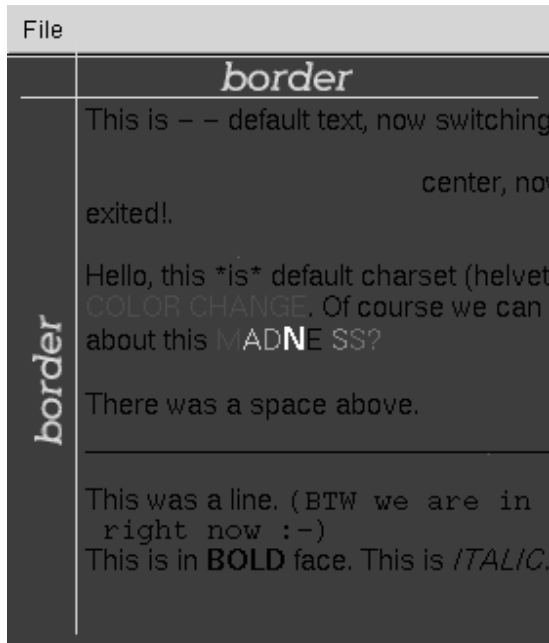
Selects the word at position *pos*. Note that *pos* is relative to the top of displayed page, not to window's origin, use *CalcUnscrolledPosition* (p. 1417) to convert physical coordinate.

### See also

*SelectAll* (p. 878), *SelectLine* (p. 878)

**wxHtmlWindow::SetBorders****void SetBorders(int *b*)**

This function sets the space between border of window and HTML contents. See image:

**Parameters***b*

indentation from borders in pixels

**wxHtmlWindow::SetFont**
**void SetFont(const wxString& *normal\_face*, const wxString& *fixed\_face*, const int  
 \**sizes* = NULL)**

This function sets font sizes and faces.

**Parameters***normal\_face*

This is face name for normal (i.e. non-fixed) font. It can be either empty string (then the default face is chosen) or platform-specific face name. Examples are "helvetica" under Unix or "Times New Roman" under Windows.

*fixed\_face*

The same thing for fixed face ( <TT>..<</TT> )

*sizes*

This is an array of 7 items of *int* type. The values represent size of font with HTML size from -2 to +4 ( <FONT SIZE=-2> to <FONT SIZE=+4> ). Default sizes are used if sizes is NULL.

### Defaults

Default font sizes are defined by constants `wxHTML_FONT_SIZE_1`, `wxHTML_FONT_SIZE_2`, ..., `wxHTML_FONT_SIZE_7`. Note that they differ among platforms. Default face names are empty strings.

### **wxHtmlWindow::SetPage**

**bool SetPage(const wxString& source)**

Sets HTML page and display it. This won't **load** the page!! It will display the *source*. See example:

```
htmlwin -> SetPage( "<html><body>Hello, world!</body></html>" );
```

If you want to load a document from some location use *LoadPage* (p. 875) instead.

### Parameters

*source*

The HTML document source to be displayed.

### Return value

false if an error occurred, true otherwise.

### **wxHtmlWindow::SetRelatedFrame**

**void SetRelatedFrame(wxFrame\* frame, const wxString& format)**

Sets the frame in which page title will be displayed. *format* is format of frame title, e.g. "HtmlHelp : %s". It must contain exactly one %s. This %s is substituted with HTML page title.

### **wxHtmlWindow::SetRelatedStatusBar**

**void SetRelatedStatusBar(int bar)**

**After** calling *SetRelatedFrame* (p. 880), this sets statusbar slot where messages will be displayed. (Default is -1 = no messages.)

### Parameters

*bar*

statusbar slot number (0..n)



**wxHtmlWindow::ToText****wxString ToText()**

Returns content of currently displayed page as plain text.

**wxHtmlWindow::WriteCustomization**

**virtual void WriteCustomization(wxConfigBase \*cfg, wxString path = wxEmptyString)**

Saves custom settings into wxConfig. It uses the path 'path' if given, otherwise it saves info into currently selected path. Regardless of whether the path is given or not, the function creates sub-path `wxHtmlWindow`.

Saved values: all things set by `SetFont`s, `SetBorder`s.

**Parameters**

*cfg*

wxConfig to which you want to save the configuration.

*path*

Optional path in config tree. If not given, the current path is used.

**wxHtmlLinkEvent**

This event class is used for the events generated by *wxHtmlWindow* (p. 872).

**Derived from**

*wxCommandEvent* (p. 250)

*wxEvent* (p. 572)

*wxObject* (p. 1148)

**Include files**

<wx/html/htmlwin.h>

**Event handling**

To process input from a *wxHtmlLinkEvent*, use one of these event handler macros to direct input to member function that take a *wxHtmlLinkEvent* (p. 881) argument:

**EVT\_HTML\_LINK\_CLICKED(id, func)** User clicked on an hyperlink.

**wxHtmlLinkEvent::wxHtmlLinkEvent**

**wxHyperlinkEvent(int id, const wxHtmlLinkInfo & linkinfo)**

The constructor is not normally used by the user code.

### **wxHtmlLinkEvent::GetLinkInfo**

**const wxHtmlLinkInfo& GetLinkInfo() const**

Returns the *wxHtmlLinkInfo* (p. 851) which contains info about the cell clicked and the hyperlink it contains.

## **wxHtmlCellEvent**

This event class is used for the events generated by *wxHtmlWindow* (p. 872).

### **Derived from**

*wxCommandEvent* (p. 250)

*wxEvent* (p. 572)

*wxObject* (p. 1148)

### **Include files**

<wx/html/htmlwin.h>

### **Event handling**

To process input from a *wxHtmlCellEvent*, use one of these event handler macros to direct input to member function that take a *wxHtmlCellEvent* (p. 882) argument:

**EVT\_HTML\_CELL\_HOVER(id, func)** User moved the mouse over a *wxHtmlCell* (p. 819).

**EVT\_HTML\_CELL\_CLICKED(id, func)** User clicked on a *wxHtmlCell* (p. 819). When handling this event, remember to use *wxHtmlCell::SetLinkClicked(true)* (p. 883) if the cell contains a link.

### **wxHtmlCellEvent::wxHtmlCellEvent**

**wxHtmlCellEvent(wxEventType commandType, int id, wxHtmlCell \* cell, const wxPoint & point)**

The constructor is not normally used by the user code.

### **wxHtmlCellEvent::GetCell**

**wxHtmlCell \* GetCell() const**

Returns the *wxHtmlCellEvent* (p. 882) associated with the event.

**wxHtmlCellEvent::GetPoint****wxPoint GetPoint() const**

Returns the *wxPoint* (p. 1193) associated with the event.

**wxHtmlCellEvent::SetLinkClicked****bool SetLinkClicked(bool linkclicked)**

Call this function with *linkclicked* set to *true* if the cell which has been clicked contained a link or *false* otherwise (which is the default). With this function the event handler can return info to the *wxHtmlWindow* which sent the event.

**wxHtmlCellEvent::GetLinkClicked****bool GetLinkClicked() const**

Returns *true* if *SetLinkClicked(true)* (p. 883) has previously been called; *false* otherwise.

**wxHtmlWinParser**

This class is derived from *wxHtmlParser* (p. 858) and its main goal is to parse HTML input so that it can be displayed in *wxHtmlWindow* (p. 872). It uses a special *wxHtmlWinTagHandler* (p. 888).

**Notes**

The product of parsing is a *wxHtmlCell* (resp. *wxHtmlContainer*) object.

**Derived from**

*wxHtmlParser* (p. 858)

**Include files**

<wx/html/winpars.h>

**See Also**

*Handlers overview* (p. 2183)

**wxHtmlWinParser::wxHtmlWinParser****wxHtmlWinParser()****wxHtmlWinParser(wxHtmlWindowInterface \*wndiface)**

Constructor. Don't use the default one, use constructor with *wndIface* parameter (*wndIface* is a pointer to interface object for the associated *wxHtmlWindow* (p. 872) or other HTML rendering window such as *wxHtmlListBox* (p. 853)).

### **wxHtmlWinParser::AddModule**

**static void AddModule(wxHtmlTagsModule \*module)**

Adds *module* (p. 2183) to the list of wxHtmlWinParser tag handler.

### **wxHtmlWinParser::CloseContainer**

**wxHtmlContainerCell\* CloseContainer()**

Closes the container, sets actual container to the parent one and returns pointer to it (see *Overview* (p. 2182)).

### **wxHtmlWinParser::CreateCurrentFont**

**virtual wxFont\* CreateCurrentFont()**

Creates font based on current setting (see *SetFontSize* (p. 888), *SetFontBold* (p. 887), *SetFontItalic* (p. 888), *SetFontFixed* (p. 887), *SetFontUnderlined* (p. 888)) and returns pointer to it. If the font was already created only a pointer is returned.

### **wxHtmlWinParser::GetActualColor**

**const wxColour& GetActualColor() const**

Returns actual text colour.

### **wxHtmlWinParser::GetAlign**

**int GetAlign() const**

Returns default horizontal alignment.

### **wxHtmlWinParser::GetCharHeight**

**int GetCharHeight() const**

Returns (average) char height in standard font. It is used as DC-independent metrics.

**Note:** This function doesn't return the *actual* height. If you want to know the height of the current font, call `GetDC -> GetCharHeight()`.

### **wxHtmlWinParser::GetCharWidth**

**int GetCharWidth() const**

Returns average char width in standard font. It is used as DC-independent metrics.

**Note:** This function doesn't return the *actual* width. If you want to know the height of the current font, call `GetDC -> GetCharWidth()`

### **wxHtmlWinParser::GetContainer**

**wxHtmlContainerCell\* GetContainer() const**

Returns pointer to the currently opened container (see *Overview* (p. 2182)). Common use:

```
m_WParser -> GetContainer() -> InsertCell(new ...);
```

### **wxHtmlWinParser::GetDC**

**wxDC\* GetDC()**

Returns pointer to the DC used during parsing.

### **wxHtmlWinParser::GetEncodingConverter**

**wxEncodingConverter \* GetEncodingConverter() const**

Returns *wxEncodingConverter* (p. 568) class used to do conversion between *input encoding* (p. 886) and *output encoding* (p. 886).

### **wxHtmlWinParser::GetFontBold**

**int GetFontBold() const**

Returns true if actual font is bold, false otherwise.

### **wxHtmlWinParser::GetFontFace**

**wxString GetFontFace() const**

Returns actual font face name.

### **wxHtmlWinParser::GetFontFixed**

**int GetFontFixed() const**

Returns true if actual font is fixed face, false otherwise.

### **wxHtmlWinParser::GetFontItalic**

**int GetFontItalic() const**

Returns true if actual font is italic, false otherwise.

**wxHtmlWinParser::GetFontSize****int GetFontSize() const**

Returns actual font size (HTML size varies from -2 to +4)

**wxHtmlWinParser::GetFontUnderlined****int GetFontUnderlined() const**

Returns true if actual font is underlined, false otherwise.

**wxHtmlWinParser::GetInputEncoding****wxFontEncoding GetInputEncoding() const**

Returns input encoding.

**wxHtmlWinParser::GetLink****const wxHtmlLinkInfo& GetLink() const**

Returns actual hypertext link. (This value has a non-empty *Href* (p. 852) string if the parser is between `<A>` and `</A>` tags, `wxEmptyString` otherwise.)

**wxHtmlWinParser::GetLinkColor****const wxColour& GetLinkColor() const**

Returns the colour of hypertext link text.

**wxHtmlWinParser::GetOutputEncoding****wxFontEncoding GetOutputEncoding() const**

Returns output encoding, i.e. closest match to document's input encoding that is supported by operating system.

**wxHtmlWinParser::GetWindow****wxHtmlWindow\* GetWindow()**

Returns associated window (`wxHtmlWindow`). This may be `NULL`! (You should always test if it is non-`NULL`. For example `TITLE` handler sets window title only if some window is associated, otherwise it does nothing)

**wxHtmlWinParser::OpenContainer****wxHtmlContainerCell\* OpenContainer()**

Opens new container and returns pointer to it (see *Overview* (p. 2182)).

### **wxHtmlWinParser::SetActualColor**

**void SetActualColor(const wxColour& clr)**

Sets actual text colour. Note: this DOESN'T change the colour! You must create *wxHtmlColourCell* (p. 825) yourself.

### **wxHtmlWinParser::SetAlign**

**void SetAlign(int a)**

Sets default horizontal alignment (see *wxHtmlContainerCell::SetAlignHor* (p. 827).) Alignment of newly opened container is set to this value.

### **wxHtmlWinParser::SetContainer**

**wxHtmlContainerCell\* SetContainer(wxHtmlContainerCell \*c)**

Allows you to directly set opened container. This is not recommended - you should use *OpenContainer* wherever possible.

### **wxHtmlWinParser::SetDC**

**virtual void SetDC(wxDC \*dc, double pixel\_scale = 1.0)**

Sets the DC. This must be called before *Parse* (p. 861)! *pixel\_scale* can be used when rendering to high-resolution DCs (e.g. printer) to adjust size of pixel metrics. (Many dimensions in HTML are given in pixels -- e.g. image sizes. 300x300 image would be only one inch wide on typical printer. With *pixel\_scale* = 3.0 it would be 3 inches.)

### **wxHtmlWinParser::SetFontBold**

**void SetFontBold(int x)**

Sets bold flag of actualfont. *x* is either true or false.

### **wxHtmlWinParser::SetFontFace**

**void SetFontFace(const wxString& face)**

Sets current font face to *face*. This affects either fixed size font or proportional, depending on context (whether the parser is inside `<TT>` tag or not).

### **wxHtmlWinParser::SetFontFixed**

**void SetFontFixed(int x)**

Sets fixed face flag of actualfont. *x* is either true or false.

**wxHtmlWinParser::SetFontItalic****void SetFontItalic(int x)**

Sets italic flag of actualfont. x is either true or false.

**wxHtmlWinParser::SetFontSize****void SetFontSize(int s)**

Sets actual font size (HTML size varies from 1 to 7)

**wxHtmlWinParser::SetFontUnderlined****void SetFontUnderlined(int x)**

Sets underlined flag of actualfont. x is either true or false.

**wxHtmlWinParser::SetFonts****void SetFonts(const wxString& normal\_face, const wxString& fixed\_face, const int \*sizes = NULL)**

Sets fonts. See *wxHtmlWindow::SetFonts* (p. 879) for detailed description.

**wxHtmlWinParser::SetInputEncoding****void SetInputEncoding(wxFontEncoding enc)**

Sets input encoding. The parser uses this information to build conversion tables from document's encoding to some encoding supported by operating system.

**wxHtmlWinParser::SetLink****void SetLink(const wxHtmlLinkInfo& link)**

Sets actual hypertext link. Empty link is represented by *wxHtmlLinkInfo* (p. 851) with *Href* equal to *wxEmptyString*.

**wxHtmlWinParser::SetLinkColor****void SetLinkColor(const wxColour& clr)**

Sets colour of hypertext link.

**wxHtmlWinTagHandler**

This is basically *wxHtmlTagHandler* except that it is extended with protected member *m\_WParser* pointing to the *wxHtmlWinParser* object (value of this member is identical to



wxHtmlParser's m\_Parser).

**Derived from**

*wxHtmlTagHandler* (p. 869)

**Include files**

<wx/html/winpars.h>

**wxHtmlWinTagHandler::m\_WParser****wxHtmlWinParser\* m\_WParser**

Value of this attribute is identical to value of m\_Parser. The only different is that m\_WParser points to wxHtmlWinParser object while m\_Parser points to wxHtmlParser object. (The same object, but overcast.)

**wxHTTP****Derived from**

*wxProtocol* (p. 1235)

**Include files**

<wx/protocol/http.h>

**See also**

*wxSocketBase* (p. 1472), *wxURL* (p. 1763)

**wxHTTP::GetResponse****int GetResponse() const**

Returns the HTTP response code returned by the server. Please refer to RFC 2616 (<http://www.faqs.org/rfcs/rfc2616.html>) for the list of responses.

**wxHTTP::GetInputStream****wxInputStream \* GetInputStream(const wxString& path)**

Creates a new input stream on the specified path. Notice that this stream is unseekable, i.e. SeekI() and TellI() methods shouldn't be used.

Note that you can still know the size of the file you are getting using *wxStreamBase::GetSize()* (p. 1545). However there is a limitation: in HTTP protocol, the size is not always specified so sometimes `(size_t)-1` can be returned to indicate that the size is unknown. In such case, you may want to use *wxInputStream::LastRead()* (p. 942)

method in a loop to get the total size.

### **Return value**

Returns the initialized stream. You must delete it yourself once you don't use it anymore and this must be done before the `wxHTTP` object itself is destroyed. The destructor closes the network connection. The next time you will try to get a file the network connection will have to be reestablished, but you don't have to take care of this since `wxHTTP` reestablishes it automatically.

### **See also**

`wxInputStream` (p. 941)

### **wxHTTP::SetHeader**

**void SetHeader(const wxString& header, const wxString& h\_data)**

It sets data of a field to be sent during the next request to the HTTP server. The field name is specified by *header* and the content by *h\_data*. This is a low level function and it assumes that you know what you are doing.

### **wxHTTP::GetHeader**

**wxString GetHeader(const wxString& header)**

Returns the data attached with a field whose name is specified by *header*. If the field doesn't exist, it will return an empty string and not a NULL string.

### **Note**

The header is not case-sensitive, i.e. "CONTENT-TYPE" and "content-type" represent the same header.

## **wxHyperlinkCtrl**

This class shows a static text element which links to an URL. Appearance and behaviour is completely customizable. In fact, when the user clicks on the hyperlink, a `wxHyperlinkEvent` (p. 894) is sent but if that event is not handled (or it's skipped; see `wxEvent::Skip` (p. 575)), then a call to `wxLaunchDefaultBrowser` (p. 1959) is done with the hyperlink's URL.

Note that standard `wxWindow` (p. 1795) functions like `SetBackgroundColour` (p. 1835), `SetFont` (p. 1840), `SetCursor` (p. 1837), `SetLabel` (p. 1841) can be used to customize appearance of the hyperlink.

### **Derived from**

`wxControl` (p. 285)

`wxWindow` (p. 1795)

`wxEvtHandler` (p. 576)

*wxObject* (p. 1148)

### Include files

<wx/hyperlink.h>

### Window styles

<b>wxHL_ALIGN_LEFT</b>	Align the text to the left.
<b>wxHL_ALIGN_RIGHT</b>	Align the text to the right.
<b>wxHL_ALIGN_CENTRE</b>	Center the text (horizontally).
<b>wxHL_CONTEXTMENU</b>	Pop up a context menu when the hyperlink is right-clicked. The context menu contains a "Copy URL" menu item which is automatically handled by the hyperlink and which just copies in the clipboard the URL (not the label) of the control.
<b>wxHL_DEFAULT_STYLE</b>	The default style for <code>wxHyperlinkCtrl</code> : <code>wxNO_BORDER   wxHL_CONTEXTMENU   wxHL_ALIGN_CENTRE</code> .

See also *window styles overview* (p. 2089).

### Event handling

To process input from an hyperlink control, use these event handler macros to direct input to member functions that take a *awxHyperlinkEvent* (p. 894) argument.

<b>EVT_HYPERLINK(id, func)</b>	The hyperlink was (left) clicked. If this event is not handled in user's code (or it's skipped; see <i>wxEvent::Skip</i> (p. 575)), then a call to <i>wxLaunchDefaultBrowser</i> (p. 1959) is done with the hyperlink's URL.
--------------------------------	--

### See also

*wxURL* (p. 1763), *wxHyperlinkEvent* (p. 894)

### **wxHyperlinkCtrl::wxHyperLink**

**wxHyperLink**(*wxWindow\* parent*, *wxWindowID id*, **const wxString & label**, **const wxString & url**, **const wxPoint& pos** = *wxDefaultPosition*, **const wxSize& size** = *wxDefaultSize*, **long style**, **const wxString& name** = "hyperlink")

Constructor. See *Create* (p. 891) for more info.

### **wxHyperlinkCtrl::Create**

```
bool Create(wxWindow* parent, wxWindowID id, const wxString & label, const  
wxString & url, const wxPoint& pos = wxDefaultPosition, const wxSize& size =  
wxDefaultSize, long style, const wxString& name = "hyperlink")
```

Creates the hyperlink control.

### Parameters

*parent*

Parent window. Must not be `NULL`.

*id*

Window identifier. A value of `wxID_ANY` indicates a default value.

*label*

The label of the hyperlink.

*url*

The URL associated with the given label.

*pos*

Window position.

*size*

Window size. If the `wxDefaultSize` is specified then the window is sized appropriately.

*style*

Window style. See *wxHyperlinkCtrl* (p. 890).

*validator*

Window validator.

*name*

Window name.

### **wxHyperlinkCtrl::GetHoverColour**

```
wxColour GetHoverColour() const
```

Returns the colour used to print the label of the hyperlink when the mouse is over the control.

### **wxHyperlinkCtrl::SetHoverColour**

**void SetHoverColour(const wxColour & colour)**

Sets the colour used to print the label of the hyperlink when the mouse is over the control.

**wxHyperlinkCtrl::GetNormalColour**

**wxColour GetNormalColour() const**

Returns the colour used to print the label when the link has never been clicked before (i.e. the link has not been *visited*) and the mouse is not over the control.

**wxHyperlinkCtrl::SetNormalColour**

**void SetNormalColour(const wxColour & colour)**

Sets the colour used to print the label when the link has never been clicked before (i.e. the link has not been *visited*) and the mouse is not over the control.

**wxHyperlinkCtrl::GetVisitedColour**

**wxColour GetVisitedColour() const**

Returns the colour used to print the label when the mouse is not over the control and the link has already been clicked before (i.e. the link has been *visited*).

**wxHyperlinkCtrl::SetVisitedColour**

**void SetVisitedColour(const wxColour & colour)**

Sets the colour used to print the label when the mouse is not over the control and the link has already been clicked before (i.e. the link has been *visited*).

**wxHyperlinkCtrl::GetVisited**

**bool GetVisited() const**

Returns `true` if the hyperlink has already been clicked by the user at least one time.

**wxHyperlinkCtrl::SetVisited**

**void SetVisited(bool visited = true)**

Marks the hyperlink as visited (see *SetVisitedColour* (p. 893)).

**wxHyperlinkCtrl::GetURL**

**wxString GetURL() const**

Returns the URL associated with the hyperlink.

**wxHyperlinkCtrl::SetURL****void SetURL(const wxString & url)**

Sets the URL associated with the hyperlink.

**wxHyperlinkEvent**

This event class is used for the events generated by *wxHyperlinkCtrl* (p. 890).

**Derived from***wxCommandEvent* (p. 250)*wxEvent* (p. 572)*wxObject* (p. 1148)**Include files**`<wx/hyperlink.h>`**Event handling**

To process input from a *wxHyperlinkCtrl*, use one of these event handler macros to direct input to member function that take a *wxHyperlinkEvent* (p. 894) argument:

**EVT\_HYPERLINK(id, func)**                      User clicked on an hyperlink.**wxHyperlinkEvent::wxHyperlinkEvent****wxHyperlinkEvent(wxObject \* generator, int id, const wxString & url)**

The constructor is not normally used by the user code.

**wxHyperlinkEvent::GetURL****wxString GetURL() const**

Returns the URL of the hyperlink where the user has just clicked.

**wxHyperlinkEvent::SetURL****void SetURL(const wxString & url)**

Sets the URL associated with the event.

**wxIcon**

An icon is a small rectangular bitmap usually used for denoting a minimized application. It

differs from a `wxBitmap` in always having a mask associated with it for transparent drawing. On some platforms, icons and bitmaps are implemented identically, since there is no real distinction between a `wxBitmap` with a mask and an icon; and there is no specific icon format on some platforms (X-based applications usually standardize on XPMs for small bitmaps and icons). However, some platforms (such as Windows) make the distinction, so a separate class is provided.

**Derived from**

`wxBitmap` (p. 123)  
`wxGDIObject` (p. 709)  
`wxObject` (p. 1148)

**Include files**

<wx/icon.h>

**Predefined objects**

Objects:

**wxNullIcon****Remarks**

It is usually desirable to associate a pertinent icon with a frame. Icons can also be used for other purposes, for example with `wxTreeCtrl` (p. 1728) and `wxListCtrl` (p. 980).

Icons have different formats on different platforms. Therefore, separate icons will usually be created for the different environments. Platform-specific methods for creating a **wxIcon** structure are catered for, and this is an occasion where conditional compilation will probably be required.

Note that a new icon must be created for every time the icon is to be used for a new window. In Windows, the icon will not be reloaded if it has already been used. An icon allocated to a frame will be deleted when the frame is deleted.

For more information please see *Bitmap and icon overview* (p. 2117).

**See also**

*Bitmap and icon overview* (p. 2117), *supported bitmap file formats* (p. 2118), `wxDC::DrawIcon` (p. 461), `wxCursor` (p. 297)

**wxIcon::wxIcon****wxIcon()**

Default constructor.

**wxIcon(const wxIcon& icon)**

Copy constructor.

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

Creates an icon from the given data, which can be of arbitrary type.

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

Creates an icon from an array of bits.

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

Creates a new icon.

**wxIcon**(char\*\* *bits*)

**wxIcon**(const char\*\* *bits*)

Creates an icon from XPM data.

**wxIcon**(const wxString& *name*, wxBitmapType *type*, int *desiredWidth* = -1, int  
*desiredHeight* = -1)

Loads an icon from a file or resource.

**wxIcon**(const wxIconLocation& *loc*)

Loads an icon from the specified *location* (p. 902).

### Parameters

*bits*

Specifies an array of pixel values.

*width*

Specifies the width of the icon.

*height*

Specifies the height of the icon.

*desiredWidth*

Specifies the desired width of the icon. This parameter only has an effect in Windows (32-bit) where icon resources can contain several icons of different sizes.

*desiredHeight*

Specifies the desired height of the icon. This parameter only has an effect in Windows (32-bit) where icon resources can contain several icons of different sizes.

*depth*



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

*name*

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

*loc*

The object describing the location of the native icon, see *wxIconLocation* (p. 902).

*type*

May be one of the following:

`wxBITMAP_TYPE_ICO`      Load a Windows icon file.

`wxBITMAP_TYPE_ICO_RESOURCE` Load a Windows icon from the resource database.

`wxBITMAP_TYPE_GIF`      Load a GIF bitmap file.

`wxBITMAP_TYPE_XBM`      Load an X bitmap file.

`wxBITMAP_TYPE_XPM`      Load an XPM bitmap file.

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

## Remarks

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

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

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

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

The sixth form constructs a new icon.

The seventh form constructs an icon from pixmap (XPM) data, if `wxWidgets` has been configured to incorporate this feature.

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

```
#include "mybitmap.xpm"

...

wxIcon *icon = new wxIcon(mybitmap);
```

A macro, `wxICON`, is available which creates an icon using an XPM on the appropriate platform, or an icon resource on Windows.

```
wxIcon icon(wxICON(mondrian));

// Equivalent to:

#ifdef __WXGTK__ || defined(__WXMOTIF__)
wxIcon icon(mondrian_xpm);
#endif

#ifdef __WXMSW__
wxIcon icon("mondrian");
#endif
```

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

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

### See also

### **wxIcon::CopyFromBitmap**

**void CopyFromBitmap(const wxBitmap& bmp)**

Copies *bmp* bitmap to this icon. Under MS Windows the bitmap must have mask colour set.

*wxIcon::LoadFile* (p. 899)

**wxPerl note:** Constructors supported by wxPerl are:

- `!Icon->new( width, height, depth = -1 )`
- `!Icon->new( name, type, desiredWidth = -1, desiredHeight = -1 )`
- `!Icon->newFromBits( bits, width, height, depth = 1 )`
- `!Icon->newFromXPM( data )`

### **wxIcon::~wxIcon**

**~wxIcon()**

Destructor. See *reference-counted object destruction* (p. 2046) for more info.

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

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

### **`wxIcon::GetDepth`**

**`int GetDepth() const`**

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

### **`wxIcon::GetHeight`**

**`int GetHeight() const`**

Gets the height of the icon in pixels.

### **`wxIcon::GetWidth`**

**`int GetWidth() const`**

Gets the width of the icon in pixels.

### **See also**

*`wxIcon::GetHeight`* (p. 899)

### **`wxIcon::LoadFile`**

**`bool LoadFile(const wxString& name, wxBitmapType type)`**

Loads an icon from a file or resource.

### **Parameters**

*name*

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

*type*

One of the following values:

**`wxBITMAP_TYPE_ICO`**      Load a Windows icon file.

**`wxBITMAP_TYPE_ICO_RESOURCE`** Load a Windows icon from the resource database.

**`wxBITMAP_TYPE_GIF`**      Load a GIF bitmap file.

**`wxBITMAP_TYPE_XBM`**      Load an X bitmap file.

**wxBITMAP\_TYPE\_XPM**     Load an XPM bitmap file.

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

#### **Return value**

true if the operation succeeded, false otherwise.

#### **See also**

*wxIcon::wxIcon* (p. 895)

#### **wxIcon::IsOk**

**bool IsOk() const**

Returns true if icon data is present.

#### **wxIcon::SetDepth**

**void SetDepth(int *depth*)**

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

#### **Parameters**

*depth*

Icon depth.

#### **wxIcon::SetHeight**

**void SetHeight(int *height*)**

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

#### **Parameters**

*height*

Icon height in pixels.

#### **wxIcon::SetWidth**

**void SetWidth(int *width*)**

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

#### **Parameters**

*width*

Icon width in pixels.

**wxIcon::operator =****wxIcon& operator =(const wxIcon& icon)**

Assignment operator, using *reference counting* (p. 2046).

**Parameters***icon*

Icon to assign.

**Return value**

Returns 'this' object.

**wxIconBundle**

This class contains multiple copies of an icon in different sizes, see also *wxDialog::SetIcons* (p. 503) and *wxTopLevelWindow::SetIcons* (p. 1717).

**Derived from**

No base class

**wxIconBundle::wxIconBundle****wxIconBundle()**

Default constructor.

**wxIconBundle(const wxString& file, long type)**

Initializes the bundle with the icon(s) found in the file.

**wxIconBundle(const wxIcon& icon)**

Initializes the bundle with a single icon.

**wxIconBundle(const wxIconBundle& ic)**

Copy constructor.

**wxIconBundle::~~wxIconBundle****~wxIconBundle()**

Destructor.

**wxIconBundle::AddIcon**

**void AddIcon(const wxString& file, long type)**

Adds all the icons contained in the file to the bundle; if the collection already contains icons with the same width and height, they are replaced by the new ones.

**void AddIcon(const wxIcon& icon)**

Adds the icon to the collection; if the collection already contains an icon with the same width and height, it is replaced by the new one.

**wxIconBundle::GetIcon**

**const wxIcon& GetIcon(const wxSize& size) const**

Returns the icon with the given size; if no such icon exists, returns the icon with size `wxSYS_ICON_X/wxSYS_ICON_Y`; if no such icon exists, returns the first icon in the bundle. If `size = wxSize( -1, -1 )`, returns the icon with size `wxSYS_ICON_X/wxSYS_ICON_Y`.

**const wxIcon& GetIcon(wxCoord size = -1) const**

Same as `GetIcon( wxSize( size, size ) )`.

**wxIconBundle::operator=**

**const wxIconBundle& operator=(const wxIconBundle& ic)**

Assignment operator.

## wxIconLocation

`wxIconLocation` is a tiny class describing the location of an (external, i.e. not embedded into the application resources) icon. For most platforms it simply contains the file name but under some others (notably Windows) the same file may contain multiple icons and so this class also stores the index of the icon inside the file.

In any case, its details should be of no interest to the application code and most of them are not even documented here (on purpose) as it is only meant to be used as an opaque class: the application may get the object of this class from somewhere and the only reasonable thing to do with it later is to create a `wxIcon` (p. 894) from it.

**Derived from**

None.

**Include files**

<wx/iconloc.h>

**See also**

`wxIcon` (p. 894), `wxFileType::GetIcon` (p. 642)

**wxIconLocation::IsOk****bool IsOk() const**

Returns `true` if the object is valid, i.e. was properly initialized, and `false` otherwise.

**wxIconizeEvent**

An event being sent when the frame is iconized (minimized) or restored.

Currently only `wxMSW` and `wxGTK` generate such events.

**Derived from**

*wxEvent* (p. 572)

*wxObject* (p. 1148)

**Include files**

<wx/event.h>

**Event table macros**

To process an iconize event, use this event handler macro to direct input to a member function that takes a `wxIconizeEvent` argument.

**EVT\_ICONIZE(func)**                      Process a `wxEVT_ICONIZE` event.

**See also**

*Event handling overview* (p. 2077), *wxTopLevelWindow::Iconize* (p. 1715),

*wxTopLevelWindow::IsIconized* (p. 1715)

**wxIconizeEvent::wxIconizeEvent**

**wxIconizeEvent(int id = 0, bool iconized = true)**

Constructor.

**wxIconizeEvent::IsIconized**

**bool IsIconized() const**

Returns `true` if the frame has been iconized, `false` if it has been restored.

**wxIdleEvent**

This class is used for idle events, which are generated when the system becomes idle. Note that, unless you do something specifically, the idle events are not sent if the system remains idle once it has become it, e.g. only a single idle event will be generated until something else resulting in more normal events happens and only then is the next idle event sent again. If you need to ensure a continuous stream of idle events, you can either use *RequestMore* (p. 905) method in your handler or call *wxWakeUpIdle* (p. 1912) periodically (for example from timer event), but note that both of these approaches (and especially the first one) increase the system load and so should be avoided if possible.

By default, idle events are sent to all windows (and also *wxApp* (p. 45), as usual). If this is causing a significant overhead in your application, you can call *wxIdleEvent::SetMode* (p. 905) with the value `wxIDLE_PROCESS_SPECIFIED`, and set the `wxWS_EX_PROCESS_IDLE` extra window style for every window which should receive idle events.

### Derived from

*wxEvent* (p. 572)  
*wxObject* (p. 1148)

### Include files

<wx/event.h>

### Event table macros

To process an idle event, use this event handler macro to direct input to a member function that takes a *wxIdleEvent* argument.

**EVT\_IDLE(func)**                      Process a *wxEVT\_IDLE* event.

### See also

*Event handling overview* (p. 2077), *wxUpdateUIEvent* (p. 1752),  
*wxWindow::OnInternalIdle* (p. 1827)

## **wxIdleEvent::wxIdleEvent**

**wxIdleEvent()**

Constructor.

## **wxIdleEvent::CanSend**

**static bool CanSend(wxWindow\* window)**

Returns `true` if it is appropriate to send idle events to this window.

This function looks at the mode used (see *wxIdleEvent::SetMode* (p. 905)), and the `wxWS_EX_PROCESS_IDLE` style in *window* to determine whether idle events should be sent to this window now. By default this will always return `true` because the update mode is initially `wxIDLE_PROCESS_ALL`. You can change the mode to only send idle events to



windows with the `wxWS_EX_PROCESS_IDLE` extra window style set.

**See also**

*wxIdleEvent::SetMode* (p. 905)

**wxIdleEvent::GetMode**

**static wxIdleMode GetMode()**

Static function returning a value specifying how wxWidgets will send idle events: to all windows, or only to those which specify that they will process the events.

See *wxIdleEvent::SetMode* (p. 905).

**wxIdleEvent::RequestMore**

**void RequestMore(bool needMore = true)**

Tells wxWidgets that more processing is required. This function can be called by an OnIdle handler for a window or window event handler to indicate that `wxApp::OnIdle` should forward the OnIdle event once more to the application windows. If no window calls this function during OnIdle, then the application will remain in a passive event loop (not calling OnIdle) until a new event is posted to the application by the windowing system.

**See also**

*wxIdleEvent::MoreRequested* (p. 905)

**wxIdleEvent::MoreRequested**

**bool MoreRequested() const**

Returns true if the OnIdle function processing this event requested more processing time.

**See also**

*wxIdleEvent::RequestMore* (p. 905)

**wxIdleEvent::SetMode**

**static void SetMode(wxIdleMode mode)**

Static function for specifying how wxWidgets will send idle events: to all windows, or only to those which specify that they will process the events.

*mode* can be one of the following values. The default is `wxIDLE_PROCESS_ALL`.

```
enum wxIdleMode
{
    // Send idle events to all windows
    wxIDLE_PROCESS_ALL,
```

```
        // Send idle events to windows that have
        // the wxWS_EX_PROCESS_IDLE flag specified
        wxIDLE_PROCESS_SPECIFIED
    };
```

## wxImage

This class encapsulates a platform-independent image. An image can be created from data, or using *wxBitmap::ConvertToImage* (p. 128). An image can be loaded from a file in a variety of formats, and is extensible to new formats via image format handlers. Functions are available to set and get image bits, so it can be used for basic image manipulation.

A *wxImage* cannot (currently) be drawn directly to a *wxDC* (p. 456). Instead, a platform-specific *wxBitmap* (p. 123) object must be created from it using the *wxBitmap::wxBitmap(wxImage, int depth)* (p. 124) constructor. This bitmap can then be drawn in a device context, using *wxDC::DrawBitmap* (p. 460).

One colour value of the image may be used as a mask colour which will lead to the automatic creation of a *wxMask* (p. 1036) object associated to the bitmap object.

### Alpha channel support

Starting from wxWidgets 2.5.0 *wxImage* supports alpha channel data, that is in addition to a byte for the red, green and blue colour components for each pixel it also stores a byte representing the pixel opacity. An alpha value of 0 corresponds to a transparent pixel (null opacity) while a value of 255 means that the pixel is 100% opaque.

Unlike RGB data, not all images have an alpha channel and before using *GetAlpha* (p. 915) you should check if this image contains an alpha channel with *HasAlpha* (p. 918). Note that currently only images loaded from PNG files with transparency information will have an alpha channel but alpha support will be added to the other formats as well (as well as support for saving images with alpha channel which also isn't implemented).

### Available image handlers

The following image handlers are available. **wxBMPHandler** is always installed by default. To use other image formats, install the appropriate handler with *wxImage::AddHandler* (p. 910) or call *wxInitAllImageHandlers* (p. 1911).

<i>wxBMPHandler</i>	For loading and saving, always installed.
<i>wxPNGHandler</i>	For loading (including alpha support) and saving.
<i>wxJPEGHandler</i>	For loading and saving.
<i>wxGIFHandler</i>	Only for loading, due to legal issues.
<i>wxPCXHandler</i>	For loading and saving (see below).
<i>wxPNMHandler</i>	For loading and saving (see below).
<i>wxTIFFHandler</i>	For loading and saving.

<code>wxTGAHandler</code>	For loading only.
<code>wxIFFHandler</code>	For loading only.
<code>wxXPMHandler</code>	For loading and saving.
<code>wxICOHandler</code>	For loading and saving.
<code>wxCURHandler</code>	For loading and saving.
<code>wxANIHandler</code>	For loading only.

When saving in PCX format, **wxPCXHandler** will count the number of different colours in the image; if there are 256 or less colours, it will save as 8 bit, else it will save as 24 bit.

Loading PNMs only works for ASCII or raw RGB images. When saving in PNM format, **wxPNMHandler** will always save as raw RGB.

#### Derived from

`wxObject` (p. 1148)

#### Include files

`<wx/image.h>`

#### See also

`wxBitmap` (p. 123), `wxInitAllImageHandlers` (p. 1911)

### **wxImage::wxImage**

#### **wxImage()**

Default constructor.

#### **wxImage(const wxImage& image)**

Copy constructor, uses *reference counting* (p. 2046).

#### **wxImage(const wxBitmap& bitmap)**

(Deprecated form, use `wxBitmap::ConvertToImage` (p. 128) instead.) Constructs an image from a platform-dependent bitmap. This preserves mask information so that bitmaps and images can be converted back and forth without loss in that respect.

#### **wxImage(int width, int height, bool clear=true)**

Creates an image with the given width and height. If *clear* is true, the new image will be initialized to black. Otherwise, the image data will be uninitialized.

#### **wxImage(int width, int height, unsigned char\* data, bool static\_data=false)**

Creates an image from given data with the given width and height. If *static\_data* is true, then `wxImage` will not delete the actual image data in its destructor, otherwise it will free it by calling `free()`.

**`wxImage(const wxString& name, long type = wxBITMAP_TYPE_ANY, int index = -1)`**

**`wxImage(const wxString& name, const wxString& mimetype, int index = -1)`**

Loads an image from a file.

**`wxImage(wxInputStream& stream, long type = wxBITMAP_TYPE_ANY, int index = -1)`**

**`wxImage(wxInputStream& stream, const wxString& mimetype, int index = -1)`**

Loads an image from an input stream.

**`wxImage(const char* const* xpmData)`**

Creates an image from XPM data.

### Parameters

*width*

Specifies the width of the image.

*height*

Specifies the height of the image.

*name*

Name of the file from which to load the image.

*stream*

Opened input stream from which to load the image. Currently, the stream must support seeking.

*type*

May be one of the following:

<code>wxBITMAP_TYPE_BMP</code>	Load a Windows bitmap file.
<code>wxBITMAP_TYPE_GIF</code>	Load a GIF bitmap file.
<code>wxBITMAP_TYPE_JPEG</code>	Load a JPEG bitmap file.
<code>wxBITMAP_TYPE_PNG</code>	Load a PNG bitmap file.
<code>wxBITMAP_TYPE_PCX</code>	Load a PCX bitmap file.
<code>wxBITMAP_TYPE_PNM</code>	Load a PNM bitmap file.
<code>wxBITMAP_TYPE_TIF</code>	Load a TIFF bitmap file.

<code>wxBITMAP_TYPE_TGA</code>	Load a TGA bitmap file.
<code>wxBITMAP_TYPE_XPM</code>	Load a XPM bitmap file.
<code>wxBITMAP_TYPE_ICO</code>	Load a Windows icon file (ICO).
<code>wxBITMAP_TYPE_CUR</code>	Load a Windows cursor file (CUR).
<code>wxBITMAP_TYPE_ANI</code>	Load a Windows animated cursor file (ANI).
<code>wxBITMAP_TYPE_ANY</code>	Will try to autodetect the format.

### *mimetype*

MIME type string (for example 'image/jpeg')

### *index*

Index of the image to load in the case that the image file contains multiple images. This is only used by GIF, ICO and TIFF handlers. The default value (-1) means "choose the default image" and is interpreted as the first image (index=0) by the GIF and TIFF handler and as the largest and most colourful one by the ICO handler.

### *xpmData*

A pointer to XPM image data.

### **Remarks**

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

Note: any handler other than BMP must be previously initialized with `wxImage::AddHandler` (p. 910) or `wxInitAllImageHandlers` (p. 1911).

```
Note: you can use GetOptionInt (p. 919) to get the hotspot for loaded cursor file:      int
hotspot_x = image.GetOptionInt(wxIMAGE_OPTION_CUR_HOTSPOT_X);
int hotspot_y =
image.GetOptionInt(wxIMAGE_OPTION_CUR_HOTSPOT_Y);
```

### **See also**

`wxImage::LoadFile` (p. 920)

**wxPython note:** Constructors supported by wxPython are:

<b><code>wxImage(name, flag)</code></b>	Loads an image from a file
<b><code>wxNullImage()</code></b>	Create a null image (has no size or image data)
<b><code>wxEmptyImage(width, height)</code></b>	Creates an empty image of the given size
<b><code>wxImageFromMime(name, mimetype)</code></b>	Creates an image from the given file of the given mimetype

**wxImageFromBitmap(bitmap)** Creates an image from a platform-dependent bitmap

**wxPerl note:** Constructors supported by wxPerl are:

- `Image->new( bitmap )`
- `Image->new( icon )`
- `Image->new( width, height )`
- `Image->new( width, height, data )`
- `Image->new( file, type, index )`
- `Image->new( file, mimetype, index )`
- `Image->new( stream, type, index )`
- `Image->new( stream, mimetype, index )`

### **wxImage::~wxImage**

**~wxImage()**

Destructor. See *reference-counted object destruction* (p. 2046) for more info.

### **wxImage::AddHandler**

**static void AddHandler(wxImageHandler\* handler)**

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

*handler*

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

### **See also**

*wxImageHandler* (p. 930)

**bool CanRead(const wxString& filename)**

returns true if the current image handlers can read this file

**wxPython note:** In wxPython this static method is named `wxImage_AddHandler`.

### **wxImage::Blur**

**wxImage Blur(int blurRadius)**

Blurs the image in both horizontal and vertical directions by the specified pixel *blurRadius*. This should not be used when using a single mask colour for transparency.

**See also**

*BlurHorizontal* (p. 911) *BlurVertical* (p. 911)

**wxImage::BlurHorizontal**

**wxImage BlurHorizontal(int blurRadius)**

Blurs the image in the horizontal direction only. This should not be used when using a single mask colour for transparency. **See also**

*Blur* (p. 910) *BlurVertical* (p. 911)

**wxImage::BlurVertical**

**wxImage BlurVertical(int blurRadius)**

Blurs the image in the vertical direction only. This should not be used when using a single mask colour for transparency.

**See also**

*Blur* (p. 910) *BlurHorizontal* (p. 911)

**wxImage::CleanUpHandlers**

**static void CleanUpHandlers()**

Deletes all image handlers.

This function is called by wxWidgets on exit.

**wxImage::ComputeHistogram**

**unsigned long ComputeHistogram(wxImageHistogram& histogram) const**

Computes the histogram of the image. *histogram* is a reference to wxImageHistogram object. wxImageHistogram is a specialization of wxHashMap (p. 798) "template" and is defined as follows:

```
class WXDLLEXPORT wxImageHistogramEntry
{
public:
    wxImageHistogramEntry() : index(0), value(0) {}
    unsigned long index;
    unsigned long value;
};

WX_DECLARE_EXPORTED_HASH_MAP(unsigned long, wxImageHistogramEntry,
                             wxIntegerHash, wxIntegerEqual,
```

```
wxImageHistogram);
```

**Return value**

Returns number of colours in the histogram.

**wxImage::ConvertAlphaToMask**

**bool ConvertAlphaToMask(unsigned char threshold = 128)**

If the image has alpha channel, this method converts it to mask. All pixels with alpha value less than *threshold* are replaced with mask colour and the alpha channel is removed. Mask colour is chosen automatically using *FindFirstUnusedColour* (p. 913).

If the image image doesn't have alpha channel, ConvertAlphaToMask does nothing.

**Return value**

false if FindFirstUnusedColour returns false, true otherwise.

**wxImage::ConvertToBitmap**

**wxBitmap ConvertToBitmap() const**

Deprecated, use equivalent *wxBitmap constructor* (p. 124)(which takes wxImage and depth as its arguments) instead.

**wxImage::ConvertToGreyscale**

**wxImage ConvertToGreyscale(double lr = 0.299, double lg = 0.587, double lb = 0.114) const**

Returns a greyscale version of the image. The returned image uses the luminance component of the original to calculate the greyscale. Defaults to using ITU-T BT.601 when converting to YUV, where every pixel equals  $(R * lr) + (G * lg) + (B * lb)$ .

**wxImage::ConvertToMono**

**wxImage ConvertToMono(unsigned char r, unsigned char g, unsigned char b) const**

Returns monochromatic version of the image. The returned image has white colour where the original has  $(r,g,b)$  colour and black colour everywhere else.

**wxImage::Copy**

**wxImage Copy() const**

Returns an identical copy of the image.

**wxImage::Create**



**bool Create(int width, int height, bool clear=true)**

Creates a fresh image. If *clear* is true, the new image will be initialized to black. Otherwise, the image data will be uninitialized.

**Parameters**

*width*

The width of the image in pixels.

*height*

The height of the image in pixels.

**Return value**

true if the call succeeded, false otherwise.

**wxImage::Destroy**

**void Destroy()**

Destroys the image data.

**wxImage::FindFirstUnusedColour**

**bool FindFirstUnusedColour(unsigned char \* r, unsigned char \* g, unsigned char \* b, unsigned char startR = 1, unsigned char startG = 0, unsigned char startB = 0)**

**Parameters**

*r,g,b*

Pointers to variables to save the colour.

*startR,startG,startB*

Initial values of the colour. Returned colour will have RGB values equal to or greater than these.

Finds the first colour that is never used in the image. The search begins at given initial colour and continues by increasing R, G and B components (in this order) by 1 until an unused colour is found or the colour space exhausted.

**Return value**

Returns false if there is no unused colour left, true on success.

**Notes**

Note that this method involves computing the histogram, which is computationally intensive operation.

**wxImage::FindHandler****static wxImageHandler\* FindHandler(const wxString& name)**

Finds the handler with the given name.

**static wxImageHandler\* FindHandler(const wxString& extension, long imageType)**

Finds the handler associated with the given extension and type.

**static wxImageHandler\* FindHandler(long imageType)**

Finds the handler associated with the given image type.

**static wxImageHandler\* FindHandlerMime(const wxString& mimetype)**

Finds the handler associated with the given MIME type.

*name*

The handler name.

*extension*

The file extension, such as "bmp".

*imageType*

The image type, such as wxBITMAP\_TYPE\_BMP.

*mimetype*

MIME type.

**Return value**

A pointer to the handler if found, NULL otherwise.

**See also**

*wxImageHandler* (p. 930)

**wxImage::GetImageExtWildcard****static wxString GetImageExtWildcard()**

Iterates all registered wxImageHandler objects, and returns a string containing file extension masks suitable for passing to file open/save dialog boxes.

**Return value**

The format of the returned string is "(\*.ext1;\*.ext2)|\*.ext1;\*.ext2".

It is usually a good idea to prepend a description before passing the result to the dialog.

Example:

```
wxFileDialog FileDlg( this, "Choose Image", ::wxGetCwd(), "",
    _( "Image Files " ) + wxImage::GetImageExtWildcard(), wxFD_OPEN );
```

### See also

*wxImageHandler* (p. 930)

## **wxImage::GetAlpha**

**unsigned char GetAlpha(int x, int y) const**

Returns the alpha value for the given pixel. This function may only be called for the images with alpha channel, use *HasAlpha* (p. 918) to check for this.

The returned value is the *opacity* of the image, i.e. the value of 0 corresponds to the transparent pixels while the value of 255 -- to the opaque ones.

**unsigned char \* GetAlpha() const**

Returns pointer to the array storing the alpha values for this image. This pointer is `NULL` for the images without the alpha channel. If the image does have it, this pointer may be used to directly manipulate the alpha values which are stored as the *RGB* (p. 915) ones.

## **wxImage::GetBlue**

**unsigned char GetBlue(int x, int y) const**

Returns the blue intensity at the given coordinate.

## **wxImage::GetData**

**unsigned char\* GetData() const**

Returns the image data as an array. This is most often used when doing direct image manipulation. The return value points to an array of characters in `RGBRGBRGB...` format in the top-to-bottom, left-to-right order, that is the first RGB triplet corresponds to the pixel first pixel of the first row, the second one --- to the second pixel of the first row and so on until the end of the first row, with second row following after it and so on.

You should not delete the returned pointer nor pass it to *wxImage::SetData* (p. 927).

## **wxImage::GetGreen**

**unsigned char GetGreen(int x, int y) const**

Returns the green intensity at the given coordinate.

## **wxImage::GetImageCount**

```
static int GetImageCount(const wxString& filename, long type =  
wxBITMAP_TYPE_ANY)
```

```
static int GetImageCount(wxInputStream& stream, long type =  
wxBITMAP_TYPE_ANY)
```

If the image file contains more than one image and the image handler is capable of retrieving these individually, this function will return the number of available images.

*name*

Name of the file to query.

*stream*

Opened input stream with image data. Currently, the stream must support seeking.

*type*

May be one of the following:

wxBITMAP_TYPE_BMP	Load a Windows bitmap file.
wxBITMAP_TYPE_GIF	Load a GIF bitmap file.
wxBITMAP_TYPE_JPEG	Load a JPEG bitmap file.
wxBITMAP_TYPE_PNG	Load a PNG bitmap file.
wxBITMAP_TYPE_PCX	Load a PCX bitmap file.
wxBITMAP_TYPE_PNM	Load a PNM bitmap file.
wxBITMAP_TYPE_TIF	Load a TIFF bitmap file.
wxBITMAP_TYPE_XPM	Load a XPM bitmap file.
wxBITMAP_TYPE_ICO	Load a Windows icon file (ICO).
wxBITMAP_TYPE_CUR	Load a Windows cursor file (CUR).
wxBITMAP_TYPE_ANI	Load a Windows animated cursor file (ANI).
wxBITMAP_TYPE_ANY	Will try to autodetect the format.

### **Return value**

Number of available images. For most image handlers, this is 1 (exceptions are TIFF and ICO formats).

### **wxImage::GetHandlers**

```
static wxList& GetHandlers()
```

Returns the static list of image format handlers.

**See also**

*wxImageHandler* (p. 930)

**wxImage::GetHeight**

**int GetHeight() const**

Gets the height of the image in pixels.

**wxImage::GetMaskBlue**

**unsigned char GetMaskBlue() const**

Gets the blue value of the mask colour.

**wxImage::GetMaskGreen**

**unsigned char GetMaskGreen() const**

Gets the green value of the mask colour.

**wxImage::GetMaskRed**

**unsigned char GetMaskRed() const**

Gets the red value of the mask colour.

**wxImage::GetOrFindMaskColour**

**bool GetOrFindMaskColour(unsigned char \*r, unsigned char \*g, unsigned char \*b) const**

Get the current mask colour or find a suitable unused colour that could be used as a mask colour. Returns `true` if the image currently has a mask.

**wxImage::GetPalette**

**const wxPalette& GetPalette() const**

Returns the palette associated with the image. Currently the palette is only used when converting to `wxBitmap` under Windows. Some of the `wxImage` handlers have been modified to set the palette if one exists in the image file (usually 256 or less colour images in GIF or PNG format).

**wxImage::GetRed**

**unsigned char GetRed(int x, int y) const**

Returns the red intensity at the given coordinate.

### **wxImage::GetSubImage**

**wxImage GetSubImage(const wxRect& rect) const**

Returns a sub image of the current one as long as the rect belongs entirely to the image.

### **wxImage::GetWidth**

**int GetWidth() const**

Gets the width of the image in pixels.

#### **See also**

*wxImage::GetHeight* (p. 917)

### **HSVValue::HSVValue**

**HSVValue(double h = 0.0, double s = 0.0, double v = 0.0)**

Constructor for HSVValue, an object that contains values for hue, saturation and value which represent the value of a color. It is used by *wxImage::HSVtoRGB* (p. 918) and *wxImage::RGBtoHSV* (p. 922), which converts between HSV color space and RGB color space.

**wxPython note:** use `wxImage_HSVValue` in wxPython

### **wxImage::HSVtoRGB**

**wxImage::RGBValue HSVtoRGB(const HSVValue & hsv)**

Converts a color in HSV color space to RGB color space.

### **wxImage::HasAlpha**

**bool HasAlpha() const**

Returns true if this image has alpha channel, false otherwise.

#### **See also**

*GetAlpha* (p. 915), *SetAlpha* (p. 927)

### **wxImage::HasMask**

**bool HasMask() const**

Returns true if there is a mask active, false otherwise.

**wxImage::GetOption****wxString GetOption(const wxString& name) const**

Gets a user-defined option. The function is case-insensitive to *name*.

For example, when saving as a JPEG file, the option **quality** is used, which is a number between 0 and 100 (0 is terrible, 100 is very good).

**See also**

*wxImage::SetOption* (p. 929), *wxImage::GetOptionInt* (p. 919), *wxImage::HasOption* (p. 919)

**wxImage::GetOptionInt****int GetOptionInt(const wxString& name) const**

Gets a user-defined option as an integer. The function is case-insensitive to *name*.

If the given option is not present, the function returns 0. Use *wxImage::HasOption* (p. 919) if 0 is a possibly valid value for the option.

Options for `wxPNGHandler`: `wxIMAGE_OPTION_PNG_FORMAT`    Format for saving a PNG file.

`wxIMAGE_OPTION_PNG_BITDEPTH`    Bit depth for every channel (R/G/B/A).

Supported values for `wxIMAGE_OPTION_PNG_FORMAT`: `wxPNG_TYPE_COLOUR`  
Stores RGB image.

`wxPNG_TYPE_GREY`    Stores grey image, converts from RGB.

`wxPNG_TYPE_GREY_RED` Stores grey image, uses red value as grey.

**See also**

*wxImage::SetOption* (p. 929), *wxImage::GetOption* (p. 919)

**wxImage::HasOption****bool HasOption(const wxString& name) const**

Returns true if the given option is present. The function is case-insensitive to *name*.

**See also**

*wxImage::SetOption* (p. 929), *wxImage::GetOption* (p. 919), *wxImage::GetOptionInt* (p. 919)

**wxImage::InitAlpha****void InitAlpha()**

Initializes the image alpha channel data. It is an error to call it if the image already has alpha data. If it doesn't, alpha data will be by default initialized to all pixels being fully opaque. But if the image has a mask colour, all mask pixels will be completely transparent.

### **wxImage::InitStandardHandlers**

#### **static void InitStandardHandlers()**

Internal use only. Adds standard image format handlers. It only install BMP for the time being, which is used by wxBitmap.

This function is called by wxWidgets on startup, and shouldn't be called by the user.

#### **See also**

*wxImageHandler* (p. 930), *wxInitAllImageHandlers* (p. 1911)

### **wxImage::InsertHandler**

#### **static void InsertHandler(wxImageHandler\* handler)**

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

*handler*

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

#### **See also**

*wxImageHandler* (p. 930)

### **wxImage::IsTransparent**

#### **bool IsTransparent(int x, int y, unsigned char threshold = 128) const**

Returns `true` if the given pixel is transparent, i.e. either has the mask colour if this image has a mask or if this image has alpha channel and alpha value of this pixel is strictly less than *threshold*.

### **wxImage::LoadFile**

#### **bool LoadFile(const wxString& name, long type = wxBITMAP\_TYPE\_ANY, int index = -1)**

#### **bool LoadFile(const wxString& name, const wxString& mimetype, int index = -1)**

Loads an image from a file. If no handler type is provided, the library will try to autodetect the format.

#### **bool LoadFile(wxInputStream& stream, long type, int index = -1)**



**bool LoadFile(wxInputStream& stream, const wxString& mimetype, int index = -1)**

Loads an image from an input stream.

### Parameters

*name*

Name of the file from which to load the image.

*stream*

Opened input stream from which to load the image. Currently, the stream must support seeking.

*type*

One of the following values:

<b>wxBITMAP_TYPE_BMP</b>	Load a Windows image file.
<b>wxBITMAP_TYPE_GIF</b>	Load a GIF image file.
<b>wxBITMAP_TYPE_JPEG</b>	Load a JPEG image file.
<b>wxBITMAP_TYPE_PCX</b>	Load a PCX image file.
<b>wxBITMAP_TYPE_PNG</b>	Load a PNG image file.
<b>wxBITMAP_TYPE_PNM</b>	Load a PNM image file.
<b>wxBITMAP_TYPE_TIF</b>	Load a TIFF image file.
<b>wxBITMAP_TYPE_XPM</b>	Load a XPM image file.
<b>wxBITMAP_TYPE_ICO</b>	Load a Windows icon file (ICO).
<b>wxBITMAP_TYPE_CUR</b>	Load a Windows cursor file (CUR).
<b>wxBITMAP_TYPE_ANI</b>	Load a Windows animated cursor file (ANI).
<b>wxBITMAP_TYPE_ANY</b>	Will try to autodetect the format.

*mimetype*

MIME type string (for example 'image/jpeg')

*index*

Index of the image to load in the case that the image file contains multiple images. This is only used by GIF, ICO and TIFF handlers. The default value (-1) means "choose the default image" and is interpreted as the first image (index=0) by the GIF and TIFF handler and as the largest and most colourful one by the ICO handler.

### Remarks

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

Note: you can use *GetOptionInt* (p. 919) to get the hotspot for loaded cursor file:     int  
hotspot\_x = image.GetOptionInt(wxIMAGE\_OPTION\_CUR\_HOTSPOT\_X);  
          int hotspot\_y =  
image.GetOptionInt(wxIMAGE\_OPTION\_CUR\_HOTSPOT\_Y);

### Return value

true if the operation succeeded, false otherwise. If the optional index parameter is out of range, false is returned and a call to *wxLogError()* takes place.

### See also

*wxImage::SaveFile* (p. 924)

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

**LoadFile(filename, type)**     Loads an image of the given type from a file

**LoadMimeFile(filename, mimetype)**     Loads an image of the given  
  mimetype from a file

**wxPerl note:** Methods supported by wxPerl are:

- >LoadFile( name, type )
- >LoadFile( name, mimetype )

### **wxImage::IsOk**

**bool IsOk() const**

Returns true if image data is present.

### **RGBValue::RGBValue**

**RGBValue(unsigned char *r* = 0, unsigned char *g* = 0, unsigned char *b* = 0)**

Constructor for RGBValue, an object that contains values for red, green and blue which represent the value of a color. It is used by *wxImage::HSVtoRGB* (p. 918) and *wxImage::RGBtoHSV* (p. 922), which converts between HSV color space and RGB color space.

**wxPython note:** use *wxImage\_RGBValue* in wxPython

### **wxImage::RGBtoHSV**

**wxImage::HSVValue RGBtoHSV(const RGBValue& *rgb*)**

Converts a color in RGB color space to HSV color space.

### **wxImage::RemoveHandler**

**static bool RemoveHandler(const wxString& name)**

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

*name*

The handler name.

### **Return value**

true if the handler was found and removed, false otherwise.

### **See also**

*wxImageHandler* (p. 930)

### **wxImage::Mirror**

**wxImage Mirror(bool horizontally = true) const**

Returns a mirrored copy of the image. The parameter *horizontally* indicates the orientation.

### **wxImage::Replace**

**void Replace(unsigned char r1, unsigned char g1, unsigned char b1, unsigned char r2, unsigned char g2, unsigned char b2)**

Replaces the colour specified by *r1,g1,b1* by the colour *r2,g2,b2*.

### **wxImage::Rescale**

**wxImage & Rescale(int width, int height, int quality = wxIMAGE\_QUALITY\_NORMAL)**

Changes the size of the image in-place by scaling it: after a call to this function, the image will have the given width and height.

For a description of the *quality* parameter, see the *Scale* (p. 926) function.

Returns the (modified) image itself.

### **See also**

*Scale* (p. 926)

### **wxImage::Resize**

**wxImage & Resize(const wxSize& size, const wxPoint& pos, int red = -1, int green = -1, int blue = -1)**

Changes the size of the image in-place without scaling it by adding either a border with the given colour or cropping as necessary. The image is pasted into a new image with the given *size* and background colour at the position *pos* relative to the upper left of the new image. If *red* = *green* = *blue* = -1 then use either the current mask colour if set or find, use, and set a suitable mask colour for any newly exposed areas.

Returns the (modified) image itself.

#### See also

*Size* (p. 927)

### wxImage::Rotate

**wxImage Rotate(double angle, const wxPoint& rotationCentre, bool interpolating = true, wxPoint\* offsetAfterRotation = NULL)**

Rotates the image about the given point, by *angle* radians. Passing true to *interpolating* results in better image quality, but is slower. If the image has a mask, then the mask colour is used for the uncovered pixels in the rotated image background. Else, black (rgb 0, 0, 0) will be used.

Returns the rotated image, leaving this image intact.

### wxImage::RotateHue

**void RotateHue(double angle)**

Rotates the hue of each pixel in the image by *angle*, which is a double in the range of -1.0 to +1.0, where -1.0 corresponds to -360 degrees and +1.0 corresponds to +360 degrees.

### wxImage::Rotate90

**wxImage Rotate90(bool clockwise = true) const**

Returns a copy of the image rotated 90 degrees in the direction indicated by *clockwise*.

### wxImage::SaveFile

**bool SaveFile(const wxString& name, int type) const**

**bool SaveFile(const wxString& name, const wxString& mimetype) const**

Saves an image in the named file.

**bool SaveFile(const wxString& name) const**

Saves an image in the named file. File type is determined from the extension of the file name. Note that this function may fail if the extension is not recognized! You can use one of the forms above to save images to files with non-standard extensions.

**bool SaveFile(wxOutputStream& stream, int type) const**

**bool SaveFile(wxOutputStream& stream, const wxString& mimetype) const**

Saves an image in the given stream.

#### Parameters

*name*

Name of the file to save the image to.

*stream*

Opened output stream to save the image to.

*type*

Currently these types can be used:

<b>wxBITMAP_TYPE_BMP</b>	Save a BMP image file.
<b>wxBITMAP_TYPE_JPEG</b>	Save a JPEG image file.
<b>wxBITMAP_TYPE_PNG</b>	Save a PNG image file.
<b>wxBITMAP_TYPE_PCX</b>	Save a PCX image file (tries to save as 8-bit if possible, falls back to 24-bit otherwise).
<b>wxBITMAP_TYPE_PNM</b>	Save a PNM image file (as raw RGB always).
<b>wxBITMAP_TYPE_TIFF</b>	Save a TIFF image file.
<b>wxBITMAP_TYPE_XPM</b>	Save a XPM image file.
<b>wxBITMAP_TYPE_ICO</b>	Save a Windows icon file (ICO) (the size may be up to 255 wide by 127 high. A single image is saved in 8 colors at the size supplied).
<b>wxBITMAP_TYPE_CUR</b>	Save a Windows cursor file (CUR).

*mimetype*

MIME type.

#### Return value

true if the operation succeeded, false otherwise.

#### Remarks

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

Note: you can use *GetOptionInt* (p. 919) to set the hotspot before saving an image into a cursor file (default hotspot is in the centre of the image):

```
image.SetOption(wxIMAGE_OPTION_CUR_HOTSPOT_X, hotspotX);  
image.SetOption(wxIMAGE_OPTION_CUR_HOTSPOT_Y, hotspotY);
```

**See also**

*wxImage::LoadFile* (p. 920)

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

**SaveFile(filename, type)** Saves the image using the given type to the named file

**SaveMimeFile(filename, mimetype)** Saves the image using the given mimetype to the named file

**wxPerl note:** Methods supported by wxPerl are:

- >SaveFile( name, type )
- >SaveFile( name, mimetype )

**wxImage::Scale**

**wxImage Scale(int width, int height, int quality = wxIMAGE\_QUALITY\_NORMAL) const**

Returns a scaled version of the image. This is also useful for scaling bitmaps in general as the only other way to scale bitmaps is to blit a wxMemoryDC into another wxMemoryDC.

*quality*

Determines what method to use for resampling the image. Can be one of the following:

**wxIMAGE\_QUALITY\_NORMAL** Uses the normal default scaling method of pixel replication

**wxIMAGE\_QUALITY\_HIGH** Uses bicubic and box averaging resampling methods for upsampling and downsampling respectively

It should be noted that although using wxIMAGE\_QUALITY\_HIGH produces much nicer looking results it is a slower method. Downsampling will use the box averaging method which seems to operate very fast. If you are upsampling larger images using this method you will most likely notice that it is a bit slower and in extreme cases it will be quite substantially slower as the bicubic algorithm has to process a lot of data.

It should also be noted that the high quality scaling may not work as expected when using a single mask colour for transparency, as the scaling will blur the image and will therefore remove the mask partially. Using the alpha channel will work.

Example:

```
// get the bitmap from somewhere
```

```
wxBitmap bmp = ...;

// rescale it to have size of 32*32
if ( bmp.GetWidth() != 32 || bmp.GetHeight() != 32 )
{
    wxImage image = bmp.ConvertToImage();
    bmp = wxBitmap(image.Scale(32, 32));

    // another possibility:
    image.Rescale(32, 32);
    bmp = image;
}
```

### See also

*Rescale* (p. 923)

### wxImage::Size

**wxImage Size(const wxSize& size, const wxPoint& pos, int red = -1, int green = -1, int blue = -1) const**

Returns a resized version of this image without scaling it by adding either a border with the given colour or cropping as necessary. The image is pasted into a new image with the given size and background colour at the position *pos* relative to the upper left of the new image. If *red* = *green* = *blue* = -1 then use either the current mask colour if set or find, use, and set a suitable mask colour for any newly exposed areas.

### See also

*Resize* (p. 923)

### wxImage::SetAlpha

**void SetAlpha(unsigned char \*alpha = NULL, bool static\_data = false)**

This function is similar to *SetData* (p. 927) and has similar restrictions. The pointer passed to it may however be `NULL` in which case the function will allocate the alpha array internally -- this is useful to add alpha channel data to an image which doesn't have any. If the pointer is not `NULL`, it must have one byte for each image pixel and be allocated with `malloc()`. `wxImage` takes ownership of the pointer and will free it unless *static\_data* parameter is set to `true` -- in this case the caller should do it.

**void SetAlpha(int x, int y, unsigned char alpha)**

Sets the alpha value for the given pixel. This function should only be called if the image has alpha channel data, use *HasAlpha* (p. 918) to check for this.

### wxImage::SetData

**void SetData(unsigned char\* data)**

Sets the image data without performing checks. The data given must have the size (width\*height\*3) or results will be unexpected. Don't use this method if you aren't sure you know what you are doing.

The data must have been allocated with `malloc()`, **NOT** with `operator new`.

After this call the pointer to the data is owned by the `wxImage` object, that will be responsible for deleting it. Do not pass to this function a pointer obtained through `wxImage::GetData` (p. 915).

### **wxImage::SetMask**

**void SetMask**(bool *hasMask* = true)

Specifies whether there is a mask or not. The area of the mask is determined by the current mask colour.

### **wxImage::SetMaskColour**

**void SetMaskColour**(unsigned char *red*, unsigned char *green*, unsigned char *blue*)

Sets the mask colour for this image (and tells the image to use the mask).

### **wxImage::SetMaskFromImage**

**bool SetMaskFromImage**(const wxImage& *mask*, unsigned char *mr*, unsigned char *mg*, unsigned char *mb*)

#### **Parameters**

*mask*

The mask image to extract mask shape from. Must have same dimensions as the image.

*mr,mg,mb*

RGB value of pixels in *mask* that will be used to create the mask.

Sets image's mask so that the pixels that have RGB value of *mr,mg,mb* in *mask* will be masked in the image. This is done by first finding an unused colour in the image, setting this colour as the mask colour and then using this colour to draw all pixels in the image who corresponding pixel in *mask* has given RGB value.

#### **Return value**

Returns false if *mask* does not have same dimensions as the image or if there is no unused colour left. Returns true if the mask was successfully applied.

#### **Notes**

Note that this method involves computing the histogram, which is computationally intensive operation.



**wxImage::SetOption****void SetOption(const wxString& name, const wxString& value)****void SetOption(const wxString& name, int value)**

Sets a user-defined option. The function is case-insensitive to *name*.

For example, when saving as a JPEG file, the option **quality** is used, which is a number between 0 and 100 (0 is terrible, 100 is very good).

**See also**

*wxImage::GetOption* (p. 919), *wxImage::GetOptionInt* (p. 919), *wxImage::HasOption* (p. 919)

**wxImage::SetPalette****void SetPalette(const wxPalette& palette)**

Associates a palette with the image. The palette may be used when converting *wxImage* to *wxBitmap* (MSW only at present) or in file save operations (none as yet).

**wxImage::SetRGB****void SetRGB(int x, int y, unsigned char red, unsigned char green, unsigned char blue)**

Sets the pixel at the given coordinate. This routine performs bounds-checks for the coordinate so it can be considered a safe way to manipulate the data, but in some cases this might be too slow so that the data will have to be set directly. In that case you will have to get access to the image data using the *GetData* (p. 915) method.

**wxImage::SetRGB****void SetRGB(wxRect & rect, unsigned char red, unsigned char green, unsigned char blue)**

Sets the colour of the pixels within the given rectangle. This routine performs bounds-checks for the coordinate so it can be considered a safe way to manipulate the data.

**wxImage::operator =****wxImage& operator =(const wxImage& image)**

Assignment operator, using *reference counting* (p. 2046).

**Parameters**

*image*

Image to assign.

**Return value**

Returns 'this' object.

## **wxImageHandler**

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

If you wish to extend the capabilities of `wxImage`, derive a class from `wxImageHandler` and add the handler using `wxImage::AddHandler` (p. 910) in your application initialisation.

**Note (Legal Issue)**

This software is based in part on the work of the Independent JPEG Group.

(Applies when `wxWidgets` is linked with JPEG support. `wxJPEGHandler` uses `libjpeg` created by IJG.)

**Derived from**

`wxObject` (p. 1148)

**Include files**

`<wx/image.h>`

**See also**

`wxImage` (p. 906), `wxInitAllImageHandlers` (p. 1911)

### **wxImageHandler::wxImageHandler**

**wxImageHandler()**

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

### **wxImageHandler::~~wxImageHandler**

**~wxImageHandler()**

Destroys the `wxImageHandler` object.

### **wxImageHandler::GetName**

**const wxString& GetName() const**

Gets the name of this handler.

### **wxImageHandler::GetExtension**

**const wxString& GetExtension() const**

Gets the file extension associated with this handler.

### **wxImageHandler::GetImageCount**

**int GetImageCount(wxInputStream& stream)**

If the image file contains more than one image and the image handler is capable of retrieving these individually, this function will return the number of available images.

*stream*

Opened input stream for reading image data. Currently, the stream must support seeking.

### **Return value**

Number of available images. For most image handlers, this is 1 (exceptions are TIFF and ICO formats).

### **wxImageHandler::GetType**

**long GetType() const**

Gets the image type associated with this handler.

### **wxImageHandler::GetMimeType**

**const wxString& GetMimeType() const**

Gets the MIME type associated with this handler.

### **wxImageHandler::LoadFile**

**bool LoadFile(wxImage\* image, wxInputStream& stream, bool verbose=true, int index=0)**

Loads a image from a stream, putting the resulting data into *image*. If the image file contains more than one image and the image handler is capable of retrieving these individually, *index* indicates which image to read from the stream.

### **Parameters**

*image*

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

*stream*

Opened input stream for reading image data.

*verbose*

If set to true, errors reported by the image handler will produce wxLogMessages.

*index*

The index of the image in the file (starting from zero).

### **Return value**

true if the operation succeeded, false otherwise.

### **See also**

*wxImage::LoadFile* (p. 920), *wxImage::SaveFile* (p. 924), *wxImageHandler::SaveFile* (p. 932)

### **wxImageHandler::SaveFile**

**bool SaveFile(wxImage\* image, wxOutputStream& stream)**

Saves a image in the output stream.

### **Parameters**

*image*

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

*stream*

Opened output stream for writing the data.

### **Return value**

true if the operation succeeded, false otherwise.

### **See also**

*wxImage::LoadFile* (p. 920), *wxImage::SaveFile* (p. 924), *wxImageHandler::LoadFile* (p. 931)

### **wxImageHandler::SetName**

**void SetName(const wxString& name)**

Sets the handler name.

### **Parameters**

*name*

Handler name.

### **wxImageHandler::SetExtension**

**void SetExtension(const wxString& *extension*)**

Sets the handler extension.

#### **Parameters**

*extension*

Handler extension.

### **wxImageHandler::SetMimeType**

**void SetMimeType(const wxString& *mimetype*)**

Sets the handler MIME type.

#### **Parameters**

*mimename*

Handler MIME type.

### **wxImageHandler::SetType**

**void SetType(long *type*)**

Sets the handler type.

#### **Parameters**

*name*

Handler type.

## **wxImageList**

A `wxImageList` contains a list of images, which are stored in an unspecified form. Images can have masks for transparent drawing, and can be made from a variety of sources including bitmaps and icons.

`wxImageList` is used principally in conjunction with `wxTreeCtrl` (p. 1728) and `wxListCtrl` (p. 980) classes.

#### **Derived from**

`wxObject` (p. 1148)

#### **Include files**

<wx/imaglist.h>

### See also

*wxTreeCtrl* (p. 1728), *wxListCtrl* (p. 980)

## **wxImageList::wxImageList**

### **wxImageList()**

Default constructor.

**wxImageList(int width, int height, const bool mask = true, int initialCount = 1)**

Constructor specifying the image size, whether image masks should be created, and the initial size of the list.

### Parameters

*width*

Width of the images in the list.

*height*

Height of the images in the list.

*mask*

true if masks should be created for all images.

*initialCount*

The initial size of the list.

### See also

*wxImageList::Create* (p. 935)

## **wxImageList::Add**

**int Add(const wxBitmap& bitmap, const wxBitmap& mask = wxNullBitmap)**

Adds a new image or images using a bitmap and optional mask bitmap.

**int Add(const wxBitmap& bitmap, const wxColour& maskColour)**

Adds a new image or images using a bitmap and mask colour.

**int Add(const wxIcon& icon)**

Adds a new image using an icon.

**Parameters***bitmap*

Bitmap representing the opaque areas of the image.

*mask*

Monochrome mask bitmap, representing the transparent areas of the image.

*maskColour*

Colour indicating which parts of the image are transparent.

*icon*

Icon to use as the image.

**Return value**

The new zero-based image index.

**Remarks**

The original bitmap or icon is not affected by the **Add** operation, and can be deleted afterwards.

If the bitmap is wider than the images in the list, then the bitmap will automatically be split into smaller images, each matching the dimensions of the image list. This does not apply when adding icons.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

**Add(bitmap, mask=wxNullBitmap)**

**AddWithColourMask(bitmap, colour)**

**AddIcon(icon)**

**wxImageList::Create**

**bool Create(int width, int height, const bool mask = true, int initialCount = 1)**

Initializes the list. See *wxImageList::wxImageList* (p. 934) for details.

**wxImageList::Draw**

**bool Draw(int index, wxDC& dc, int x, int y, int flags = wxIMAGELIST\_DRAW\_NORMAL, const bool solidBackground = false)**

Draws a specified image onto a device context.

**Parameters***index*

Image index, starting from zero.

*dc*

Device context to draw on.

*x*

X position on the device context.

*y*

Y position on the device context.

*flags*

How to draw the image. A bitlist of a selection of the following:

**wxIMAGELIST\_DRAW\_NORMAL** Draw the image normally.

**wxIMAGELIST\_DRAW\_TRANSPARENT** Draw the image with transparency.

**wxIMAGELIST\_DRAW\_SELECTED** Draw the image in selected state.

**wxIMAGELIST\_DRAW\_FOCUSED** Draw the image in a focused state.

*solidBackground*

For optimisation - drawing can be faster if the function is told that the background is solid.

**wxImageList::GetBitmap**

**wxBitmap GetBitmap(int *index*) const**

Returns the bitmap corresponding to the given index.

**wxImageList::GetIcon**

**wxIcon GetIcon(int *index*) const**

Returns the icon corresponding to the given index.

**wxImageList::GetImageCount**

**int GetImageCount() const**

Returns the number of images in the list.



**wxImageList::GetSize****bool GetSize(int *index*, int& *width*, int &*height*) const**

Retrieves the size of the images in the list. Currently, the *index* parameter is ignored as all images in the list have the same size.

**Parameters***index*

currently unused, should be 0

*width*

receives the width of the images in the list

*height*

receives the height of the images in the list

**Return value**

true if the function succeeded, false if it failed (for example, if the image list was not yet initialized).

**wxImageList::Remove****bool Remove(int *index*)**

Removes the image at the given position.

**wxImageList::RemoveAll****bool RemoveAll()**

Removes all the images in the list.

**wxImageList::Replace****bool Replace(int *index*, const wxBitmap& *bitmap*, const wxBitmap& *mask* = wxNullBitmap)**

Replaces the existing image with the new image.

Windows only.

**bool Replace(int *index*, const wxIcon& *icon*)**

Replaces the existing image with the new image.

**Parameters***bitmap*

Bitmap representing the opaque areas of the image.

*mask*

Monochrome mask bitmap, representing the transparent areas of the image.

*icon*

Icon to use as the image.

### Return value

true if the replacement was successful, false otherwise.

### Remarks

The original bitmap or icon is not affected by the **Replace** operation, and can be deleted afterwards.

**wxPython note:** The second form is called `ReplaceIcon` in wxPython.

## wxIndividualLayoutConstraint

Objects of this class are stored in the `wxLayoutConstraint` class as one of eight possible constraints that a window can be involved in.

Constraints are initially set to have the relationship `wxUnconstrained`, which means that their values should be calculated by looking at known constraints.

### Derived from

*wxObject* (p. 1148)

### Include files

<wx/layout.h>

### See also

*Overview and examples* (p. 2094), *wxLayoutConstraints* (p. 964), *wxWindow::SetConstraints* (p. 1837).

## Edges and relationships

The *wxEdge* enumerated type specifies the type of edge or dimension of a window.

<code>wxLeft</code>	The left edge.
<code>wxTop</code>	The top edge.
<code>wxRight</code>	The right edge.

<code>wxBottom</code>	The bottom edge.
<code>wxCentreX</code>	The x-coordinate of the centre of the window.
<code>wxCentreY</code>	The y-coordinate of the centre of the window.

The *wxRelationship* enumerated type specifies the relationship that this edge or dimension has with another specified edge or dimension. Normally, the user doesn't use these directly because functions such as *Below* and *RightOf* are a convenience for using the more general *Set* function.

<code>wxUnconstrained</code>	The edge or dimension is unconstrained (the default for edges).
<code>wxAsIs</code>	The edge or dimension is to be taken from the current window position or size (the default for dimensions).
<code>wxAbove</code>	The edge should be above another edge.
<code>wxBelow</code>	The edge should be below another edge.
<code>wxLeftOf</code>	The edge should be to the left of another edge.
<code>wxRightOf</code>	The edge should be to the right of another edge.
<code>wxSameAs</code>	The edge or dimension should be the same as another edge or dimension.
<code>wxPercentOf</code>	The edge or dimension should be a percentage of another edge or dimension.
<code>wxAbsolute</code>	The edge or dimension should be a given absolute value.

### **`wxIndividualLayoutConstraint::wxIndividualLayoutConstraint`**

#### **`void wxIndividualLayoutConstraint()`**

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()**

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()**

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 per)**

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.

## wxInitDialogEvent

A wxInitDialogEvent is sent as a dialog or panel is being initialised. Handlers for this event can transfer data to the window. The default handler calls *wxWindow::TransferDataToWindow* (p. 1851).

### Derived from

*wxEvt* (p. 572)

*wxObject* (p. 1148)

### Include files

<wx/event.h>

### Event table macros

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

**EVT\_INIT\_DIALOG(func)**                      Process a wxEVT\_INIT\_DIALOG event.

### See also

*Event handling overview* (p. 2077)

## wxInitDialogEvent::wxInitDialogEvent

**wxInitDialogEvent(int id = 0)**

Constructor.

## wxInputStream

wxInputStream is an abstract base class which may not be used directly.

### Derived from

*wxStreamBase* (p. 1544)

### Include files

<wx/stream.h>

**wxInputStream::wxInputStream****wxInputStream()**

Creates a dummy input stream.

**wxInputStream::~~wxInputStream****~wxInputStream()**

Destructor.

**wxInputStream::CanRead****bool CanRead() const**

Returns true if some data is available in the stream right now, so that calling *Read()* (p. 943) wouldn't block.

**wxInputStream::GetC****char GetC()**

Returns the first character in the input queue and removes it, blocking until it appears if necessary.

**Note**

If EOF, return value is undefined and *LastRead()* will return 0 and not 1.

**wxInputStream::Eof****bool Eof() const**

Returns true after an attempt has been made to read past the end of the stream.

**Note**

In wxWidgets 2.6.x and below some streams returned *Eof()* when the last byte had been read rather than when an attempt had been made to read past the last byte. If you want to avoid depending on one behaviour or the other then call *LastRead()* (p. 942) to check the number of bytes actually read.

**wxInputStream::LastRead****size\_t LastRead() const**

Returns the last number of bytes read.

**wxInputStream::Peek****char Peek()**

Returns the first character in the input queue without removing it.

**Note**

Blocks until something appears in the stream if necessary, if nothing ever does (i.e. EOF) `LastRead()` will return 0 (and the return value is undefined), otherwise `LastRead()` returns 1.

**wxInputStream::Read****wxInputStream& Read(void \*buffer, size\_t size)**

Reads the specified amount of bytes and stores the data in *buffer*.

**Warning**

The buffer absolutely needs to have at least the specified size.

**Return value**

This function returns a reference on the current object, so the user can test any states of the stream right away.

**wxInputStream& Read(wxOutputStream& stream\_out)**

Reads data from the input queue and stores it in the specified output stream. The data is read until an error is raised by one of the two streams.

**Return value**

This function returns a reference on the current object, so the user can test any states of the stream right away.

**wxInputStream::SeekI****off\_t SeekI(off\_t pos, wxSeekMode mode = wxFromStart)**

Changes the stream current position.

**Parameters***pos*

Offset to seek to.

*mode*

One of **wxFromStart**, **wxFromEnd**, **wxFromCurrent**.

**Return value**

The new stream position or `wxInvalidOffset` on error.

### **wxInputStream::Tell**

**off\_t Tell() const**

Returns the current stream position.

### **wxInputStream::Ungetch**

**size\_t Ungetch(const char\* buffer, size\_t size)**

This function is only useful in *read* mode. It is the manager of the "Write-Back" buffer. This buffer acts like a temporary buffer where data which has to be read during the next read IO call are put. This is useful when you get a big block of data which you didn't want to read: you can replace them at the top of the input queue by this way.

Be very careful about this call in connection with calling `SeekI()` on the same stream. Any call to `SeekI()` will invalidate any previous call to this method (otherwise you could `SeekI()` to one position, "unread" a few bytes there, `SeekI()` to another position and data would be either lost or corrupted).

#### **Return value**

Returns the amount of bytes saved in the Write-Back buffer.

**bool Ungetch(char c)**

This function acts like the previous one except that it takes only one character: it is sometimes shorter to use than the generic function.

## **wxIPAddress**

`wxIPAddress` is an abstract base class for all internet protocol address objects. Currently, only `wxIPv4Address` (p. 946) is implemented. An experimental implementation for IPv6, `wxIPv6Address`, is being developed.

#### **Derived from**

`wxSocketAddress` (p. 1472)

#### **Include files**

<wx/socket.h>

### **wxIPAddress::Hostname**

**virtual bool Hostname(const wxString& hostname)**

Set the address to *hostname*, which can be a host name or an IP-style address in a format



dependent on implementation.

**Return value**

Returns true on success, false if something goes wrong (invalid hostname or invalid IP address).

**virtual wxString Hostname()**

Returns the hostname which matches the IP address.

**wxIPAddress::IPAddress****virtual wxString IPAddress()**

Returns a wxString containing the IP address.

**wxIPAddress::Service****virtual bool Service(const wxString& service)**

Set the port to that corresponding to the specified *service*.

**Return value**

Returns true on success, false if something goes wrong (invalid service).

**virtual bool Service(unsigned short service)**

Set the port to that corresponding to the specified *service*.

**Return value**

Returns true on success, false if something goes wrong (invalid service).

**virtual unsigned short Service()**

Returns the current service.

**wxIPAddress::AnyAddress****virtual bool AnyAddress()**

Internally, this is the same as setting the IP address to **INADDR\_ANY**.

On IPV4 implementations, 0.0.0.0

On IPV6 implementations, ::

**Return value**

Returns true on success, false if something went wrong.

**wxIPAddress::LocalHost****virtual bool LocalHost()**

Set address to localhost.

On IPV4 implementations, 127.0.0.1

On IPV6 implementations, ::1

**Return value**

Returns true on success, false if something went wrong.

**wxIPAddress::IsLocalHost****virtual bool IsLocalHost()**

Determines if current address is set to localhost.

**Return value**

Returns true if address is localhost, false if internet address.

**wxIPv4address****Derived from**

*wxIPAddress* (p. 944)

**Include files**

<wx/socket.h>

**wxIPv4address::Hostname****bool Hostname(const wxString& *hostname*)**

Set the address to *hostname*, which can be a host name or an IP-style address in dot notation (a.b.c.d)

**Return value**

Returns true on success, false if something goes wrong (invalid hostname or invalid IP address).

**wxString Hostname()**

Returns the hostname which matches the IP address.

**wxIPv4address::IPAddress****wxString IPAddress()**

Returns a wxString containing the IP address in dot quad (127.0.0.1) format.

**wxIPv4address::Service****bool Service(const wxString& service)**

Set the port to that corresponding to the specified *service*.

**Return value**

Returns true on success, false if something goes wrong (invalid service).

**bool Service(unsigned short service)**

Set the port to that corresponding to the specified *service*.

**Return value**

Returns true on success, false if something goes wrong (invalid service).

**unsigned short Service()**

Returns the current service.

**wxIPv4address::AnyAddress****bool AnyAddress()**

Set address to any of the addresses of the current machine. Whenever possible, use this function instead of *wxIPv4address::LocalHost* (p. 947), as this correctly handles multi-homed hosts and avoids other small problems. Internally, this is the same as setting the IP address to **INADDR\_ANY**.

**Return value**

Returns true on success, false if something went wrong.

**wxIPv4address::LocalHost****bool LocalHost()**

Set address to localhost (127.0.0.1). Whenever possible, use the *wxIPv4address::AnyAddress* (p. 947), function instead of this one, as this will correctly handle multi-homed hosts and avoid other small problems.

**Return value**

Returns true on success, false if something went wrong.

## wxJoystick

wxJoystick allows an application to control one or more joysticks.

### Derived from

*wxObject* (p. 1148)

### Include files

<wx/joystick.h>

### See also

*wxJoystickEvent* (p. 954)

### wxJoystick::wxJoystick

**wxJoystick(int joystick = wxJOYSTICK1)**

Constructor. *joystick* may be one of wxJOYSTICK1, wxJOYSTICK2, indicating the joystick controller of interest.

### wxJoystick::~~wxJoystick

**~wxJoystick()**

Destroys the wxJoystick object.

### wxJoystick::GetButtonState

**int GetButtonState() const**

Returns the state of the joystick buttons. Every button is mapped to a single bit in the returned integer, with the first button being mapped to the least significant bit, and so on. A bitlist of wxJOY\_BUTTONn identifiers, where n is 1, 2, 3 or 4 is available for historical reasons.

### wxJoystick::GetManufacturerId

**int GetManufacturerId() const**

Returns the manufacturer id.

### wxJoystick::GetMovementThreshold

**int GetMovementThreshold() const**

Returns the movement threshold, the number of steps outside which the joystick is

deemed to have moved.

**wxJoystick::GetNumberAxes****int GetNumberAxes() const**

Returns the number of axes for this joystick.

**wxJoystick::GetNumberButtons****int GetNumberButtons() const**

Returns the number of buttons for this joystick.

**wxJoystick::GetNumberJoysticks****static int GetNumberJoysticks()**

Returns the number of joysticks currently attached to the computer.

**wxJoystick::GetPollingMax****int GetPollingMax() const**

Returns the maximum polling frequency.

**wxJoystick::GetPollingMin****int GetPollingMin() const**

Returns the minimum polling frequency.

**wxJoystick::GetProductId****int GetProductId() const**

Returns the product id for the joystick.

**wxJoystick::GetProductName****wxString GetProductName() const**

Returns the product name for the joystick.

**wxJoystick::GetPosition****wxPoint GetPosition() const**

Returns the x, y position of the joystick.

**wxJoystick::GetPOVPosition****int GetPOVPosition() const**

Returns the point-of-view position, expressed in continuous, one-hundredth of a degree units, but limited to return 0, 9000, 18000 or 27000. Returns -1 on error.

**wxJoystick::GetPOVCTSPosition****int GetPOVCTSPosition() const**

Returns the point-of-view position, expressed in continuous, one-hundredth of a degree units. Returns -1 on error.

**wxJoystick::GetRudderMax****int GetRudderMax() const**

Returns the maximum rudder position.

**wxJoystick::GetRudderMin****int GetRudderMin() const**

Returns the minimum rudder position.

**wxJoystick::GetRudderPosition****int GetRudderPosition() const**

Returns the rudder position.

**wxJoystick::GetUMax****int GetUMax() const**

Returns the maximum U position.

**wxJoystick::GetUMin****int GetUMin() const**

Returns the minimum U position.

**wxJoystick::GetUPosition****int GetUPosition() const**

Gets the position of the fifth axis of the joystick, if it exists.

**wxJoystick::GetVMax****int GetVMax() const**

Returns the maximum V position.

**wxJoystick::GetVMin****int GetVMin() const**

Returns the minimum V position.

**wxJoystick::GetVPosition****int GetVPosition() const**

Gets the position of the sixth axis of the joystick, if it exists.

**wxJoystick::GetXMax****int GetXMax() const**

Returns the maximum x position.

**wxJoystick::GetXMin****int GetXMin() const**

Returns the minimum x position.

**wxJoystick::GetYMax****int GetYMax() const**

Returns the maximum y position.

**wxJoystick::GetYMin****int GetYMin() const**

Returns the minimum y position.

**wxJoystick::GetZMax****int GetZMax() const**

Returns the maximum z position.

**wxJoystick::GetZMin**

**int GetZMin() const**

Returns the minimum z position.

**wxJoystick::GetZPosition**

**int GetZPosition() const**

Returns the z position of the joystick.

**wxJoystick::HasPOV**

**bool HasPOV() const**

Returns true if the joystick has a point of view control.

**wxJoystick::HasPOV4Dir**

**bool HasPOV4Dir() const**

Returns true if the joystick point-of-view supports discrete values (centered, forward, backward, left, and right).

**wxJoystick::HasPOVCTS**

**bool HasPOVCTS() const**

Returns true if the joystick point-of-view supports continuous degree bearings.

**wxJoystick::HasRudder**

**bool HasRudder() const**

Returns true if there is a rudder attached to the computer.

**wxJoystick::HasU**

**bool HasU() const**

Returns true if the joystick has a U axis.

**wxJoystick::HasV**

**bool HasV() const**

Returns true if the joystick has a V axis.

**wxJoystick::HasZ**

**bool HasZ() const**



Returns true if the joystick has a Z axis.

### **wxJoystick::IsOk**

**bool IsOk() const**

Returns true if the joystick is functioning.

### **wxJoystick::ReleaseCapture**

**bool ReleaseCapture()**

Releases the capture set by **SetCapture**.

#### **Return value**

true if the capture release succeeded.

#### **See also**

*wxJoystick::SetCapture* (p. 953), *wxJoystickEvent* (p. 954)

### **wxJoystick::SetCapture**

**bool SetCapture(wxWindow\* win, int pollingFreq = 0)**

Sets the capture to direct joystick events to *win*.

#### **Parameters**

*win*

The window that will receive joystick events.

*pollingFreq*

If zero, movement events are sent when above the threshold. If greater than zero, events are received every *pollingFreq* milliseconds.

#### **Return value**

true if the capture succeeded.

#### **See also**

*wxJoystick::ReleaseCapture* (p. 953), *wxJoystickEvent* (p. 954)

### **wxJoystick::SetMovementThreshold**

**void SetMovementThreshold(int threshold)**

Sets the movement threshold, the number of steps outside which the joystick is deemed to have moved.

## wxJoystickEvent

This event class contains information about mouse events, particularly events received by windows.

### Derived from

*wxEvent* (p. 572)

### Include files

<wx/event.h>

### Event table macros

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

<b>EVT_JOY_BUTTON_DOWN(func)</b>	Process a <i>wxEVT_JOY_BUTTON_DOWN</i> event.
<b>EVT_JOY_BUTTON_UP(func)</b>	Process a <i>wxEVT_JOY_BUTTON_UP</i> event.
<b>EVT_JOY_MOVE(func)</b>	Process a <i>wxEVT_JOY_MOVE</i> event.
<b>EVT_JOY_ZMOVE(func)</b>	Process a <i>wxEVT_JOY_ZMOVE</i> event.
<b>EVT_JOYSTICK_EVENTS(func)</b>	Processes all joystick events.

### See also

*wxJoystick* (p. 948)

## wxJoystickEvent::wxJoystickEvent

**wxJoystickEvent(WXTYPE eventType = 0, int state = 0, int joystick = wxJOYSTICK1, int change = 0)**

Constructor.

## wxJoystickEvent::ButtonDown

**bool ButtonDown(int button = wxJOY\_BUTTON\_ANY) const**

Returns true if the event was a down event from the specified button (or any button).

### Parameters

*button*

Can be *wxJOY\_BUTTONn* where *n* is 1, 2, 3 or 4; or *wxJOY\_BUTTON\_ANY* to indicate any button down event.

**wxJoystickEvent::ButtonIsDown****bool ButtonIsDown(int *button* = wxJOY\_BUTTON\_ANY) const**

Returns true if the specified button (or any button) was in a down state.

**Parameters***button*

Can be wxJOY\_BUTTONn where n is 1, 2, 3 or 4; or wxJOY\_BUTTON\_ANY to indicate any button down event.

**wxJoystickEvent::ButtonUp****bool ButtonUp(int *button* = wxJOY\_BUTTON\_ANY) const**

Returns true if the event was an up event from the specified button (or any button).

**Parameters***button*

Can be wxJOY\_BUTTONn where n is 1, 2, 3 or 4; or wxJOY\_BUTTON\_ANY to indicate any button down event.

**wxJoystickEvent::GetButtonChange****int GetButtonChange() const**

Returns the identifier of the button changing state. This is a wxJOY\_BUTTONn identifier, where n is one of 1, 2, 3, 4.

**wxJoystickEvent::GetButtonState****int GetButtonState() const**

Returns the down state of the buttons. This is a bitlist of wxJOY\_BUTTONn identifiers, where n is one of 1, 2, 3, 4.

**wxJoystickEvent::GetJoystick****int GetJoystick() const**

Returns the identifier of the joystick generating the event - one of wxJOYSTICK1 and wxJOYSTICK2.

**wxJoystickEvent::GetPosition****wxPoint GetPosition() const**

Returns the x, y position of the joystick event.

**wxJoystickEvent::GetZPosition****int GetZPosition() const**

Returns the z position of the joystick event.

**wxJoystickEvent::IsButton****bool IsButton() const**

Returns true if this was a button up or down event (*not* 'is any button down?').

**wxJoystickEvent::IsMove****bool IsMove() const**

Returns true if this was an x, y move event.

**wxJoystickEvent::IsZMove****bool IsZMove() const**

Returns true if this was a z move event.

**wxKeyEvent**

This event class contains information about keypress (character) events.

Notice that there are three different kinds of keyboard events in wxWidgets: key down and up events and char events. The difference between the first two is clear - the first corresponds to a key press and the second to a key release - otherwise they are identical. Just note that if the key is maintained in a pressed state you will typically get a lot of (automatically generated) down events but only one up so it is wrong to assume that there is one up event corresponding to each down one.

Both key events provide untranslated key codes while the char event carries the translated one. The untranslated code for alphanumeric keys is always an upper case value. For the other keys it is one of `WXK_XXX` values from the *keycodes table* (p. 1993). The translated key is, in general, the character the user expects to appear as the result of the key combination when typing the text into a text entry zone, for example.

A few examples to clarify this (all assume that CAPS LOCK is unpressed and the standard US keyboard): when the 'A' key is pressed, the key down event key code is equal to `ASCII A == 65`. But the char event key code is `ASCII a == 97`. On the other hand, if you press both SHIFT and 'A' keys simultaneously, the key code in key down event will still be just 'A' while the char event key code parameter will now be 'A' as well.

Although in this simple case it is clear that the correct key code could be found in the key down event handler by checking the value returned by `ShiftDown()` (p. 961), in general you should use `EVT_CHAR` for this as for non-alphanumeric keys the translation is

keyboard-layout dependent and can only be done properly by the system itself.

Another kind of translation is done when the control key is pressed: for example, for CTRL-A key press the key down event still carries the same key code 'a' as usual but the char event will have key code of 1, the ASCII value of this key combination.

You may discover how the other keys on your system behave interactively by running the *text* (p. 2038) wxWidgets sample and pressing some keys in any of the text controls shown in it.

**Note:** If a key down (`EVT_KEY_DOWN`) event is caught and the event handler does not call `event.Skip()` then the corresponding char event (`EVT_CHAR`) will not happen. This is by design and enables the programs that handle both types of events to be a bit simpler.

**Note for Windows programmers:** The key and char events in wxWidgets are similar to but slightly different from Windows `WM_KEYDOWN` and `WM_CHAR` events. In particular, Alt-x combination will generate a char event in wxWidgets (unless it is used as an accelerator).

**Tip:** be sure to call `event.Skip()` for events that you don't process in key event function, otherwise menu shortcuts may cease to work under Windows.

### Derived from

*wxEvt* (p. 572)

### Include files

<wx/event.h>

### Event table macros

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

<b>EVT_KEY_DOWN(func)</b>	Process a <code>wxEVT_KEY_DOWN</code> event (any key has been pressed).
<b>EVT_KEY_UP(func)</b>	Process a <code>wxEVT_KEY_UP</code> event (any key has been released).
<b>EVT_CHAR(func)</b>	Process a <code>wxEVT_CHAR</code> event.

### *wxKeyEvent::m\_altDown*

#### **bool m\_altDown**

**Deprecated:** Please use *GetModifiers* (p. 959) instead!

true if the Alt key is pressed down.

### *wxKeyEvent::m\_controlDown*

**bool m\_controlDown**

**Deprecated:** Please use *GetModifiers* (p. 959) instead!

true if control is pressed down.

**wxKeyEvent::m\_keyCode****long m\_keyCode**

**Deprecated:** Please use *GetKeyCode* (p. 959) instead!

Virtual keycode. See *Keycodes* (p. 1993) for a list of identifiers.

**wxKeyEvent::m\_metaDown****bool m\_metaDown**

**Deprecated:** Please use *GetModifiers* (p. 959) instead!

true if the Meta key is pressed down.

**wxKeyEvent::m\_shiftDown****bool m\_shiftDown**

**Deprecated:** Please use *GetModifiers* (p. 959) instead!

true if shift is pressed down.

**wxKeyEvent::m\_x****int m\_x**

**Deprecated:** Please use *GetX* (p. 960) instead!

X position of the event.

**wxKeyEvent::m\_y****int m\_y**

**Deprecated:** Please use *GetY* (p. 961) instead!

Y position of the event.

**wxKeyEvent::wxKeyEvent****wxKeyEvent(WXTYPE *keyEventType*)**

Constructor. Currently, the only valid event types are `wxEVT_CHAR` and `wxEVT_CHAR_HOOK`.

**wxKeyEvent::AltDown****bool AltDown() const**

Returns true if the Alt key was down at the time of the key event.

Notice that *GetModifiers* (p. 959) is easier to use correctly than this function so you should consider using it in new code.

**wxKeyEvent::CmdDown****bool CmdDown() const**

CMD is a pseudo key which is the same as Control for PC and Unix platforms but the special APPLE (a.k.a as COMMAND) key under Macs: it makes often sense to use it instead of, say, *ControlDown()* because Cmd key is used for the same thing under Mac as Ctrl elsewhere (but Ctrl still exists, just not used for this purpose under Mac). So for non-Mac platforms this is the same as *ControlDown()* (p. 959) and under Mac this is the same as *MetaDown()* (p. 961).

**wxKeyEvent::ControlDown****bool ControlDown() const**

Returns true if the control key was down at the time of the key event.

Notice that *GetModifiers* (p. 959) is easier to use correctly than this function so you should consider using it in new code.

**wxKeyEvent::GetKeyCode****int GetKeyCode() const**

Returns the virtual key code. ASCII events return normal ASCII values, while non-ASCII events return values such as **WXK\_LEFT** for the left cursor key. See *Keycodes* (p. 1993) for a full list of the virtual key codes.

Note that in Unicode build, the returned value is meaningful only if the user entered a character that can be represented in current locale's default charset. You can obtain the corresponding Unicode character using *GetUnicodeKey* (p. 960).

**wxKeyEvent::GetModifiers****int GetModifiers() const**

Return the bitmask of modifier keys which were pressed when this event happened. See *key modifier constants* (p. 1995) for the full list of modifiers.

Notice that this function is easier to use correctly than, for example, *ControlDown* (p. 959) because when using the latter you also have to remember to test that none of the other modifiers is pressed:

```
if ( ControlDown() && !AltDown() && !ShiftDown() && !MetaDown() )  
    ... handle Ctrl-XXX ...
```

and forgetting to do it can result in serious program bugs (e.g. program not working with European keyboard layout where ALTGR key which is seen by the program as combination of CTRL and ALT is used). On the other hand, you can simply write

```
if ( GetModifiers() == wxMOD_CONTROL )  
    ... handle Ctrl-XXX ...
```

with this function.

### **wxKeyEvent::GetPosition**

**wxPoint GetPosition() const**

**void GetPosition(long \*x, long \*y) const**

Obtains the position (in client coordinates) at which the key was pressed.

### **wxKeyEvent::GetRawKeyCode**

**wxUint32 GetRawKeyCode() const**

Returns the raw key code for this event. This is a platform-dependent scan code which should only be used in advanced applications.

**NB:** Currently the raw key codes are not supported by all ports, use `#ifdef wxHAS_RAW_KEY_CODES` to determine if this feature is available.

### **wxKeyEvent::GetRawKeyFlags**

**wxUint32 GetRawKeyFlags() const**

Returns the low level key flags for this event. The flags are platform-dependent and should only be used in advanced applications.

**NB:** Currently the raw key flags are not supported by all ports, use `#ifdef wxHAS_RAW_KEY_CODES` to determine if this feature is available.

### **wxKeyEvent::GetUnicodeKey**

**wxChar GetUnicodeKey() const**

Returns the Unicode character corresponding to this key event.

This function is only available in Unicode build, i.e. when `wxUSE_UNICODE` is 1.

### **wxKeyEvent::GetX**

**long GetX() const**



Returns the X position (in client coordinates) of the event.

### **wxKeyEvent::GetY**

**long GetY() const**

Returns the Y (in client coordinates) position of the event.

### **wxKeyEvent::HasModifiers**

**bool HasModifiers() const**

Returns true if either CTRL or ALT keys was down at the time of the key event. Note that this function does not take into account neither SHIFT nor META key states (the reason for ignoring the latter is that it is common for NUMLOCK key to be configured as META under X but the key presses even while NUMLOCK is on should be still processed normally).

### **wxKeyEvent::MetaDown**

**bool MetaDown() const**

Returns true if the Meta key was down at the time of the key event.

Notice that *GetModifiers* (p. 959) is easier to use correctly than this function so you should consider using it in new code.

### **wxKeyEvent::ShiftDown**

**bool ShiftDown() const**

Returns true if the shift key was down at the time of the key event.

Notice that *GetModifiers* (p. 959) is easier to use correctly than this function so you should consider using it in new code.

## **wxLayoutAlgorithm**

*wxLayoutAlgorithm* implements layout of subwindows in MDI or SDI frames. It sends a *wxCalculateLayoutEvent* event to children of the frame, asking them for information about their size. For MDI parent frames, the algorithm allocates the remaining space to the MDI client window (which contains the MDI child frames). For SDI (normal) frames, a 'main' window is specified as taking up the remaining space.

Because the event system is used, this technique can be applied to any windows, which are not necessarily 'aware' of the layout classes (no virtual functions in *wxWindow* refer to *wxLayoutAlgorithm* or its events). However, you may wish to use *wxSashLayoutWindow* (p. 1394) for your subwindows since this class provides handlers for the required events, and accessors to specify the desired size of the window. The sash behaviour in the base class can be used, optionally, to make the windows user-resizable.

`wxLayoutAlgorithm` is typically used in IDE (integrated development environment) applications, where there are several resizable windows in addition to the MDI client window, or other primary editing window. Resizable windows might include toolbars, a project window, and a window for displaying error and warning messages.

When a window receives an `OnCalculateLayout` event, it should call `SetRect` in the given event object, to be the old supplied rectangle minus whatever space the window takes up. It should also set its own size accordingly. `wxSashLayoutWindow::OnCalculateLayout` generates an `OnQueryLayoutInfo` event which it sends to itself to determine the orientation, alignment and size of the window, which it gets from internal member variables set by the application.

The algorithm works by starting off with a rectangle equal to the whole frame client area. It iterates through the frame children, generating `OnCalculateLayout` events which subtract the window size and return the remaining rectangle for the next window to process. It is assumed (by `wxSashLayoutWindow::OnCalculateLayout`) that a window stretches the full dimension of the frame client, according to the orientation it specifies. For example, a horizontal window will stretch the full width of the remaining portion of the frame client area. In the other orientation, the window will be fixed to whatever size was specified by `OnQueryLayoutInfo`. An alignment setting will make the window 'stick' to the left, top, right or bottom of the remaining client area. This scheme implies that order of window creation is important. Say you wish to have an extra toolbar at the top of the frame, a project window to the left of the MDI client window, and an output window above the status bar. You should therefore create the windows in this order: toolbar, output window, project window. This ensures that the toolbar and output window take up space at the top and bottom, and then the remaining height in-between is used for the project window.

`wxLayoutAlgorithm` is quite independent of the way in which `OnCalculateLayout` chooses to interpret a window's size and alignment. Therefore you could implement a different window class with a new `OnCalculateLayout` event handler, that has a more sophisticated way of laying out the windows. It might allow specification of whether stretching occurs in the specified orientation, for example, rather than always assuming stretching. (This could, and probably should, be added to the existing implementation).

*Note:* `wxLayoutAlgorithm` has nothing to do with `wxLayoutConstraints`. It is an alternative way of specifying layouts for which the normal constraint system is unsuitable.

### **Derived from**

`wxObject` (p. 1148)

### **Include files**

<wx/laywin.h>

### **Event handling**

The algorithm object does not respond to events, but itself generates the following events in order to calculate window sizes.

<b>EVT_QUERY_LAYOUT_INFO(func)</b>	Process a <code>wxEVT_QUERY_LAYOUT_INFO</code> event, to get size, orientation and alignment from a window. See <code>wxQueryLayoutInfoEvent</code>
------------------------------------	---

(p. 1238).

**EVT\_CALCULATE\_LAYOUT(func)**

Process a `wxEVT_CALCULATE_LAYOUT` event, which asks the window to take a 'bite' out of a rectangle provided by the algorithm. See *wxCalculateLayoutEvent* (p. 167).

**Data types**

```
enum wxLayoutOrientation {  
    wxLAYOUT_HORIZONTAL,  
    wxLAYOUT_VERTICAL  
};  
  
enum wxLayoutAlignment {  
    wxLAYOUT_NONE,  
    wxLAYOUT_TOP,  
    wxLAYOUT_LEFT,  
    wxLAYOUT_RIGHT,  
    wxLAYOUT_BOTTOM,  
};
```

**See also**

*wxSashEvent* (p. 1392), *wxSashLayoutWindow* (p. 1394), *Event handling overview* (p. 2077)

*wxCalculateLayoutEvent* (p. 167), *wxQueryLayoutInfoEvent* (p. 1238), *wxSashLayoutWindow* (p. 1394), *wxSashWindow* (p. 1397)

**wxLayoutAlgorithm::wxLayoutAlgorithm**

**wxLayoutAlgorithm()**

Default constructor.

**wxLayoutAlgorithm::~~wxLayoutAlgorithm**

**~wxLayoutAlgorithm()**

Destructor.

**wxLayoutAlgorithm::LayoutFrame**

**bool LayoutFrame(wxFrame\* frame, wxWindow\* mainWindow = NULL) const**

Lays out the children of a normal frame. *mainWindow* is set to occupy the remaining space.

This function simply calls `wxLayoutAlgorithm::LayoutWindow` (p. 964).

### **wxLayoutAlgorithm::LayoutMDIFrame**

**bool LayoutMDIFrame(wxMDIParentFrame\* frame, wxRect\* rect = NULL) const**

Lays out the children of an MDI parent frame. If *rect* is non-NULL, the given rectangle will be used as a starting point instead of the frame's client area.

The MDI client window is set to occupy the remaining space.

### **wxLayoutAlgorithm::LayoutWindow**

**bool LayoutWindow(wxWindow\* parent, wxWindow\* mainWindow = NULL) const**

Lays out the children of a normal frame or other window.

*mainWindow* is set to occupy the remaining space. If this is not specified, then the last window that responds to a calculate layout event in query mode will get the remaining space (that is, a non-query `OnCalculateLayout` event will not be sent to this window and the window will be set to the remaining size).

## **wxLayoutConstraints**

**Note:** constraints are now deprecated and you should use *sizers* (p. 2098) instead.

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.

**wxPerl note:** In wxPerl the constraints are accessed as `constraint = Wx::LayoutConstraints->new();`  
`constraint->centreX->AsIs();`  
`constraint->centreY->Unconstrained();`

### Derived from

*wxObject* (p. 1148)

### Include files

`<wx/layout.h>`

### See also

*Overview and examples* (p. 2094), *wxIndividualLayoutConstraint* (p. 938), *wxWindow::SetConstraints* (p. 1837)

## **wxLayoutConstraints::wxLayoutConstraints**

**wxLayoutConstraints()**

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.

## **wxList<T>**

The `wxList<T>` class provides linked list functionality. It has been written to be type safe and to provide the full API of the STL `std::list` container and should be used like it. The exception is that `wxList<T>` actually stores pointers and therefore its iterators return pointers and not references to the actual objects in the list (see example below). In other words *value\_type* is defined as *T\**.

Unfortunately, the new `wxList<T>` class requires that you declare and define each `wxList<T>` class in your program. This is done with `WX_DECLARE_LIST` and `WX_DEFINE_LIST` macros (see example). We hope that we'll be able to provide a proper template class providing both the STL `std::list` and the old `wxList` API in the future.

Please refer to the STL `std::list` documentation for further information on how to use the class. Below we documented both the supported STL and the legacy API that originated from the old `wxList` class and which can still be used alternatively for the the same class.

Note that if you compile `wxWidgets` in STL mode (`wxUSE_STL` defined as 1) then `wxList<T>` will actually derive from `std::list` and just add a legacy compatibility layer for the old `wxList` class.

### **Example**

```
// this part might be in a header or source (.cpp) file
class MyListElement
```

```
{
    ... // whatever
};

// this macro declares and partly implements MyList class
WX_DECLARE_LIST(MyListElement, MyList);

...

// the only requirement for the rest is to be AFTER the full
// declaration of
// MyListElement (for WX_DECLARE_LIST forward declaration is
// enough), but
// usually it will be found in the source file and not in the header

#include <wx/listimpl.cpp>
WX_DEFINE_LIST(MyList);

MyList list;
MyListElement element;
list.Append(&element);    // ok
list.Append(17);          // error: incorrect type

// let's iterate over the list in STL syntax
MyList::iterator iter;
for (iter = list.begin(); iter != list.end(); ++iter)
{
    MyListElement *current = *iter;

    ...process the current element...
}

// the same with the legacy API from the old wxList class
MyList::compatibility_iterator node = list.GetFirst();
while (node)
{
    MyListElement *current = node->GetData();

    ...process the current element...

    node = node->GetNext();
}
```

For compatibility with previous versions `wxList` and `wxStringList` classes are still defined, but their usage is deprecated and they will disappear in the future versions completely. The use of the latter is especially discouraged as it is not only unsafe but is also much less efficient than `wxArrayString` (p. 83) class.

### Include files

`<wx/list.h>`

### Library

*wxBase* (p. 15)

**See also**

*wxArray* (p. 71)

**wxList<T>::wxList<T>**

**wxList<T>()**

**wxList<T>(size\_t count, T \*elements[])**

Constructors.

**wxList<T>::~~wxList<T>**

**~wxList<T>()**

Destroys the list, but does not delete the objects stored in the list unless you called `DeleteContents(true)`.

**wxList<T>::Append**

**wxList<T>::compatibility\_iterator Append(T \*object)**

Appends the pointer to *object* to the list.

**wxList<T>::Clear**

**void Clear()**

Clears the list, but does not delete the objects stored in the list unless you called `DeleteContents(true)`.

**wxList<T>::DeleteContents**

**void DeleteContents(bool destroy)**

If *destroy* is `true`, instructs the list to call *delete* on objects stored in the list whenever they are removed. The default is `false`.

**wxList<T>::DeleteNode**

**bool DeleteNode(const compatibility\_iterator&iter)**

Deletes the given element referred to by *iter* from the list, returning `true` if successful.

**wxList<T>::DeleteObject**



**bool DeleteObject(T \*object)**

Finds the given *object* and removes it from the list, returning `true` if successful. The application must delete the actual object separately.

**wxList<T>::Erase****void Erase(const compatibility\_iterator&iter)**

Removes element referred to be *iter*.

**wxList<T>::Find****wxList<T>::compatibility\_iterator Find(T \* object) const**

Returns the iterator referring to *object* or `NULL` if none found.

**wxList<T>::GetCount****size\_t GetCount() const**

Returns the number of elements in the list.

**wxList<T>::GetFirst****wxList<T>::compatibility\_iterator GetFirst() const**

Returns the first iterator in the list (`NULL` if the list is empty).

**wxList<T>::GetLast****wxList<T>::compatibility\_iterator GetLast() const**

Returns the last iterator in the list (`NULL` if the list is empty).

**wxList<T>::IndexOf****int IndexOf(T\* obj) const**

Returns the index of *obj* within the list or `wxNOT_FOUND` if *obj* is not found in the list.

**wxList<T>::Insert****wxList<T>::compatibility\_iterator Insert(T \*object)**

Insert object at the front of list.

**wxList<T>::compatibility\_iterator Insert(size\_t position, T \*object)**

Insert object before *position*, i.e. the index of the new item in the list will be equal to

*position*. *position* should be less than or equal to *GetCount* (p. 969); if it is equal to it, this is the same as calling *Append* (p. 968).

**wxList<T>::compatibility\_iterator Insert(compatibility\_iterator iter, T \*object)**

Inserts the object before the object referred to be *iter*.

**wxList<T>::IsEmpty**

**bool IsEmpty() const**

Returns *true* if the list is empty, *false* otherwise.

**wxList<T>::Item**

**wxList<T>::compatibility\_iterator Item(size\_t index) const**

Returns the iterator referring to the object at the given *index* in the list.

**wxList<T>::Member**

**wxList<T>::compatibility\_iterator Member(T \* object) const**

**NB:** This function is deprecated, use *Find* (p. 969) instead.

**wxList<T>::Nth**

**wxList<T>::compatibility\_iterator Nth(int n) const**

**NB:** This function is deprecated, use *Item* (p. 970) instead.

Returns the *nth* node in the list, indexing from zero (NULL if the list is empty or the *nth* node could not be found).

**wxList<T>::Number**

**int Number() const**

**NB:** This function is deprecated, use *GetCount* (p. 969) instead.

Returns the number of elements in the list.

**wxList<T>::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.

### **wxList<T>::assign**

**void assign(const\_iterator first, const const\_iterator& last)**

**void assign(size\_type n, const\_reference v = value\_type())**

### **wxList<T>::back**

**reference back()**

**const\_reference back() const**

Returns the last item of the list.

### **wxList<T>::begin**

**iterator begin()**

**const\_iterator begin() const**

Returns a (const) iterator pointing to the beginning of the list.

### **wxList<T>::clear**

**void clear()**

Removes all items from the list.

### **wxList<T>::empty**

**bool empty() const**

Returns *true* if the list is empty.

### **wxList<T>::end**

**iterator end()**

**const\_iterator end() const**

Returns a (const) iterator pointing at the end of the list.

### **wxList<T>::erase**

**iterator erase(const iterator& it)**

Erases the item pointed to by *it*.

**iterator erase(const iterator& first, const iterator& last)**

Erases the items from *first* to *last*.

**wxList<T>::front**

**reference** front()

**const\_reference** front() const

Returns the first item in the list.

**wxList<T>::insert**

**iterator** insert(const iterator& *it*, const\_reference *v* = value\_type())

**void** insert(const iterator& *it*, size\_type *n*, const\_reference *v* = value\_type())

**void** insert(const iterator& *it*, const\_iterator *first*, const const\_iterator& *last*)

Inserts an item (or several) at the given position.

**wxList<T>::max\_size**

**size\_type** max\_size() const

Returns the largest possible size of the list.

**wxList<T>::pop\_back**

**void** pop\_back()

Removes the list item.

**wxList<T>::pop\_front**

**void** pop\_front()

Removes the first item.

**wxList<T>::push\_back**

**void** push\_back(const\_reference *v* = value\_type())

Adds an item to end of the list.

**wxList<T>::push\_front**

**void** push\_front(const\_reference *v* = value\_type())

Adds an item to the front of the list.

**wxList<T>::rbegin****reverse\_iterator rbegin()****const\_reverse\_iterator rbegin() const**

Returns a (const) reverse iterator pointer to the beginning of the reversed list.

**wxList<T>::remove****void remove(const\_reference v)**

Removes an item from the list.

**wxList<T>::rend****reverse\_iterator rend()****const\_reverse\_iterator rend() const**

Returns a (const) reverse iterator pointer to the end of the reversed list.

**wxList<T>::resize****void resize(size\_type n, value\_type v = value\_type())**

Resizes the list. If the the list is enlarges items with the value *v* are appended to the list.

**wxList<T>::reverse****void reverse()**

Reverses the list.

**wxList<T>::size****size\_type size() const**

Returns the size of the list.

**wxList<T>::splice****void splice(const iterator& it, wxList<T>& l)****void splice(const iterator& it, wxList<T>& l, const iterator& first)****void splice(const iterator& it, wxList<T>& l, const iterator& first, const iterator& last)**

Moves part of the list into another list, starting from *first* and ending at *last* if specified.

## wxListbook

wxListbook is a class similar to *wxNotebook* (p. 1135) but which uses a *wxListCtrl* (p. 980) to show the labels instead of the tabs.

There is no documentation for this class yet but its usage is identical to *wxNotebook* (except for the features clearly related to tabs only), so please refer to that class documentation for now. You can also use the *notebook sample* (p. 2036) to see wxListbook in action.

### Derived from

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/listbook.h>

### Window styles

<b>wxLB_DEFAULT</b>	Choose the default location for the labels depending on the current platform (left everywhere except Mac where it is top).
<b>wxLB_TOP</b>	Place labels above the page area.
<b>wxLB_LEFT</b>	Place labels on the left side.
<b>wxLB_RIGHT</b>	Place labels on the right side.
<b>wxLB_BOTTOM</b>	Place labels below the page area.

### See also

*wxBookCtrl* (p. 2127), *wxNotebook* (p. 1135), *notebook sample* (p. 2036)

## wxListBox

A listbox is used to select one or more of a list of strings. The strings are displayed in a scrolling box, with the selected string(s) marked in reverse video. A listbox can be single selection (if an item is selected, the previous selection is removed) or multiple selection (clicking an item toggles the item on or off independently of other selections).

List box elements are numbered from zero. Their number may be limited under some platforms.

A listbox callback gets an event `wxEVT_COMMAND_LISTBOX_SELECTED` for single clicks, and `wxEVT_COMMAND_LISTBOX_DOUBLECLICKED` for double clicks.

### Derived from

*wxControlWithItems* (p. 286)

*wxControl* (p. 285)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### Include files

<wx/listbox.h>

### Window styles

<b>wxLB_SINGLE</b>	Single-selection list.
<b>wxLB_MULTIPLE</b>	Multiple-selection list: the user can toggle multiple items on and off.
<b>wxLB_EXTENDED</b>	Extended-selection list: the user can select multiple items using the SHIFT key and the mouse or special key combinations.
<b>wxLB_HSCROLL</b>	Create horizontal scrollbar if contents are too wide (Windows only).
<b>wxLB_ALWAYS_SB</b>	Always show a vertical scrollbar.
<b>wxLB_NEEDED_SB</b>	Only create a vertical scrollbar if needed.
<b>wxLB_SORT</b>	The listbox contents are sorted in alphabetical order.

Note that `wxLB_SINGLE`, `wxLB_MULTIPLE` and `wxLB_EXTENDED` styles are mutually exclusive and you can specify at most one of them (single selection is the default).

See also *window styles overview* (p. 2089).

### Event handling

<b>EVT_LISTBOX(id, func)</b>	Process a <code>wxEVT_COMMAND_LISTBOX_SELECTED</code> event, when an item on the list is selected or the selection changes.
<b>EVT_LISTBOX_DCLICK(id, func)</b>	Process a <code>wxEVT_COMMAND_LISTBOX_DOUBLECLICKED</code> event, when the listbox is double-clicked.

### See also

*wxChoice* (p. 186), *wxComboBox* (p. 225), *wxListCtrl* (p. 980), *wxCommandEvent* (p. 250)

### **wxListBox::wxListBox**

**wxListBox()**

Default constructor.

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

```
wxListBox(wxWindow* parent, wxWindowID id, const wxPoint& pos, const wxSize&  
size, const wxStringArray& choices, long style = 0, const wxValidator& validator =  
wxDefaultValidator, const wxString& name = "listBox")
```

Constructor, creating and showing a list box.

**Parameters**

*parent*

Parent window. Must not be NULL.

*id*

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

*pos*

Window position.

*size*

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

*n*

Number of strings with which to initialise the control.

*choices*

An array of strings with which to initialise the control.

*style*

Window style. See *wxListBox* (p. 974).

*validator*

Window validator.

*name*

Window name.

**See also**



*wxListBox::Create* (p. 977), *wxValidator* (p. 1767)

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

**wxPerl note:** In *wxPerl* there is just an array reference in place of `nand choices`.

### **wxListBox::~~wxListBox**

**void ~wxListBox()**

Destructor, destroying the list box.

### **wxListBox::Create**

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

**bool Create(wxWindow\* parent, wxWindowID id, const wxPoint& pos, const wxSize& size, const wxArrayString& choices, long style = 0, const wxValidator& validator = wxDefaultValidator, const wxString& name = "listBox")**

Creates the listbox for two-step construction. See *wxListBox::wxListBox* (p. 975) for further details.

### **wxListBox::Deselect**

**void Deselect(int n)**

Deselects an item in the list box.

#### **Parameters**

*n*

The zero-based item to deselect.

#### **Remarks**

This applies to multiple selection listboxes only.

### **wxListBox::GetSelections**

**int GetSelections(wxArrayInt& selections) const**

Fill an array of ints with the positions of the currently selected items.

#### **Parameters**

*selections*

A reference to an `wxArrayInt` instance that is used to store the result of the query.

**Return value**

The number of selections.

**Remarks**

Use this with a multiple selection listbox.

**See also**

`wxControlWithItems::GetSelection` (p. 289), `wxControlWithItems::GetStringSelection` (p. 290), `wxControlWithItems::SetSelection` (p. 292)

**wxPython note:** The wxPython version of this method takes no parameters and returns a tuple of the selected items.

**wxPerl note:** In wxPerl this method takes no parameters and return the selected items as a list.

**wxListBox::InsertItems**

**void InsertItems(int *nItems*, const wxString \**items*, unsigned int *pos*)**

**void InsertItems(const wxArrayString& *nItems*, unsigned int *pos*)**

Insert the given number of strings before the specified position.

**Parameters**

*nItems*

Number of items in the array *items*

*items*

Labels of items to be inserted

*pos*

Position before which to insert the items: for example, if *pos* is 0 the items will be inserted in the beginning of the listbox

**wxPython note:** The first two parameters are collapsed into a single parameter for wxPython, which is a list of strings.

**wxPerl note:** In wxPerl there is just an array reference in place of *nItems* and *items*.

**wxListBox::HitTest**

**int HitTest(const wxPoint& *point*) const**

Returns the item located at *point*, or `wxNOT_FOUND` if there is no item located at *point*.

This function is new since wxWidgets version 2.7.0. It is currently implemented for wxMSW, wxMac and wxGTK2 ports.

**Parameters**

*point*

Point of item (in client coordinates) to obtain

**Return value**

Item located at *point*, or `wxNOT_FOUND` if unimplemented or the item does not exist.

**wxListBox::IsSelected**

**bool IsSelected(int *n*) const**

Determines whether an item is selected.

**Parameters**

*n*

The zero-based item index.

**Return value**

true if the given item is selected, false otherwise.

**wxListBox::Set**

**void Set(int *n*, const wxString\* *choices*, void \*\**clientData* = NULL)**

**void Set(const wxArrayString& *choices*, void \*\**clientData* = NULL)**

Clears the list box and adds the given strings to it.

**Parameters**

*n*

The number of strings to set.

*choices*

An array of strings to set.

*clientData*

Options array of client data pointers

**Remarks**

You may free the array from the calling program after this function has been called.

**wxListBox::SetFirstItem****void SetFirstItem(int *n*)****void SetFirstItem(const wxString& *string*)**

Set the specified item to be the first visible item.

**Parameters***n*

The zero-based item index.

*string*

The string that should be visible.

**wxListCtrl**

A list control presents lists in a number of formats: list view, report view, icon view and small icon view. In any case, elements are numbered from zero. For all these modes, the items are stored in the control and must be added to it using *InsertItem* (p. 992) method.

A special case of report view quite different from the other modes of the list control is a virtual control in which the items data (including text, images and attributes) is managed by the main program and is requested by the control itself only when needed which allows to have controls with millions of items without consuming much memory. To use virtual list control you must use *SetItemCount* (p. 997) first and overload at least *OnGetItemText* (p. 994) (and optionally *OnGetItemImage* (p. 993) or *OnGetItemColumnImage* (p. 993) and *OnGetItemAttr* (p. 993)) to return the information about the items when the control requests it. Virtual list control can be used as a normal one except that no operations which can take time proportional to the number of items in the control happen -- this is required to allow having a practically infinite number of items. For example, in a multiple selection virtual list control, the selections won't be sent when many items are selected at once because this could mean iterating over all the items.

Using many of wxListCtrl features is shown in the *corresponding sample* (p. 2036).

To intercept events from a list control, use the event table macros described in *wxListEvent* (p. 1000).

**Mac Note:** Starting with 2.8, wxListCtrl uses a native implementation for report mode, and uses a generic implementation for other modes. You can use the generic implementation for report mode as well by setting the `mac.listctrl.always_use_generic` *wxSystemOption* (p. 1590) to 1.

**Derived from***wxControl* (p. 285)*wxWindow* (p. 1795)*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### Include files

<wx/listctrl.h>

### Window styles

<b>wxLC_LIST</b>	Multicolumn list view, with optional small icons. Columns are computed automatically, i.e. you don't set columns as in <b>wxLC_REPORT</b> . In other words, the list wraps, unlike a <code>wxListBox</code> .
<b>wxLC_REPORT</b>	Single or multicolumn report view, with optional header.
<b>wxLC_VIRTUAL</b>	The application provides items text on demand. May only be used with <b>wxLC_REPORT</b> .
<b>wxLC_ICON</b>	Large icon view, with optional labels.
<b>wxLC_SMALL_ICON</b>	Small icon view, with optional labels.
<b>wxLC_ALIGN_TOP</b>	Icons align to the top. Win32 default, Win32 only.
<b>wxLC_ALIGN_LEFT</b>	Icons align to the left.
<b>wxLC_AUTOARRANGE</b>	Icons arrange themselves. Win32 only.
<b>wxLC_EDIT_LABELS</b>	Labels are editable: the application will be notified when editing starts.
<b>wxLC_NO_HEADER</b>	No header in report mode.
<b>wxLC_SINGLE_SEL</b>	Single selection (default is multiple).
<b>wxLC_SORT_ASCENDING</b>	Sort in ascending order (must still supply a comparison callback in <code>SortItems</code> ).
<b>wxLC_SORT_DESCENDING</b>	Sort in descending order (must still supply a comparison callback in <code>SortItems</code> ).
<b>wxLC_HRULES</b>	Draws light horizontal rules between rows in report mode.
<b>wxLC_VRULES</b>	Draws light vertical rules between columns in report mode.

See also *window styles overview* (p. 2089).

### Event handling

To process input from a list control, use these event handler macros to direct input to member functions that take a *wxListEvent* (p. 1000) argument.

- EVT\_LIST\_BEGIN\_DRAG(id, func)** Begin dragging with the left mouse button.
- EVT\_LIST\_BEGIN\_RDRAG(id, func)** Begin dragging with the right mouse button.
- EVT\_LIST\_BEGIN\_LABEL\_EDIT(id, func)** Begin editing a label. This can be prevented by calling *Veto()* (p. 1147).
- EVT\_LIST\_END\_LABEL\_EDIT(id, func)** Finish editing a label. This can be prevented by calling *Veto()* (p. 1147).
- EVT\_LIST\_DELETE\_ITEM(id, func)** Delete an item.
- EVT\_LIST\_DELETE\_ALL\_ITEMS(id, func)** Delete all items.
- EVT\_LIST\_ITEM\_SELECTED(id, func)** The item has been selected.
- EVT\_LIST\_ITEM\_DESELECTED(id, func)** The item has been deselected.
- EVT\_LIST\_ITEM\_ACTIVATED(id, func)** The item has been activated (ENTER or double click).
- EVT\_LIST\_ITEM\_FOCUSED(id, func)** The currently focused item has changed.
- EVT\_LIST\_ITEM\_MIDDLE\_CLICK(id, func)** The middle mouse button has been clicked on an item.
- EVT\_LIST\_ITEM\_RIGHT\_CLICK(id, func)** The right mouse button has been clicked on an item.
- EVT\_LIST\_KEY\_DOWN(id, func)** A key has been pressed.
- EVT\_LIST\_INSERT\_ITEM(id, func)** An item has been inserted.
- EVT\_LIST\_COL\_CLICK(id, func)** A column (**m\_col**) has been left-clicked.
- EVT\_LIST\_COL\_RIGHT\_CLICK(id, func)** A column (**m\_col**) has been right-clicked.
- EVT\_LIST\_COL\_BEGIN\_DRAG(id, func)** The user started resizing a column - can be vetoed.
- EVT\_LIST\_COL\_DRAGGING(id, func)** The divider between columns is being dragged.
- EVT\_LIST\_COL\_END\_DRAG(id, func)** A column has been resized by the user.
- EVT\_LIST\_CACHE\_HINT(id, func)** Prepare cache for a virtual list control

**See also**

*wxListCtrl* overview (p. 2126), *wxListView* (p. 1008), *wxListBox* (p. 974), *wxTreeCtrl* (p. 1728), *wxImageList* (p. 933), *wxListEvent* (p. 1000), *wxListItem* (p. 1003)

**wxListCtrl::wxListCtrl**

**wxListCtrl()**

Default constructor.

```
wxListCtrl(wxWindow* parent, wxWindowID id, const wxPoint& pos =  
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxLC_ICON,  
const wxValidator& validator = wxDefaultValidator, const wxString& name =  
wxListCtrlNameStr)
```

Constructor, creating and showing a list control.

### Parameters

*parent*

Parent window. Must not be NULL.

*id*

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

*pos*

Window position.

*size*

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

*style*

Window style. See *wxListCtrl* (p. 980).

*validator*

Window validator.

*name*

Window name.

### See also

*wxListCtrl::Create* (p. 984), *wxValidator* (p. 1767)

### **wxListCtrl::~wxListCtrl**

```
void ~wxListCtrl()
```

Destructor, destroying the list control.

### **wxListCtrl::Arrange**

```
bool Arrange(int flag = wxLIST_ALIGN_DEFAULT)
```

Arranges the items in icon or small icon view. This only has effect on Win32. *flag* is one of:

`wxLIST_ALIGN_DEFAULT` Default alignment.

`wxLIST_ALIGN_LEFT` Align to the left side of the control.

`wxLIST_ALIGN_TOP` Align to the top side of the control.

`wxLIST_ALIGN_SNAP_TO_GRID` Snap to grid.

### **`wxListCtrl::AssignImageList`**

**`void AssignImageList(wxImageList* imageList, int which)`**

Sets the image list associated with the control and takes ownership of it (i.e. the control will, unlike when using `SetImageList`, delete the list when destroyed). *which* is one of `wxIMAGE_LIST_NORMAL`, `wxIMAGE_LIST_SMALL`, `wxIMAGE_LIST_STATE` (the last is unimplemented).

#### **See also**

*`wxListCtrl::SetImageList`* (p. 995)

### **`wxListCtrl::ClearAll`**

**`void ClearAll()`**

Deletes all items and all columns.

### **`wxListCtrl::Create`**

**`bool Create(wxWindow* parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxLC_ICON, const wxValidator& validator = wxDefaultValidator, const wxString& name = wxListCtrlNameStr)`**

Creates the list control. See *`wxListCtrl::wxListCtrl`* (p. 982) for further details.

### **`wxListCtrl::DeleteAllItems`**

**`bool DeleteAllItems()`**

Deletes all items in the list control.

**NB:** This function does *not* send the `wxEVT_COMMAND_LIST_DELETE_ITEM` event because deleting many items from the control would be too slow then (unlike *`DeleteItem`* (p. 985)).

### **`wxListCtrl::DeleteColumn`**

**`bool DeleteColumn(int col)`**

Deletes a column.



**wxListCtrl::DeleteItem****bool DeleteItem(long item)**

Deletes the specified item. This function sends the `wxEVT_COMMAND_LIST_DELETE_ITEM` event for the item being deleted.

See also: *DeleteAllItems* (p. 984)

**wxListCtrl::EditLabel****void EditLabel(long item)**

Starts editing the label of the given item. This function generates a `EVT_LIST_BEGIN_LABEL_EDIT` event which can be vetoed so that no text control will appear for in-place editing.

If the user changed the label (i.e. s/he does not press ESC or leave the text control without changes, a `EVT_LIST_END_LABEL_EDIT` event will be sent which can be vetoed as well.

**wxListCtrl::EnsureVisible****bool EnsureVisible(long item)**

Ensures this item is visible.

**wxListCtrl::FindItem****long FindItem(long start, const wxString& str, const bool partial = false)**

Find an item whose label matches this string, starting from *start* or the beginning if *start* is -1. The string comparison is case insensitive. If *partial* is true then this method will look for items which begin with *str*.

**long FindItem(long start, long data)**

Find an item whose data matches this data, starting from *start* or the beginning if 'start' is -1.

**long FindItem(long start, const wxPoint& pt, int direction)**

Find an item nearest this position in the specified direction, starting from *start* or the beginning if *start* is -1.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

**FindItem(start, str, partial=false)**

**FindItemData(start, data)**

**FindItemAtPos(start, point, direction)**

**wxPerl note:** In wxPerl there are three methods instead of a single overloaded method:

**FindItem( start, str, partial = false )**

**FindItemData( start, data )**

**FindItemAtPos( start, point, direction )**

### **wxListCtrl::GetColumn**

**bool GetColumn(int col, wxListItem& item) const**

Gets information about this column. See *wxListCtrl::SetItem* (p. 995) for more information.

**wxPerl note:** In wxPerl this method takes only the **col** parameter and returns a *Wx::ListItem* ( or *undef* ).

### **wxListCtrl::GetColumnCount**

**int GetColumnCount() const**

Returns the number of columns.

### **wxListCtrl::GetColumnWidth**

**int GetColumnWidth(int col) const**

Gets the column width (report view only).

### **wxListCtrl::GetCountPerPage**

**int GetCountPerPage() const**

Gets the number of items that can fit vertically in the visible area of the list control (list or report view) or the total number of items in the list control (icon or small icon view).

### **wxListCtrl::GetEditControl**

**wxTextCtrl \* GetEditControl() const**

Returns the edit control being currently used to edit a label. Returns *NULL* if no label is being edited.

**NB:** It is currently only implemented for wxMSW and the generic version, not for the native Mac OS X version.

### **wxListCtrl::GetImageList**

**wxImageList\* GetImageList(int which) const**

Returns the specified image list. *which* may be one of:

**wxIMAGE\_LIST\_NORMAL** The normal (large icon) image list.

**wxIMAGE\_LIST\_SMALL** The small icon image list.

**wxIMAGE\_LIST\_STATE** The user-defined state image list (unimplemented).

### **wxListCtrl::GetItem**

**bool GetItem(wxListItem& *info*) const**

Gets information about the item. See *wxListCtrl::SetItem* (p. 995) for more information.

You must call *info.SetId()* to the ID of item you're interested in before calling this method.

**wxPython note:** The wxPython version of this method takes an integer parameter for the item ID, an optional integer for the column number, and returns the wxListItem object.

**wxPerl note:** In wxPerl this method takes as parameter the **ID** of the item and ( optionally ) the column, and returns a Wx::ListItem object.

### **wxListCtrl::GetItemBackgroundColour**

**wxColour GetItemBackgroundColour(long *item*) const**

Returns the colour for this item. If the item has no specific colour, returns an invalid colour (and not the default background control of the control itself).

**See also**

*GetItemTextColour* (p. 989)

### **wxListCtrl::GetItemCount**

**int GetItemCount() const**

Returns the number of items in the list control.

### **wxListCtrl::GetItemData**

**long GetItemData(long *item*) const**

Gets the application-defined data associated with this item.

### **wxListCtrl::GetItemFont**

**wxFont GetItemFont(long *item*) const**

Returns the item's font.

### **wxListCtrl::GetItemPosition**

**bool GetItemPosition(long item, wxPoint& pos) const**

Returns the position of the item, in icon or small icon view.

**wxPython note:** The wxPython version of this method accepts only the item ID and returns the wxPoint.

**wxPerl note:** In wxPerl this method takes only the **item** parameter and returns a Wx::Point ( or undef ).

**wxListCtrl::GetItemRect****bool GetItemRect(long item, wxRect& rect, int code = wxLIST\_RECT\_BOUNDS) const**

Returns the rectangle representing the item's size and position, in physical coordinates.

*code* is one of wxLIST\_RECT\_BOUNDS, wxLIST\_RECT\_ICON, wxLIST\_RECT\_LABEL.

**wxPython note:** The wxPython version of this method accepts only the item ID and code and returns the wxRect.

**wxPerl note:** In wxPerl this method takes only the **item** parameter and returns a Wx::Rect ( or undef ).

**wxListCtrl::GetSubItemRect****bool GetSubItemRect(long item, long subItem, wxRect& rect, int code = wxLIST\_RECT\_BOUNDS) const**

Returns the rectangle representing the size and position, in physical coordinates, of the given subitem, i.e. the part of the row *item* in the column *subItem*.

This method is only meaningful when the wxListCtrl is in the report mode. If *subItem* parameter is equal to the special value wxLIST\_GETSUBITEMRECT\_WHOLEITEM the return value is the same as for *GetItemRect* (p. 988).

*code* can be one of wxLIST\_RECT\_BOUNDS, wxLIST\_RECT\_ICON or wxLIST\_RECT\_LABEL.

This function is new since wxWidgets version 2.7.0

**wxListCtrl::GetItemSpacing****wxSize GetItemSpacing() const**

Retrieves the spacing between icons in pixels: horizontal spacing is returned as *x* component of the wxSize (p. 1440) object and the vertical spacing as its *y* component.

**wxListCtrl::GetItemState****int GetItemState(long item, long stateMask) const**

Gets the item state. For a list of state flags, see *wxListCtrl::SetItem* (p. 995).

The **stateMask** indicates which state flags are of interest.

### **wxListCtrl::GetItemText**

**wxString GetItemText(long item) const**

Gets the item text for this item.

### **wxListCtrl::GetItemTextColour**

**wxColour GetItemTextColour(long item) const**

Returns the colour for this item. If the item has no specific colour, returns an invalid colour (and not the default foreground control of the control itself as this wouldn't allow distinguishing between items having the same colour as the current control foreground and items with default colour which, hence, have always the same colour as the control).

### **wxListCtrl::GetNextItem**

**long GetNextItem(long item, int geometry = wxLIST\_NEXT\_ALL, int state = wxLIST\_STATE\_DONTCARE) const**

Searches for an item with the given geometry or state, starting from *item* but excluding the *item* itself. If *item* is -1, the first item that matches the specified flags will be returned.

Returns the first item with given state following *item* or -1 if no such item found.

This function may be used to find all selected items in the control like this:

```
long item = -1;
for ( ;; )
{
    item = listctrl->GetNextItem(item,
                                wxLIST_NEXT_ALL,
                                wxLIST_STATE_SELECTED);

    if ( item == -1 )
        break;

    // this item is selected - do whatever is needed with it
    wxLogMessage("Item %ld is selected.", item);
}
```

*geometry* can be one of:

wxLIST_NEXT_ABOVE	Searches for an item above the specified item.
wxLIST_NEXT_ALL	Searches for subsequent item by index.
wxLIST_NEXT_BELOW	Searches for an item below the specified item.
wxLIST_NEXT_LEFT	Searches for an item to the left of the specified item.

`wxLIST_NEXT_RIGHT` Searches for an item to the right of the specified item.

**NB:** this parameter is only supported by wxMSW currently and ignored on other platforms.

*state* can be a bitlist of the following:

`wxLIST_STATE_DONTCARE` Don't care what the state is.

`wxLIST_STATE_DROPHILITED` The item indicates it is a drop target.

`wxLIST_STATE_FOCUSED` The item has the focus.

`wxLIST_STATE_SELECTED` The item is selected.

`wxLIST_STATE_CUT` The item is selected as part of a cut and paste operation.

### **wxListCtrl::GetSelectedItemCount**

**int GetSelectedItemCount() const**

Returns the number of selected items in the list control.

### **wxListCtrl::GetTextColour**

**wxColour GetTextColour() const**

Gets the text colour of the list control.

### **wxListCtrl::GetTopItem**

**long GetTopItem() const**

Gets the index of the topmost visible item when in list or report view.

### **wxListCtrl::GetViewRect**

**wxRect GetViewRect() const**

Returns the rectangle taken by all items in the control. In other words, if the controls client size were equal to the size of this rectangle, no scrollbars would be needed and no free space would be left.

Note that this function only works in the icon and small icon views, not in list or report views (this is a limitation of the native Win32 control).

### **wxListCtrl::HitTest**

**long HitTest(const wxPoint& point, int& flags, long \*ptrSubItem) const**

Determines which item (if any) is at the specified point, giving details in *flags*. Returns index of the item or `wxNOT_FOUND` if no item is at the specified point. *flags* will be a combination of the following flags:

`wxLIST_HITTEST_ABOVE` Above the client area.

`wxLIST_HITTEST_BELOW` Below the client area.

`wxLIST_HITTEST_NOWHERE` In the client area but below the last item.

`wxLIST_HITTEST_ONITEMICON` On the bitmap associated with an item.

`wxLIST_HITTEST_ONITEMLABEL` On the label (string) associated with an item.

`wxLIST_HITTEST_ONITEMRIGHT` In the area to the right of an item.

`wxLIST_HITTEST_ONITEMSTATEICON` On the state icon for a tree view item that is in a user-defined state.

`wxLIST_HITTEST_TOLEFT` To the right of the client area.

`wxLIST_HITTEST_TORIGHT` To the left of the client area.

`wxLIST_HITTEST_ONITEM` Combination of `wxLIST_HITTEST_ONITEMICON`, `wxLIST_HITTEST_ONITEMLABEL`, `wxLIST_HITTEST_ONITEMSTATEICON`.

If *ptrSubItem* is not `NULL` and the `wxListCtrl` is in the report mode the subitem (or column) number will also be provided. This feature is only available in version 2.7.0 or higher and is currently only implemented under `wxMSW` and requires at least `comctl32.dll` of version 4.70 on the host system or the value stored in *ptrSubItem* will be always -1. To compile this feature into `wxWidgets` library you need to have access to `commctrl.h` of version 4.70 that is provided by Microsoft.

**wxPython note:** A tuple of values is returned in the `wxPython` version of this method. The first value is the item id and the second is the flags value mentioned above.

**wxPerl note:** In `wxPerl` this method only takes the **point** parameter and returns a 2-element list ( *item*, *flags* ).

### **wxListCtrl::InsertColumn**

**long InsertColumn(long col, wxListItem& info)**

**long InsertColumn(long col, const wxString& heading, int format = `wxLIST_FORMAT_LEFT`, int width = -1)**

For report view mode (only), inserts a column. For more details, see `wxListCtrl::SetItem` (p. 995).

**wxPython note:** In place of a single overloaded method name, `wxPython` implements the following methods:

**InsertColumn(col, heading, format=`wxLIST_FORMAT_LEFT`, width=-1)**  
Creates a column using a header string only.

**InsertColumnItem(col, item)** Creates a column using a `wxListItem`.

**wxListCtrl::InsertItem****long InsertItem(wxListItem& *info*)**

Inserts an item, returning the index of the new item if successful, -1 otherwise.

**long InsertItem(long *index*, const wxString& *label*)**

Inserts a string item.

**long InsertItem(long *index*, int *imageIndex*)**

Inserts an image item.

**long InsertItem(long *index*, const wxString& *label*, int *imageIndex*)**

Insert an image/string item.

**Parameters***info*

wxListItem object

*index*

Index of the new item, supplied by the application

*label*

String label

*imageIndex*

index into the image list associated with this control and view style

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

**InsertItem(item)**                      Inserts an item using a wxListItem.

**InsertStringItem(index, label)**   Inserts a string item.

**InsertImageItem(index, imageIndex)**   Inserts an image item.

**InsertImageStringItem(index, label, imageIndex)**   Insert an image/string item.

**wxPerl note:** In wxPerl there are four methods instead of a single overloaded method:

**InsertItem( item )**                      Inserts a Wx::ListItem

**InsertStringItem( index, label )** Inserts a string item



**InsertImageItem( index, imageIndex )** Inserts an image item

**InsertImageStringItem( index, label, imageIndex )** Inserts an item with a string and an image

### **wxListCtrl::OnGetItemAttr**

**virtual wxListItemAttr \* OnGetItemAttr(long item) const**

This function may be overloaded in the derived class for a control with `wxLC_VIRTUAL` style. It should return the attribute for the specified `item` or `NULL` to use the default appearance parameters.

`wxListCtrl` will not delete the pointer or keep a reference of it. You can return the same `wxListItemAttr` pointer for every `OnGetItemAttr` call.

The base class version always returns `NULL`.

#### **See also**

*OnGetItemImage* (p. 993),  
*OnGetItemColumnImage* (p. 993),  
*OnGetItemText* (p. 994)

### **wxListCtrl::OnGetItemImage**

**virtual int OnGetItemImage(long item) const**

This function must be overloaded in the derived class for a control with `wxLC_VIRTUAL` style having an *image list* (p. 995) (if the control doesn't have an image list, it is not necessary to overload it). It should return the index of the item's image in the control's image list or -1 for no image. In a control with `wxLC_REPORT` style, `OnGetItemImage` only gets called for the first column of each line.

The base class version always returns -1.

#### **See also**

*OnGetItemText* (p. 994),  
*OnGetItemColumnImage* (p. 993),  
*OnGetItemAttr* (p. 993)

### **wxListCtrl::OnGetItemColumnImage**

**virtual int OnGetItemColumnImage(long item, long column) const**

Overload this function in the derived class for a control with `wxLC_VIRTUAL` and `wxLC_REPORT` styles in order to specify the image index for the given line and column.

The base class version always calls `OnGetItemImage` for the first column, else it returns -1.

**See also**

*OnGetItemText* (p. 994),  
*OnGetItemImage* (p. 993),  
*OnGetItemAttr* (p. 993)

**wxListCtrl::OnGetItemText**

**virtual wxString OnGetItemText(long *item*, long *column*) const**

This function **must** be overloaded in the derived class for a control with `wxLC_VIRTUAL` style. It should return the string containing the text of the given *column* for the specified *item*.

**See also**

*SetItemCount* (p. 997),  
*OnGetItemImage* (p. 993),  
*OnGetItemColumnImage* (p. 993),  
*OnGetItemAttr* (p. 993)

**wxListCtrl::RefreshItem**

**void RefreshItem(long *item*)**

Redraws the given *item*. This is only useful for the virtual list controls as without calling this function the displayed value of the item doesn't change even when the underlying data does change.

**See also**

*RefreshItems* (p. 994)

**wxListCtrl::RefreshItems**

**void RefreshItems(long *itemFrom*, long *itemTo*)**

Redraws the items between *itemFrom* and *itemTo*. The starting item must be less than or equal to the ending one.

Just as *RefreshItem* (p. 994) this is only useful for virtual list controls.

**wxListCtrl::ScrollList**

**bool ScrollList(int *dx*, int *dy*)**

Scrolls the list control. If in icon, small icon or report view mode, *dx* specifies the number of pixels to scroll. If in list view mode, *dx* specifies the number of columns to scroll. *dy* always specifies the number of pixels to scroll vertically.

**NB:** This method is currently only implemented in the Windows version.

**wxListCtrl::SetBackgroundColour****void SetBackgroundColour(const wxColour& col)**

Sets the background colour (GetBackgroundColour already implicit in wxWindow class).

**wxListCtrl::SetColumn****bool SetColumn(int col, wxListItem& item)**

Sets information about this column. See *wxListCtrl::SetItem* (p. 995) for more information.

**wxListCtrl::SetColumnWidth****bool SetColumnWidth(int col, int width)**

Sets the column width.

*width* can be a width in pixels or wxLIST\_AUTOSIZE (-1) or wxLIST\_AUTOSIZE\_USEHEADER (-2). wxLIST\_AUTOSIZE will resize the column to the length of its longest item. wxLIST\_AUTOSIZE\_USEHEADER will resize the column to the length of the header (Win32) or 80 pixels (other platforms).

In small or normal icon view, *col* must be -1, and the column width is set for all columns.

**wxListCtrl::SetImageList****void SetImageList(wxImageList\* imageList, int which)**

Sets the image list associated with the control. *which* is one of wxIMAGE\_LIST\_NORMAL, wxIMAGE\_LIST\_SMALL, wxIMAGE\_LIST\_STATE (the last is unimplemented).

This method does not take ownership of the image list, you have to delete it yourself.

**See also**

*wxListCtrl::AssignImageList* (p. 984)

**wxListCtrl::SetItem****bool SetItem(wxListItem& info)****long SetItem(long index, int col, const wxString& label, int imageId = -1)**

Sets information about the item.

wxListItem is a class with the following members:

long m_mask	Indicates which fields are valid. See the list of valid mask flags below.
long m_itemId	The zero-based item position.

int m_col	Zero-based column, if in report mode.
long m_state	The state of the item. See the list of valid state flags below.
long m_stateMask	A mask indicating which state flags are valid. See the list of valid state flags below.
wxString m_text	The label/header text.
int m_image	The zero-based index into an image list.
long m_data	Application-defined data.
int m_format	For columns only: the format. Can be wxLIST_FORMAT_LEFT, wxLIST_FORMAT_RIGHT or wxLIST_FORMAT_CENTRE.
int m_width	For columns only: the column width.

The **m\_mask** member contains a bitlist specifying which of the other fields are valid. The flags are:

wxLIST_MASK_STATE	The <b>m_state</b> field is valid.
wxLIST_MASK_TEXT	The <b>m_text</b> field is valid.
wxLIST_MASK_IMAGE	The <b>m_image</b> field is valid.
wxLIST_MASK_DATA	The <b>m_data</b> field is valid.
wxLIST_MASK_WIDTH	The <b>m_width</b> field is valid.
wxLIST_MASK_FORMAT	The <b>m_format</b> field is valid.

The **m\_stateMask** and **m\_state** members take flags from the following:

wxLIST_STATE_DONTCARE	Don't care what the state is. Win32 only.
wxLIST_STATE_DROPHILITED	The item is highlighted to receive a drop event. Win32 only.
wxLIST_STATE_FOCUSED	The item has the focus.
wxLIST_STATE_SELECTED	The item is selected.
wxLIST_STATE_CUT	The item is in the cut state. Win32 only.

The wxListItem object can also contain item-specific colour and font information: for this you need to call one of SetTextColour(), SetBackgroundColour() or SetFont() functions on it passing it the colour/font to use. If the colour/font is not specified, the default list control colour/font is used.

**long SetItem(long index, int col, const wxString& label, int imageId = -1)**

Sets a string field at a particular column.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

<b>SetItem(item)</b>	Sets information about the given wxListItem.
<b>SetStringItem(index, col, label, imageld)</b>	Sets a string or image at a given location.

### **wxListCtrl::SetItemBackgroundColour**

**void SetItemBackgroundColour(long item, const wxColour& col)**

Sets the background colour for this item. This function only works in report view.

The colour can be retrieved using *GetItemBackgroundColour* (p. 987).

### **wxListCtrl::SetItemCount**

**void SetItemCount(long count)**

This method can only be used with virtual list controls. It is used to indicate to the control the number of items it contains. After calling it, the main program should be ready to handle calls to various item callbacks (such as *OnGetItemText* (p. 994)) for all items in the range from 0 to *count*.

### **wxListCtrl::SetItemData**

**bool SetItemData(long item, long data)**

Associates application-defined data with this item.

Notice that this function cannot be used to associate pointers with the control items, use *SetItemPtrData* (p. 998) instead.

### **wxListCtrl::SetItemFont**

**void SetItemFont(long item, const wxFont& font)**

Sets the item's font.

### **wxListCtrl::SetItemImage**

**bool SetItemImage(long item, int image)**

Sets the image associated with the item. The image is an index into the image list associated with the list control. In report view, this only sets the image for the first column.

**bool SetItemImage(long item, int image, int selImage)**

Sets the unselected and selected images associated with the item. The images are indices into the image list associated with the list control. This form is deprecated:

*selImage* is not used.

### **wxListCtrl::SetItemColumnImage**

**bool SetItemImage(long item, long column, int image)**

Sets the image associated with the item. In report view, you can specify the column. The image is an index into the image list associated with the list control.

### **wxListCtrl::SetItemPosition**

**bool SetItemPosition(long item, const wxPoint& pos)**

Sets the position of the item, in icon or small icon view. Windows only.

### **wxListCtrl::SetItemPtrData**

**bool SetItemPtrData(long item, wxUIntPtr data)**

Associates application-defined data with this item. The *data* parameter may be either an integer or a pointer cast to the `wxUIntPtr` type which is guaranteed to be large enough to be able to contain all integer types and pointers.

This function is new since wxWidgets version 2.8.4

### **wxListCtrl::SetItemState**

**bool SetItemState(long item, long state, long stateMask)**

Sets the item state. For a list of state flags, see *wxListCtrl::SetItem* (p. 995).

The **stateMask** indicates which state flags are valid.

### **wxListCtrl::SetItemText**

**void SetItemText(long item, const wxString& text)**

Sets the item text for this item.

### **wxListCtrl::SetItemTextColour**

**void SetItemTextColour(long item, const wxColour& col)**

Sets the colour for this item. This function only works in report view.

The colour can be retrieved using *GetItemTextColour* (p. 989).

### **wxListCtrl::SetSingleStyle**

**void SetSingleStyle(long style, const bool add = true)**

Adds or removes a single window style.

### **wxListCtrl::SetTextColour**

**void SetTextColour(const wxColour& col)**

Sets the text colour of the list control.

### **wxListCtrl::SetWindowStyleFlag**

**void SetWindowStyleFlag(long style)**

Sets the whole window style, deleting all items.

### **wxListCtrl::SortItems**

**bool SortItems(wxListCtrlCompare fnSortCallBack, long data)**

Call this function to sort the items in the list control. Sorting is done using the specified *fnSortCallBack* function. This function must have the following prototype:

```
int wxCALLBACK wxListCompareFunction(long item1, long item2, long
sortData)
```

It is called each time when the two items must be compared and should return 0 if the items are equal, negative value if the first item is less than the second one and positive value if the first one is greater than the second one (the same convention as used by `qsort(3)`).

#### **Parameters**

*item1*

client data associated with the first item (**NOT** the index).

*item2*

client data associated with the second item (**NOT** the index).

*data*

the value passed to `SortItems()` itself.

Notice that the control may only be sorted on client data associated with the items, so you **must** use *SetItemData* (p. 997) if you want to be able to sort the items in the control.

Please see the *listctrl sample* (p. 2036) for an example of using this function.

**wxPython note:** wxPython uses the `sortData` parameter to pass the Python function to call, so it is not available for programmer use. Call `SortItems` with a reference to a callable object that expects two parameters.

**wxPerl note:** In wxPerl the comparison function must take just two parameters; however,

you may use a closure to achieve an effect similar to the `SortItems` third parameter.

## **wxListEvent**

A list event holds information about events associated with `wxListCtrl` objects.

### **Derived from**

*wxNotifyEvent* (p. 1146)  
*wxCommandEvent* (p. 250)  
*wxEvent* (p. 572)  
*wxObject* (p. 1148)

### **Include files**

<wx/listctrl.h>

### **Event table macros**

To process input from a list control, use these event handler macros to direct input to member functions that take a `wxListEvent` argument.

**EVT\_LIST\_BEGIN\_DRAG(id, func)** Begin dragging with the left mouse button.

**EVT\_LIST\_BEGIN\_RDRAG(id, func)** Begin dragging with the right mouse button.

**EVT\_LIST\_BEGIN\_LABEL\_EDIT(id, func)** Begin editing a label. This can be prevented by calling *Veto()* (p. 1147).

**EVT\_LIST\_END\_LABEL\_EDIT(id, func)** Finish editing a label. This can be prevented by calling *Veto()* (p. 1147).

**EVT\_LIST\_DELETE\_ITEM(id, func)** Delete an item.

**EVT\_LIST\_DELETE\_ALL\_ITEMS(id, func)** Delete all items.

**EVT\_LIST\_ITEM\_SELECTED(id, func)** The item has been selected.

**EVT\_LIST\_ITEM\_DESELECTED(id, func)** The item has been deselected.

**EVT\_LIST\_ITEM\_ACTIVATED(id, func)** The item has been activated (ENTER or double click).

**EVT\_LIST\_ITEM\_FOCUSED(id, func)** The currently focused item has changed.

**EVT\_LIST\_ITEM\_MIDDLE\_CLICK(id, func)** The middle mouse button has been clicked on an item.

**EVT\_LIST\_ITEM\_RIGHT\_CLICK(id, func)** The right mouse button has been clicked on an item.

**EVT\_LIST\_KEY\_DOWN(id, func)** A key has been pressed.

**EVT\_LIST\_INSERT\_ITEM(id, func)** An item has been inserted.



- EVT\_LIST\_COL\_CLICK(id, func)** A column (**m\_col**) has been left-clicked.
- EVT\_LIST\_COL\_RIGHT\_CLICK(id, func)** A column (**m\_col**) (which can be -1 if the click occurred outside any column) has been right-clicked.
- EVT\_LIST\_COL\_BEGIN\_DRAG(id, func)** The user started resizing a column - can be vetoed.
- EVT\_LIST\_COL\_DRAGGING(id, func)** The divider between columns is being dragged.
- EVT\_LIST\_COL\_END\_DRAG(id, func)** A column has been resized by the user.
- EVT\_LIST\_CACHE\_HINT(id, func)** Prepare cache for a virtual list control

**See also**

*wxListCtrl* (p. 980)

**wxListEvent::wxListEvent**

**wxListEvent(WXTYPE *commandType* = 0, int *id* = 0)**

Constructor.

**wxListEvent::GetCacheFrom**

**long GetCacheFrom() const**

For `EVT_LIST_CACHE_HINT` event only: return the first item which the list control advises us to cache.

**wxListEvent::GetCacheTo**

**long GetCacheTo() const**

For `EVT_LIST_CACHE_HINT` event only: return the last item (inclusive) which the list control advises us to cache.

**wxListEvent::GetKeyCode**

**int GetKeyCode() const**

Key code if the event is a keypress event.

**wxListEvent::GetIndex**

**long GetIndex() const**

The item index.

**wxListEvent::GetColumn****int GetColumn() const**

The column position: it is only used with `COL` events. For the column dragging events, it is the column to the left of the divider being dragged, for the column click events it may be -1 if the user clicked in the list control header outside any column.

**wxListEvent::GetPoint****wxPoint GetPoint() const**

The position of the mouse pointer if the event is a drag event.

**wxListEvent::GetLabel****const wxString& GetLabel() const**

The (new) item label for `EVT_LIST_END_LABEL_EDIT` event.

**wxListEvent::GetText****const wxString& GetText() const**

The text.

**wxListEvent::GetImage****int GetImage() const**

The image.

**wxListEvent::GetData****long GetData() const**

The data.

**wxListEvent::GetMask****long GetMask() const**

The mask.

**wxListEvent::GetItem****const wxListItem& GetItem() const**

An item object, used by some events. See also `wxListCtrl::SetItem` (p. 995).

**wxCommandEvent::IsEditCancelled****bool IsEditCancelled() const**

This method only makes sense for `EVT_LIST_END_LABEL_EDIT` message and returns `true` if it the label editing has been cancelled by the user (*GetLabel* (p. 1002) returns an empty string in this case but it doesn't allow the application to distinguish between really cancelling the edit and the admittedly rare case when the user wants to rename it to an empty string).

**wxListItem**

This class stores information about a `wxListCtrl` item or column.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/listctrl.h>

**wxListItem::wxListItem****wxListItem()**

Constructor.

**wxListItem::Clear****void Clear()**

Resets the item state to the default.

**wxListItem::GetAlign****wxListColumnFormat GetAlign() const**

Returns the alignment for this item. Can be one of `wxLIST_FORMAT_LEFT`, `wxLIST_FORMAT_RIGHT` or `wxLIST_FORMAT_CENTRE`.

**wxListItem::GetBackgroundColour****wxColour GetBackgroundColour() const**

Returns the background colour for this item.

**wxListItem::GetColumn**

**int GetColumn() const**

Returns the zero-based column; meaningful only in report mode.

**wxListItem::GetData****long GetData() const**

Returns client data associated with the control. Please note that client data is associated with the item and not with subitems.

**wxListItem::GetFont****wxFont GetFont() const**

Returns the font used to display the item.

**wxListItem::GetId****long GetId() const**

Returns the zero-based item position.

**wxListItem::GetImage****int GetImage() const**

Returns the zero-based index of the image associated with the item into the image list.

**wxListItem::GetMask****long GetMask() const**

Returns a bit mask indicating which fields of the structure are valid; can be any combination of the following values:

wxLIST\_MASK\_STATE      **GetState** is valid.

wxLIST\_MASK\_TEXT      **GetText** is valid.

wxLIST\_MASK\_IMAGE      **GetImage** is valid.

wxLIST\_MASK\_DATA      **GetData** is valid.

wxLIST\_MASK\_WIDTH      **GetWidth** is valid.

wxLIST\_MASK\_FORMAT      **GetFormat** is valid.

**wxListItem::GetState****long GetState() const**

Returns a bit field representing the state of the item. Can be any combination of:

`wxLIST_STATE_DONTCARE`      Don't care what the state is. Win32 only.

`wxLIST_STATE_DROPHILITED`      The item is highlighted to receive a drop event.  
Win32 only.

`wxLIST_STATE_FOCUSED`      The item has the focus.

`wxLIST_STATE_SELECTED`      The item is selected.

`wxLIST_STATE_CUT`      The item is in the cut state. Win32 only.

### **`wxListItem::GetText`**

**`const wxString& GetText() const`**

Returns the label/header text.

### **`wxListItem::GetTextColour`**

**`wxColour GetTextColour() const`**

Returns the text colour.

### **`wxListItem::GetWidth`**

**`int GetWidth() const`**

Meaningful only for column headers in report mode. Returns the column width.

### **`wxListItem::SetAlign`**

**`void SetAlign(wxListColumnFormat align)`**

Sets the alignment for the item. See also `wxListItem::GetAlign` (p. 1003)

### **`wxListItem::SetBackgroundColour`**

**`void SetBackgroundColour(const wxColour& colBack)`**

Sets the background colour for the item.

### **`wxListItem::SetColumn`**

**`void SetColumn(int col)`**

Sets the zero-based column. Meaningful only in report mode.

### **`wxListItem::SetData`**

**void SetData(long data)**

**void SetData(void\* data)**

Sets client data for the item. Please note that client data is associated with the item and not with subitems.

**wxListItem::SetFont**

**void SetFont(const wxFont& font)**

Sets the font for the item.

**wxListItem::SetId**

**void SetId(long id)**

Sets the zero-based item position.

**wxListItem::SetImage**

**void SetImage(int image)**

Sets the zero-based index of the image associated with the item into the image list.

**wxListItem::SetMask**

**void SetMask(long mask)**

Sets the mask of valid fields. See *wxListItem::GetMask* (p. 1004).

**wxListItem::SetState**

**void SetState(long state)**

Sets the item state flags (note that the valid state flags are influenced by the value of the state mask, see *wxListItem::SetStateMask* (p. 1006)). See *wxListItem::GetState* (p. 1004) for valid flag values.

**wxListItem::SetStateMask**

**void SetStateMask(long stateMask)**

Sets the bitmask that is used to determine which of the state flags are to be set. See also *wxListItem::SetState* (p. 1006).

**wxListItem::SetText**

**void SetText(const wxString& text)**

Sets the text label for the item.

### **wxListItem::SetTextColour**

**void SetTextColour(const wxColour& colText)**

Sets the text colour for the item.

### **wxListItem::SetWidth**

**void SetWidth(int width)**

Meaningful only for column headers in report mode. Sets the column width.

## **wxListItemAttr**

Represents the attributes (color, font, ...) of a *wxListCtrl* (p. 980) *wxListItem* (p. 980).

### **Include files**

<wx/listctrl.h>

### **See also**

*wxListCtrl* overview (p. 2126), *wxListCtrl* (p. 980), *wxListItem* (p. 1003)

### **wxListItemAttr::wxListItemAttr**

**wxListItemAttr()**

Default constructor.

**wxListItemAttr(const wxColour& colText, const wxColour& colBack, const wxFont& font)**

Construct a *wxListItemAttr* with the specified foreground and background colors and font.

### **wxListItemAttr::GetBackgroundColour**

**const wxColour& GetBackgroundColour() const**

Returns the currently set background color.

### **wxListItemAttr::GetFont**

**const wxFont& GetFont() const**

Returns the currently set font.

**wxListItemAttr::GetTextColour****const wxColour& GetTextColour() const**

Returns the currently set text color.

**wxListItemAttr::HasBackgroundColour****bool HasBackgroundColour() const**

Returns `true` if the currently set background color is valid.

**wxListItemAttr::HasFont****bool HasFont() const**

Returns `true` if the currently set font is valid.

**wxListItemAttr::HasTextColour****bool HasTextColour() const**

Returns `true` if the currently set text color is valid.

**wxListItemAttr::SetBackgroundColour****void SetBackgroundColour(const wxColour& colour)**

Sets a new background color.

**wxListItemAttr::SetFont****void SetFont(const wxFont& font)**

Sets a new font.

**wxListItemAttr::SetTextColour****void SetTextColour(const wxColour& colour)**

Sets a new text color.

**wxListView**

This class currently simply presents a simpler to use interface for the *wxListCtrl* (p. 980) -- it can be thought of as a *façade* for that complicated class. Using it is preferable to using *wxListCtrl* (p. 980) directly whenever possible because in the future some ports might implement *wxListView* but not the full set of *wxListCtrl* features.



Other than different interface, this class is identical to `wxListCtrl`. In particular, it uses the same events, same window styles and so on.

**Derived from**

`wxListCtrl` (p. 980)  
`wxControl` (p. 285)  
`wxWindow` (p. 1795)  
`wxEvtHandler` (p. 576)  
`wxObject` (p. 1148)

**Include files**

<wx/listctrl.h>

**`wxListView::ClearColumnImage`**

**`void ClearColumnImage(int col)`**

Resets the column image -- after calling this function, no image will be shown.

**Parameters**

*col*

the column to clear image for

**See also**

`SetColumnImage` (p. 1010)

**`wxListView::Focus`**

**`void Focus(long index)`**

Sets focus to the item with the given *index*.

**`wxListView::GetFirstSelected`**

**`long GetFirstSelected() const`**

Returns the first selected item in a (presumably) multiple selection control. Together with `GetNextSelected` (p. 1010) it can be used to iterate over all selected items in the control.

**Return value**

The first selected item, if any, -1 otherwise.

**`wxListView::GetFocusedItem`**

**`long GetFocusedItem() const`**

Returns the currently focused item or -1 if none.

**See also**

*IsSelected* (p. 1010),  
*Focus* (p. 1009)

**wxListView::GetNextSelected**

**long GetNextSelected(long *item*) const**

Used together with *GetFirstSelected* (p. 1009) to iterate over all selected items in the control.

**Return value**

Returns the next selected item or -1 if there are no more of them.

**wxListView::IsSelected**

**bool IsSelected(long *index*) const**

Returns `true` if the item with the given *index* is selected, `false` otherwise.

**See also**

*GetFirstSelected* (p. 1009),  
*GetNextSelected* (p. 1010)

**wxListView::Select**

**void Select(long *n*, bool *on* = true)**

Selects or unselects the given item.

**Parameters**

*n*

the item to select or unselect

*on*

if `true` (default), selects the item, otherwise unselects it

**See also**

*SetItemState* (p. 998)

**wxListView::SetColumnImage**

**void SetColumnImage(int *col*, int *image*)**

Sets the column image for the specified column. To use the column images, the control must have a valid image list with at least one image.

### Parameters

*col*

the column to set image for

*image*

the index of the column image in the controls image list

### See also

*ClearColumnImage* (p. 1009),  
*SetImageList* (p. 995)

## wxLocale

wxLocale class encapsulates all language-dependent settings and is a generalization of the C locale concept.

In wxWidgets this class manages message catalogs which contain the translations of the strings used to the current language.

**wxPerl note:** In wxPerl you can't use the '\_' function name, so the `Wx::Locale` module can export the `gettext` and `gettext_noop` under any given name.

```
# this imports gettext ( equivalent to Wx::GetTranslation
# and gettext_noop ( a noop )
# into your module
use Wx::Locale qw(:default);

# ....

# use the functions
print gettext( ``Panic!' ' );

button = Wx::Button->new( window, -1, gettext( ``Label!' ' ) );
```

If you need to translate a lot of strings, then adding `gettext( )` around each one is a long task ( that is why `_()` was introduced ), so just choose a shorter name for `gettext`:

```
#
use Wx::Locale 'gettext' => 't',
               'gettext_noop' => 'gettext_noop';

# ...

# use the functions
print t( ``Panic!!!' ' );

# ...
```

**Derived from**

No base class

**See also**

*Internationalization overview* (p. 2062),  
*Internat sample* (p. 2036)

**Include files**

<wx/intl.h>

**Supported languages**

See *list of recognized language constants* (p. 1996). These constants may be used to specify the language in *Init* (p. 1017) and are returned by *GetSystemLanguage* (p. 1016):

**wxLocale::wxLocale****wxLocale()**

This is the default constructor and it does nothing to initialize the object: *Init()* (p. 1017) must be used to do that.

**wxLocale(int language, int flags = wxLOCALE\_LOAD\_DEFAULT |  
wxLOCALE\_CONV\_ENCODING)**

See *Init()* (p. 1017) for parameters description.

**wxLocale(const char \*szName, const char \*szShort = NULL, const char \*szLocale = NULL, bool bLoadDefault = true, bool bConvertEncoding = false)**

See *Init()* (p. 1017) for parameters description.

The call of this function has several global side effects which you should understand: first of all, the application locale is changed - note that this will affect many of standard C library functions such as *printf()* or *strftime()*. Second, this *wxLocale* object becomes the new current global locale for the application and so all subsequent calls to *wxGetTranslation()* will try to translate the messages using the message catalogs for this locale.

**wxLocale::~~wxLocale****~wxLocale()**

The destructor, like the constructor, also has global side effects: the previously set locale is restored and so the changes described in *Init* (p. 1017) documentation are rolled back.

**wxLocale::AddCatalog**

**bool AddCatalog(const char \*szDomain)**

**bool AddCatalog(const char \*szDomain, wxLanguage msgldLanguage, const char \*msgldCharset)**

Add a catalog for use with the current locale: it is searched for in standard places (current directory first, then the system one), but you may also prepend additional directories to the search path with *AddCatalogLookupPathPrefix()* (p. 1013).

All loaded catalogs will be used for message lookup by *GetString()* (p. 1015) for the current locale.

Returns true if catalog was successfully loaded, false otherwise (which might mean that the catalog is not found or that it isn't in the correct format).

The second form of this method takes two additional arguments, *msgldLanguage* and *msgldCharset*.

*msgldLanguage* specifies the language of "msgid" strings in source code (i.e. arguments to *GetString* (p. 1015), *wxGetTranslation* (p. 1930) and the *\_()* (p. 1933) macro). It is used if AddCatalog cannot find any catalog for current language: if the language is same as source code language, then strings from source code are used instead.

*msgldCharset* lets you specify the charset used for msgids in sources in case they use 8-bit characters (e.g. German or French strings). This argument has no effect in Unicode build, because literals in sources are Unicode strings; you have to use compiler-specific method of setting the right charset when compiling with Unicode.

By default (i.e. when you use the first form), msgid strings are assumed to be in English and written only using 7-bit ASCII characters.

If you have to deal with non-English strings or 8-bit characters in the source code, see the instructions in *Writing non-English applications* (p. 2063).

### **wxLocale::AddCatalogLookupPathPrefix**

**void AddCatalogLookupPathPrefix(const wxString& prefix)**

Add a prefix to the catalog lookup path: the message catalog files will be looked up under prefix/<lang>/LC\_MESSAGES, prefix/<lang> and prefix (in this order).

This only applies to subsequent invocations of *AddCatalog()*.

### **wxLocale::AddLanguage**

**static void AddLanguage(const wxLanguageInfo& info)**

Adds custom, user-defined language to the database of known languages. This database is used in conjunction with the first form of *Init* (p. 1017).

*wxLanguageInfo* is defined as follows:

```
struct WXDLL_EXPORT wxLanguageInfo
```

```
{
    int Language;                // wxLanguage id
    wxString CanonicalName;      // Canonical name, e.g. fr_FR
#ifdef __WIN32__
    wxUint32 WinLang, WinSublang; // Win32 language identifiers
                                   // (LANG_XXXX, SUBLANG_XXXX)
#endif
    wxString Description;        // human-readable name of the language
};
```

*Language* should be greater than `wxLANGUAGE_USER_DEFINED`.

**wxPerl note:** In wxPerl `Wx::LanguageInfo` has only one method:

```
Wx::LanguageInfo->new( language, canonicalName, WinLang, WinSubLang,
Description )
```

### **wxLocale::FindLanguageInfo**

**static wxLanguageInfo \* FindLanguageInfo(const wxString& locale)**

This function may be used to find the language description structure for the given locale, specified either as a two letter ISO language code (for example, "pt"), a language code followed by the country code ("pt\_BR") or a full, human readable, language description ("Portuguese-Brazil").

Returns the information for the given language or `NULL` if this language is unknown. Note that even if the returned pointer is valid, the caller should *not* delete it.

#### **See also**

*GetLanguageInfo* (p. 1015)

### **wxLocale::GetCanonicalName**

**wxString GetCanonicalName() const**

Returns the canonical form of current locale name. Canonical form is the one that is used on UNIX systems: it is a two- or five-letter string in `xx` or `xx_YY` format, where `xx` is ISO 639 code of language and `YY` is ISO 3166 code of the country. Examples are "en", "en\_GB", "en\_US" or "fr\_FR".

This form is internally used when looking up message catalogs.

Compare *GetSysName* (p. 1016).

### **wxLocale::GetLanguage**

**int GetLanguage() const**

Returns *wxLanguage* (p. 1012) constant of current language. Note that you can call this function only if you used the form of *Init* (p. 1017) that takes *wxLanguage* argument.

**wxLocale::GetLanguageInfo****static wxLanguageInfo \* GetLanguageInfo(int lang) const**

Returns a pointer to wxLanguageInfo structure containing information about the given language or `NULL` if this language is unknown. Note that even if the returned pointer is valid, the caller should *not* delete it.

See *AddLanguage* (p. 1013) for the wxLanguageInfo description.

As with *Init* (p. 1017), `wxLANGUAGE_DEFAULT` has the special meaning if passed as an argument to this function and in this case the result of *GetSystemLanguage*() (p. 1016) is used.

**wxLocale::GetLanguageName****static wxString GetLanguageName(int lang) const**

Returns English name of the given language or empty string if this language is unknown.

See *GetLanguageInfo* (p. 1015) for a remark about special meaning of `wxLANGUAGE_DEFAULT`.

**wxLocale::GetLocale****const char\* GetLocale() const**

Returns the locale name as passed to the constructor or *Init*() (p. 1017). This is full, human-readable name, e.g. "English" or "French".

**wxLocale::GetName****const wxString& GetName() const**

Returns the current short name for the locale (as given to the constructor or the *Init*() function).

**wxLocale::GetString****const char\* GetString(const char \*szOrigString, const char \*szDomain = NULL) const****const char\* GetString(const char \*szOrigString, const char \*szOrigString2, size\_t n, const char \*szDomain = NULL) const**

Retrieves the translation for a string in all loaded domains unless the `szDomain` parameter is specified (and then only this catalog/domain is searched).

Returns original string if translation is not available (in this case an error message is generated the first time a string is not found; use *wxLogNull* (p. 2069) to suppress it).

The second form is used when retrieving translation of string that has different singular and plural form in English or different plural forms in some other language. It takes two

extra arguments: *szOrigString* parameter must contain the singular form of the string to be converted. It is also used as the key for the search in the catalog. The *szOrigString2* parameter is the plural form (in English). The parameter *n* is used to determine the plural form. If no message catalog is found *szOrigString* is returned if 'n == 1', otherwise *szOrigString2*. See GNU gettext manual ([http://www.gnu.org/manual/gettext/html\\_chapter/gettext\\_10.html#SEC150](http://www.gnu.org/manual/gettext/html_chapter/gettext_10.html#SEC150)) for additional information on plural forms handling.

This method is called by the *wxGetTranslation* (p. 1930) function and *\_()* (p. 1933) macro.

### Remarks

Domains are searched in the last to first order, i.e. catalogs added later override those added before.

### **wxLocale::GetHeaderValue**

**wxString GetHeaderValue(const char \*szHeader, const char \*szDomain = NULL) const**

Returns the header value for header *szHeader*. The search for *szHeader* is case sensitive. If an *szDomain* is passed, this domain is searched. Else all domains will be searched until a header has been found. The return value is the value of the header if found. Else this will be empty.

### **wxLocale::GetSysName**

**wxString GetSysName() const**

Returns current platform-specific locale name as passed to *setlocale()*.

Compare *GetCanonicalName* (p. 1014).

### **wxLocale::GetSystemEncoding**

**static wxFontEncoding GetSystemEncoding() const**

Tries to detect the user's default font encoding. Returns *wxFontEncoding* (p. 655) value or **wxFONTENCODING\_SYSTEM** if it couldn't be determined.

### **wxLocale::GetSystemEncodingName**

**static wxString GetSystemEncodingName() const**

Tries to detect the name of the user's default font encoding. This string isn't particularly useful for the application as its form is platform-dependent and so you should probably use *GetSystemEncoding* (p. 1016) instead.

Returns a user-readable string value or an empty string if it couldn't be determined.

### **wxLocale::GetSystemLanguage**



**static int GetSystemLanguage() const**

Tries to detect the user's default language setting. Returns *wxLanguage* (p. 1012) value or **wxLANGUAGE\_UNKNOWN** if the language-guessing algorithm failed.

**wxLocale::Init**

**bool Init(int language = wxLANGUAGE\_DEFAULT, int flags = wxLOCALE\_LOAD\_DEFAULT | wxLOCALE\_CONV\_ENCODING)**

**bool Init(const char \*szName, const char \*szShort = NULL, const char \*szLocale = NULL, bool bLoadDefault = true, bool bConvertEncoding = false)**

The second form is deprecated, use the first one unless you know what you are doing.

**Parameters***language*

*wxLanguage* (p. 1012) identifier of the locale. **wxLANGUAGE\_DEFAULT** has special meaning -- *wxLocale* will use system's default language (see *GetSystemLanguage* (p. 1016)).

*flags*

Combination of the following:

**wxLOCALE\_LOAD\_DEFAULT**      Load the message catalog for the given locale containing the translations of standard *wxWidgets* messages automatically.

**wxLOCALE\_CONV\_ENCODING**      Automatically convert message catalogs to platform's default encoding. Note that it will do only basic conversion between well-known pair like *iso8859-1* and *windows-1252* or *iso8859-2* and *windows-1250*. See *Writing non-English applications* (p. 2063) for detailed description of this behaviour. Note that this flag is meaningless in Unicode build.

*szName*

The name of the locale. Only used in diagnostic messages.

*szShort*

The standard 2 letter locale abbreviation; it is used as the directory prefix when looking for the message catalog files.

*szLocale*

The parameter for the call to *setlocale()*. Note that it is platform-specific.

*bLoadDefault*

May be set to false to prevent loading of the message catalog for the given locale containing the translations of standard wxWidgets messages. This parameter would be rarely used in normal circumstances.

#### *bConvertEncoding*

May be set to true to do automatic conversion of message catalogs to platform's native encoding. Note that it will do only basic conversion between well-known pair like iso8859-1 and windows-1252 or iso8859-2 and windows-1250. See *Writing non-English applications* (p. 2063) for detailed description of this behaviour.

The call of this function has several global side effects which you should understand: first of all, the application locale is changed - note that this will affect many of standard C library functions such as `printf()` or `strftime()`. Second, this `wxLocale` object becomes the new current global locale for the application and so all subsequent calls to `wxGetTranslation()` (p. 1930) will try to translate the messages using the message catalogs for this locale.

Returns true on success or false if the given locale couldn't be set.

### **wxLocale::IsAvailable**

#### **static bool IsAvailable(int lang)**

Check whether the operating system and/or C run time environment supports this locale. For example in Windows 2000 and Windows XP, support for many locales is not installed by default. Returns `true` if the locale is supported.

The argument *lang* is the `wxLanguage` identifier. To obtain this for a given a two letter ISO language code, use *FindLanguageInfo* (p. 1014) to obtain its `wxLanguageInfo` structure. See *AddLanguage* (p. 1013) for the `wxLanguageInfo` description.

This function is new since wxWidgets version 2.7.1.

### **wxLocale::IsLoaded**

#### **bool IsLoaded(const char\* domain) const**

Check if the given catalog is loaded, and returns true if it is.

According to GNU gettext tradition, each catalog normally corresponds to 'domain' which is more or less the application name.

See also: *AddCatalog* (p. 1012)

### **wxLocale::IsOk**

#### **bool IsOk() const**

Returns true if the locale could be set successfully.

## **wxLog**

`wxLog` class defines the interface for the *log targets* used by `wxWidgets` logging functions as explained in the *wxLog overview* (p. 2069). The only situations when you need to directly use this class is when you want to derive your own log target because the existing ones don't satisfy your needs. Another case is if you wish to customize the behaviour of the standard logging classes (all of which respect the `wxLog` settings): for example, set which trace messages are logged and which are not or change (or even remove completely) the timestamp on the messages.

Otherwise, it is completely hidden behind the `wxLogXXX()` functions and you may not even know about its existence.

See *log overview* (p. 2069) for the descriptions of `wxWidgets` logging facilities.

### Derived from

No base class

### Include files

<wx/log.h>

### Global functions

The functions in this section work with and manipulate the active log target. The `OnLog()` (p. 1022) is called by the `wxLogXXX()` functions and invokes the `DoLog()` (p. 1023) of the active log target if any. `Get/Set` methods are used to install/query the current active target and, finally, `DontCreateOnDemand()` (p. 1023) disables the automatic creation of a standard log target if none actually exists. It is only useful when the application is terminating and shouldn't be used in other situations because it may easily lead to a loss of messages.

`OnLog` (p. 1022)

`GetActiveTarget` (p. 1022)

`SetActiveTarget` (p. 1022)

`DontCreateOnDemand` (p. 1023)

`Suspend` (p. 1022)

`Resume` (p. 1022)

### Logging functions

There are two functions which must be implemented by any derived class to actually process the log messages: `DoLog` (p. 1023) and `DoLogString` (p. 1023). The second function receives a string which just has to be output in some way and the easiest way to write a new log target is to override just this function in the derived class. If more control over the output format is needed, then the first function must be overridden which allows to construct custom messages depending on the log level or even do completely different things depending on the message severity (for example, throw away all messages except warnings and errors, show warnings on the screen and forward the error messages to the user's (or programmer's) cell phone - maybe depending on whether the timestamp tells us if it is day or night in the current time zone).

There also functions to support message buffering. Why are they needed? Some of `wxLog` implementations, most notably the standard `wxLogGui` class, buffer the messages (for example, to avoid showing the user a zillion of modal message boxes one after another -- which would be really annoying). `Flush()` (p. 1023) shows them all and clears the buffer contents. This function doesn't do anything if the buffer is already empty.

`Flush` (p. 1023)

`FlushActive` (p. 1023)

## Customization

The functions below allow some limited customization of `wxLog` behaviour without writing a new log target class (which, aside of being a matter of several minutes, allows you to do anything you want).

The verbose messages are the trace messages which are not disabled in the release mode and are generated by `wxLogVerbose` (p. 1973). They are not normally shown to the user because they present little interest, but may be activated, for example, in order to help the user find some program problem.

As for the (real) trace messages, their handling depends on the settings of the (application global) *trace mask*. There are two ways to specify it: either by using `SetTraceMask` (p. 1025) and `GetTraceMask` (p. 1025) and using `wxLogTrace` (p. 1974) which takes an integer mask or by using `AddTraceMask` (p. 1021) for string trace masks.

The difference between bit-wise and string trace masks is that a message using integer trace mask will only be logged if all bits of the mask are set in the current mask while a message using string mask will be logged simply if the mask had been added before to the list of allowed ones.

For example,

```
// wxTraceOleCalls is one of standard bit masks
wxLogTrace(wxTraceRefCount | wxTraceOleCalls, "Active object ref
count: %d", nRef);
```

will do something only if the current trace mask contains both `wxTraceRefCount` and `wxTraceOle`, but

```
// wxTRACE_OleCalls is one of standard string masks
wxLogTrace(wxTRACE_OleCalls, "IFoo::Bar() called");
```

will log the message if it was preceded by

```
wxLog::AddTraceMask(wxTRACE_OleCalls);
```

Using string masks is simpler and allows to easily add custom ones, so this is the preferred way of working with trace messages. The integer trace mask is kept for compatibility and for additional (but very rarely needed) flexibility only.

The standard trace masks are given in `wxLogTrace` (p. 1974) documentation.

Finally, the `wxLog::DoLog()` function automatically prepends a time stamp to all the

messages. The format of the time stamp may be changed: it can be any string with % specifications fully described in the documentation of the standard *strftime()* function. For example, the default format is "[%d/%b/%y %H:%M:%S] " which gives something like "[17/Sep/98 22:10:16] " (without quotes) for the current date. Setting an empty string as the time format disables timestamping of the messages completely.

**NB:** Timestamping is disabled for Visual C++ users in debug builds by default because otherwise it would be impossible to directly go to the line from which the log message was generated by simply clicking in the debugger window on the corresponding error message. If you wish to enable it, please use *SetTimestamp* (p. 1024) explicitly.

*AddTraceMask* (p. 1021)  
*RemoveTraceMask* (p. 1025)  
*ClearTraceMasks* (p. 1021)  
*GetTraceMasks* (p. 1021)  
*IsAllowedTraceMask* (p. 1025)  
*SetVerbose* (p. 1023)  
*GetVerbose* (p. 1024)  
*SetTimestamp* (p. 1024)  
*GetTimestamp* (p. 1024)  
*SetTraceMask* (p. 1025)  
*GetTraceMask* (p. 1025)  
*SetRepetitionCounting* (p. 1024)  
*GetRepetitionCounting* (p. 1024)

### **wxLog::AddTraceMask**

**static void AddTraceMask(const wxString& mask)**

Add the *mask* to the list of allowed masks for *wxLogTrace* (p. 1974).

#### **See also**

*RemoveTraceMask* (p. 1025) *GetTraceMasks* (p. 1021)

### **wxLog::ClearTraceMasks**

**static void ClearTraceMasks()**

Removes all trace masks previously set with *AddTraceMask* (p. 1021).

#### **See also**

*RemoveTraceMask* (p. 1025)

### **wxLog::GetTraceMasks**

**static const wxString & GetTraceMasks()**

Returns the currently allowed list of string trace masks.

**See also**

*AddTraceMask* (p. 1021).

**wxLog::OnLog**

**static void OnLog**(wxLogLevel *level*, const char \* *message*)

Forwards the message at specified level to the *DoLog()* function of the active log target if there is any, does nothing otherwise.

**wxLog::GetActiveTarget**

**static wxLog \* GetActiveTarget()**

Returns the pointer to the active log target (may be NULL).

**wxLog::SetActiveTarget**

**static wxLog \* SetActiveTarget**(wxLog \* *logtarget*)

Sets the specified log target as the active one. Returns the pointer to the previous active log target (may be NULL). To suppress logging use a new instance of *wxLogNull* not NULL. If the active log target is set to NULL a new default log target will be created when logging occurs.

**wxLog::Suspend**

**static void Suspend()**

Suspends the logging until *Resume* (p. 1022) is called. Note that the latter must be called the same number of times as the former to undo it, i.e. if you call *Suspend()* twice you must call *Resume()* twice as well.

Note that suspending the logging means that the log sink won't be flushed periodically, it doesn't have any effect if the current log target does the logging immediately without waiting for *Flush* (p. 1023) to be called (the standard GUI log target only shows the log dialog when it is flushed, so *Suspend()* works as expected with it).

**See also**

*Resume* (p. 1022),  
*wxLogNull* (p. 2069)

**wxLog::Resume**

**static void Resume()**

Resumes logging previously suspended by a call to *Suspend* (p. 1022). All messages logged in the meanwhile will be flushed soon.

**wxLog::DoLog****virtual void DoLog(wxLogLevel level, const wxChar \*msg, time\_t timestamp)**

Called to process the message of the specified severity. *msg* is the text of the message as specified in the call of *wxLogXXX()* function which generated it and *timestamp* is the moment when the message was generated.

The base class version prepends the timestamp to the message, adds a prefix corresponding to the log level and then calls *DoLogString* (p. 1023) with the resulting string.

**wxLog::DoLogString****virtual void DoLogString(const wxChar \*msg, time\_t timestamp)**

Called to log the specified string. The timestamp is already included into the string but still passed to this function.

A simple implementation may just send the string to `stdout` or, better, `stderr`.

**wxLog::DontCreateOnDemand****static void DontCreateOnDemand()**

Instructs *wxLog* to not create new log targets on the fly if there is none currently. (Almost) for internal use only: it is supposed to be called by the application shutdown code.

Note that this function also calls *ClearTraceMasks* (p. 1021).

**wxLog::Flush****virtual void Flush()**

Shows all the messages currently in buffer and clears it. If the buffer is already empty, nothing happens.

**wxLog::FlushActive****static void FlushActive()**

Flushes the current log target if any, does nothing if there is none.

**See also**

*Flush* (p. 1023)

**wxLog::SetVerbose****static void SetVerbose(bool verbose = true)**

Activates or deactivates verbose mode in which the verbose messages are logged as the normal ones instead of being silently dropped.

### **wxLog::GetVerbose**

**static bool GetVerbose()**

Returns whether the verbose mode is currently active.

### **wxLog::SetLogLevel**

**static void SetLogLevel(wxLogLevel *logLevel*)**

Specifies that log messages with level > *logLevel* should be ignored and not sent to the active log target.

### **wxLog::GetLogLevel**

**static wxLogLevel GetLogLevel()**

Returns the current log level limit.

### **wxLog::SetRepetitionCounting**

**static void SetRepetitionCounting(bool *repetCounting* = true)**

Enables logging mode in which a log message is logged once, and in case exactly the same message successively repeats one or more times, only the number of repetitions is logged.

### **wxLog::GetRepetitionCounting**

**static bool GetRepetitionCounting()**

Returns whether the repetition counting mode is enabled.

### **wxLog::SetTimestamp**

**void SetTimestamp(const char \* *format*)**

Sets the timestamp format prepended by the default log targets to all messages. The string may contain any normal characters as well as %prefixed format specifiers, see *strftime()* manual for details. Passing a NULL value (not empty string) to this function disables message timestamping.

### **wxLog::GetTimestamp**

**const char \* GetTimestamp() const**

Returns the current timestamp format string.



**wxLog::SetTraceMask****static void SetTraceMask(wxTraceMask mask)**

Sets the trace mask, see *Customization* (p. 1020) section for details.

**wxLog::GetTraceMask**

Returns the current trace mask, see *Customization* (p. 1020) section for details.

**wxLog::IsAllowedTraceMask****static bool IsAllowedTraceMask(const wxChar \*mask)**

Returns true if the *mask* is one of allowed masks for *wxLogTrace* (p. 1974).

See also: *AddTraceMask* (p. 1021), *RemoveTraceMask* (p. 1025)

**wxLog::RemoveTraceMask****static void RemoveTraceMask(const wxString& mask)**

Remove the *mask* from the list of allowed masks for *wxLogTrace* (p. 1974).

See also: *AddTraceMask* (p. 1021)

**wxLogChain**

This simple class allows to chain log sinks, that is to install a new sink but keep passing log messages to the old one instead of replacing it completely as *SetActiveTarget* (p. 1022) does.

It is especially useful when you want to divert the logs somewhere (for example to a file or a log window) but also keep showing the error messages using the standard dialogs as *wxLogGui* (p. 2069) does by default.

Example of usage:

```
wxLogChain *logChain = new wxLogChain(new wxLogStderr);

// all the log messages are sent to stderr and also processed as usually
...

// don't delete logChain directly as this would leave a dangling
// pointer as active log target, use SetActiveTarget() instead
delete wxLog::SetActiveTarget(...something else or NULL...);
```

**Derived from**

*wxLog* (p. 1018)

**Include files**

<wx/log.h>

**wxLogChain::wxLogChain**

**wxLogChain(wxLog \*logger)**

Sets the specified `logger` (which may be `NULL`) as the default log target but the log messages are also passed to the previous log target if any.

**wxLogChain::~~wxLogChain**

**~wxLogChain()**

Destroys the previous log target.

**wxLogChain::DetachOldLog**

**void DetachOldLog()**

Detaches the old log target so it won't be destroyed when the `wxLogChain` object is destroyed.

**wxLogChain::GetOldLog**

**wxLog \* GetOldLog() const**

Returns the pointer to the previously active log target (which may be `NULL`).

**wxLogChain::IsPassingMessages**

**bool IsPassingMessages() const**

Returns `true` if the messages are passed to the previously active log target (default) or `false` if *PassMessages* (p. 1026) had been called.

**wxLogChain::PassMessages**

**void PassMessages(bool passMessages)**

By default, the log messages are passed to the previously active log target. Calling this function with `false` parameter disables this behaviour (presumably temporarily, as you shouldn't use `wxLogChain` at all otherwise) and it can be reenabled by calling it again with *passMessages* set to `true`.

**wxLogChain::SetLog**

**void SetLog(wxLog \*logger)**

Sets another log target to use (may be `NULL`). The log target specified in the *constructor* (p. 1026) or in a previous call to this function is deleted.

This doesn't change the old log target value (the one the messages are forwarded to) which still remains the same as was active when `wxLogChain` object was created.

## wxLogGui

This is the default log target for the GUI `wxWidgets` applications. It is passed to `wxLog::SetActiveTarget` (p. 1022) at the program startup and is deleted by `wxWidgets` during the program shut down.

### Derived from

`wxLog` (p. 1018)

### Include files

<wx/log.h>

## wxLogGui::wxLogGui

**wxLogGui()**

Default constructor.

## wxLogNull

This class allows to temporarily suspend logging. All calls to the log functions during the life time of an object of this class are just ignored.

In particular, it can be used to suppress the log messages given by `wxWidgets` itself but it should be noted that it is rarely the best way to cope with this problem as **all** log messages are suppressed, even if they indicate a completely different error than the one the programmer wanted to suppress.

For instance, the example of the overview:

```
wxFile file;

// wxFile.Open() normally complains if file can't be opened, we don't
want it
{
    wxLogNull logNo;
    if ( !file.Open("bar") )
        ... process error ourselves ...
} // ~wxLogNull called, old log sink restored
```

```
wxLogMessage("..."); // ok
```

would be better written as:

```
wxFile file;

// don't try to open file if it doesn't exist, we are prepared to
deal with
// this ourselves - but all other errors are not expected
if ( wxFile::Exists("bar") )
{
    // gives an error message if the file couldn't be opened
    file.Open("bar");
}
else
{
    ...
}
```

### **Derived from**

*wxLog* (p. 1018)

### **Include files**

<wx/log.h>

## **wxLogNull::wxLogNull**

**wxLogNull()**

Suspends logging.

## **wxLogNull::~~wxLogNull**

Resumes logging.

## **wxLogPassThrough**

A special version of *wxLogChain* (p. 1025) which uses itself as the new log target. Maybe more clearly, it means that this is a log target which forwards the log messages to the previously installed one in addition to processing them itself.

Unlike *wxLogChain* (p. 1025) which is usually used directly as is, this class must be derived from to implement *DoLog* (p. 1023) and/or *DoLogString* (p. 1023) methods.

### **Derived from**

*wxLogChain* (p. 1025)

### **Include files**

<wx/log.h>

### **wxLogPassThrough::wxLogPassThrough**

Default ctor installs this object as the current active log target.

## **wxLogStderr**

This class can be used to redirect the log messages to a C file stream (not to be confused with C++ streams). It is the default log target for the non-GUI wxWidgets applications which send all the output to `stderr`.

### **Derived from**

*wxLog* (p. 1018)

### **Include files**

<wx/log.h>

### **See also**

*wxLogStream* (p. 1029)

### **wxLogStderr::wxLogStderr**

**wxLogStderr**(FILE \*fp = NULL)

Constructs a log target which sends all the log messages to the given `FILE`. If it is `NULL`, the messages are sent to `stderr`.

## **wxLogStream**

This class can be used to redirect the log messages to a C++ stream.

Please note that this class is only available if wxWidgets was compiled with the standard iostream library support (`wxUSE_STD_Iostream` must be on).

### **Derived from**

*wxLog* (p. 1018)

### **Include files**

<wx/log.h>

### **See also**

*wxLogStderr* (p. 1029),  
*wxStreamToTextRedirector* (p. 1552)

### **wxLogStream::wxLogStream**

**wxLogStream**(std::ostream \*ostr = NULL)

Constructs a log target which sends all the log messages to the given output stream. If it is `NULL`, the messages are sent to `cerr`.

### **wxLogTextCtrl**

Using these target all the log messages can be redirected to a text control. The text control must have been created with `wxTE_MULTILINE` style by the caller previously.

#### **Derived from**

*wxLog* (p. 1018)

#### **Include files**

<wx/log.h>

#### **See also**

*wxTextCtrl* (p. 1633),  
*wxStreamToTextRedirector* (p. 1552)

### **wxLogTextCtrl::wxLogTextCtrl**

**wxLogTextCtrl**(wxTextCtrl \*textctrl)

Constructs a log target which sends all the log messages to the given text control. The *textctrl* parameter cannot be `NULL`.

### **wxLogWindow**

This class represents a background log window: to be precise, it collects all log messages in the log frame which it manages but also passes them on to the log target which was active at the moment of its creation. This allows, for example, to show all the log messages in a frame but still continue to process them normally by showing the standard log dialog.

#### **Derived from**

*wxLogPassThrough* (p. 1028)

#### **Include files**

<wx/log.h>

### See also

*wxLogTextCtrl* (p. 1030)

## **wxLogWindow::wxLogWindow**

**wxLogWindow(wxFFrame \*parent, const wxChar \*title, bool show = true, bool passToOld = true)**

Creates the log frame window and starts collecting the messages in it.

### Parameters

*parent*

The parent window for the log frame, may be `NULL`

*title*

The title for the log frame

*show*

`true` to show the frame initially (default), otherwise *wxLogWindow::Show* (p. 1031) must be called later.

*passToOld*

`true` to process the log messages normally in addition to logging them in the log frame (default), `false` to only log them in the log frame.

## **wxLogWindow::Show**

**void Show(bool show = true)**

Shows or hides the frame.

## **wxLogWindow::GetFrame**

**wxFFrame \* GetFrame() const**

Returns the associated log frame window. This may be used to position or resize it but use *wxLogWindow::Show* (p. 1031) to show or hide it.

## **wxLogWindow::OnFrameCreate**

**virtual void OnFrameCreate(wxFFrame \*frame)**

Called immediately after the log frame creation allowing for any extra initializations.

### **wxLogWindow::OnFrameClose**

**virtual bool OnFrameClose(wxFrame \*frame)**

Called if the user closes the window interactively, will not be called if it is destroyed for another reason (such as when program exits).

Return `true` from here to allow the frame to close, `false` to prevent this from happening.

#### **See also**

*wxLogWindow::OnFrameDelete* (p. 1032)

### **wxLogWindow::OnFrameDelete**

**virtual void OnFrameDelete(wxFrame \*frame)**

Called right before the log frame is going to be deleted: will always be called unlike *OnFrameClose()* (p. 1032).

## **wxLongLong**

This class represents a signed 64 bit long number. It is implemented using the native 64 bit type where available (machines with 64 bit longs or compilers which have (an analog of) *long long* type) and uses the emulation code in the other cases which ensures that it is the most efficient solution for working with 64 bit integers independently of the architecture.

*wxLongLong* defines all usual arithmetic operations such as addition, subtraction, bitwise shifts and logical operations as well as multiplication and division (not yet for the machines without native *long long*). It also has operators for implicit construction from and conversion to the native *long long* type if it exists and *long*.

You would usually use this type in exactly the same manner as any other (built-in) arithmetic type. Note that *wxLongLong* is a signed type, if you want unsigned values use *wxULongLong* which has exactly the same API as *wxLongLong* except when explicitly mentioned otherwise.

If a native (i.e. supported directly by the compiler) 64 bit integer type was found to exist, *wxLongLong\_t* macro will be defined to correspond to it. Also, in this case only, two additional macros will be defined: *wxLongLongFmtSpec* (p. 1952) for printing 64 bit integers using the standard `printf()` function (but see also *ToString()* (p. 1035) for a more portable solution) and *wxLL* (p. 1952) for defining 64 bit integer compile-time constants.

#### **Derived from**

No base class

#### **Include files**



<wx/longlong.h>

### **wxLongLong::wxLongLong**

**wxLongLong()**

Default constructor initializes the object to 0.

### **wxLongLong::wxLongLong**

**wxLongLong(wxLongLong\_t //)**

Constructor from native long long (only for compilers supporting it).

### **wxLongLong::wxLongLong**

**wxLongLong(long hi, unsigned long lo)**

Constructor from 2 longs: the high and low part are combined into one wxLongLong.

### **wxLongLong::operator=**

**wxLongLong& operator operator=(wxLongLong\_t //)**

Assignment operator from native long long (only for compilers supporting it).

### **wxLongLong::operator=**

**wxLongLong& operator operator=(wxULongLong\_t //)**

Assignment operator from native unsigned long long (only for compilers supporting it).

This function is new since wxWidgets version 2.7.0

### **wxLongLong::operator=**

**wxLongLong& operator operator=(long //)**

Assignment operator from long.

This function is new since wxWidgets version 2.7.0

### **wxLongLong::operator=**

**wxLongLong& operator operator=(unsigned long //)**

Assignment operator from unsigned long.

This function is new since wxWidgets version 2.7.0

**wxLongLong::operator=****wxLongLong& operator operator=(const wxULongLong & //)**

Assignment operator from unsigned long long. The sign bit will be copied too.

This function is new since wxWidgets version 2.7.0

**wxLongLong::Abs****wxLongLong Abs() const****wxLongLong& Abs()**

Returns an absolute value of wxLongLong - either making a copy (const version) or modifying it in place (the second one). Not in wxULongLong.

**wxLongLong::Assign****wxLongLong& Assign(double d)**

This allows to convert a double value to wxLongLong type. Such conversion is not always possible in which case the result will be silently truncated in a platform-dependent way. Not in wxULongLong.

**wxLongLong::GetHi****long GetHi() const**

Returns the high 32 bits of 64 bit integer.

**wxLongLong::GetLo****unsigned long GetLo() const**

Returns the low 32 bits of 64 bit integer.

**wxLongLong::GetValue****wxLongLong\_t GetValue() const**

Convert to native long long (only for compilers supporting it)

**wxLongLong::ToDouble****double ToDouble() const**

Returns the value as double.

**wxLongLong::ToLong**

**long ToLong() const**

Truncate wxLongLong to long. If the conversion loses data (i.e. the wxLongLong value is outside the range of built-in long type), an assert will be triggered in debug mode.

**wxLongLong::ToString****wxString ToString() const**

Returns the string representation of a wxLongLong.

**wxLongLong::operator+****wxLongLong operator+(const wxLongLong& //) const**

Adds 2 wxLongLongs together and returns the result.

**wxLongLong::operator+=****wxLongLong& operator+(const wxLongLong& //)**

Add another wxLongLong to this one.

**wxLongLong::operator++****wxLongLong& operator++()****wxLongLong& operator++(int)**

Pre/post increment operator.

**wxLongLong::operator-****wxLongLong operator-() const**

Returns the value of this wxLongLong with opposite sign. Not in wxULongLong.

**wxLongLong::operator-****wxLongLong operator-(const wxLongLong& //) const**

Subtracts 2 wxLongLongs and returns the result.

**wxLongLong::operator-=****wxLongLong& operator-(const wxLongLong& //)**

Subtracts another wxLongLong from this one.

**wxLongLong::operator--**

**wxLongLong& operator--()**

**wxLongLong& operator--(int)**

Pre/post decrement operator.

## wxMask

This class encapsulates a monochrome mask bitmap, where the masked area is black and the unmasked area is white. When associated with a bitmap and drawn in a device context, the unmasked area of the bitmap will be drawn, and the masked area will not be drawn.

### Derived from

*wxObject* (p. 1148)

### Include files

<wx/bitmap.h>

### Remarks

A mask may be associated with a *wxBitmap* (p. 123). It is used in *wxDC::Blit* (p. 457) when the source device context is a *wxMemoryDC* (p. 1069) with *wxBitmap* selected into it that contains a mask.

### See also

*wxBitmap* (p. 123), *wxDC::Blit* (p. 457), *wxMemoryDC* (p. 1069)

## wxMask::wxMask

**wxMask()**

Default constructor.

**wxMask(const wxBitmap (p. 123)& bitmap)**

Constructs a mask from a monochrome bitmap.

**wxPython note:** This is the default constructor for *wxMask* in *wxPython*.

**wxMask(const wxBitmap (p. 123)& bitmap, const wxColour (p. 214)& colour)**

Constructs a mask from a bitmap and a colour that indicates the background.

**wxPython note:** *wxPython* has an alternate *wxMask* constructor matching this form called *wxMaskColour*.

**wxMask(const wxBitmap& bitmap, int index)**

Constructs a mask from a bitmap and a palette index that indicates the background. Not yet implemented for GTK.

**Parameters**

*bitmap*

A valid bitmap.

*colour*

A colour specifying the transparency RGB values.

*index*

Index into a palette, specifying the transparency colour.

**wxMask::~~wxMask**

**~wxMask()**

Destroys the wxMask object and the underlying bitmap data.

**wxMask::Create**

**bool Create(const wxBitmap& *bitmap*)**

Constructs a mask from a monochrome bitmap.

**bool Create(const wxBitmap& *bitmap*, const wxColour& *colour*)**

Constructs a mask from a bitmap and a colour that indicates the background.

**bool Create(const wxBitmap& *bitmap*, int *index*)**

Constructs a mask from a bitmap and a palette index that indicates the background. Not yet implemented for GTK.

**Parameters**

*bitmap*

A valid bitmap.

*colour*

A colour specifying the transparency RGB values.

*index*

Index into a palette, specifying the transparency colour.

**wxMaximizeEvent**

An event being sent when the frame is maximized or restored.

**Derived from**

*wxEvt* (p. 572)

*wxObject* (p. 1148)

**Include files**

<wx/event.h>

**Event table macros**

To process a maximize event, use this event handler macro to direct input to a member function that takes a *wxMaximizeEvent* argument.

**EVT\_MAXIMIZE(func)**                      Process a *wxEVT\_MAXIMIZE* event.

**See also**

*Event handling overview* (p. 2077), *wxTopLevelWindow::Maximize* (p. 1716),

*wxTopLevelWindow::IsMaximized* (p. 1715)

**wxMaximizeEvent::wxMaximizeEvent**

**wxMaximizeEvent(int id = 0)**

Constructor.

**wxMBConv**

This class is the base class of a hierarchy of classes capable of converting text strings between multibyte (SBCS or DBCS) encodings and Unicode.

In the documentation for this and related classes please notice that *length* of the string refers to the number of characters in the string not counting the terminating NUL, if any. While the *size* of the string is the total number of bytes in the string, including any trailing NUL. Thus, length of wide character string `L"foo"` is 3 while its size can be either 8 or 16 depending on whether `wchar_t` is 2 bytes (as under Windows) or 4 (Unix).

**Global variables**

There are several predefined instances of this class:  
**wxConvLibc** Uses the standard ANSI C `mbstowcs()` and `wcstombs()` functions to perform the conversions; thus depends on the current locale.

**wxConvLocal**

Another conversion corresponding to the current locale but this one uses the best available conversion.

<b>wxConvUI</b>	The conversion used for the standard UI elements such as menu items and buttons. This is a pointer which is initially set to <code>wxConvLocal</code> as the program uses the current locale by default but can be set to some specific conversion if the program needs to use a specific encoding for its UI.
<b>wxConvISO8859_1</b>	Conversion to and from ISO-8859-1 (Latin I) encoding.
<b>wxConvUTF8</b>	Conversion to and from UTF-8 encoding.
<b>wxConvFile</b>	The appropriate conversion for the file names, depends on the system.

### Constants

`wxCONV_FAILED` value is defined as `(size_t)-1` and is returned by the conversion functions instead of the length of the converted string if the conversion fails.

### Derived from

No base class

### Include files

`<wx/strconv.h>`

### See also

*wxCSSConv* (p. 296), *wxEncodingConverter* (p. 568), *wxMBConv classes overview* (p. 2059)

## **wxMBConv::wxMBConv**

### **wxMBConv()**

Trivial default constructor.

## **wxMBConv::MB2WC**

**virtual size\_t MB2WC(wchar\_t \*out, const char \*in, size\_t outLen) const**

This function is deprecated, please use *ToWChar* (p. 1042) instead

Converts from a string *in* in multibyte encoding to Unicode putting up to *outLen* characters into the buffer *out*.

If *out* is `NULL`, only the length of the string which would result from the conversion is calculated and returned. Note that this is the length and not size, i.e. the returned value

does *not* include the trailing NUL. But when the function is called with a non-NULL *out* buffer, the *outLen* parameter should be one more to allow to properly NUL-terminate the string.

### Parameters

*out*

The output buffer, may be NULL if the caller is only interested in the length of the resulting string

*in*

The NUL-terminated input string, cannot be NULL

*outLen*

The length of the output buffer but *including* NUL, ignored if *out* is NULL

### Return value

The length of the converted string *excluding* the trailing NUL.

### wxMBConv::WC2MB

**virtual size\_t WC2MB(char\* buf, const wchar\_t\* psz, size\_t n) const**

This function is deprecated, please use *FromWChar* (p. 1042) instead

Converts from Unicode to multibyte encoding. The semantics of this function (including the return value meaning) is the same as for *MB2WC* (p. 1039).

Notice that when the function is called with a non-NULL buffer, the *n* parameter should be the size of the buffer and so it *should* take into account the trailing NUL, which might take two or four bytes for some encodings (UTF-16 and UTF-32) and not one.

### wxMBConv::cMB2WC

**const wxWCharBuffer cMB2WC(const char \*in) const**

**const wxWCharBuffer cMB2WC(const char \*in, size\_t inLen, size\_t \*outLen) const**

Converts from multibyte encoding to Unicode by calling *MB2WC* (p. 1039), allocating a temporary wxWCharBuffer to hold the result.

The first overload takes a NUL-terminated input string. The second one takes a string of exactly the specified length and the string may include or not the trailing NUL character(s). If the string is not NUL-terminated, a temporary NUL-terminated copy of it suitable for passing to *MB2WC* (p. 1039) is made, so it is more efficient to ensure that the string is does have the appropriate number of NUL bytes (which is usually 1 but may be 2 or 4 for UTF-16 or UTF-32, see *GetMBNulLen* (p. 1042)), especially for long strings.



If *outLen* is not-NULL, it receives the length of the converted string.

### **wxMBConv::cWC2MB**

**const wxCharBuffer cWC2MB(const wchar\_t\* in) const**

**const wxCharBuffer cWC2MB(const wchar\_t\* in, size\_t inLen, size\_t \*outLen) const**

Converts from Unicode to multibyte encoding by calling WC2MB, allocating a temporary wxCharBuffer to hold the result.

The second overload of this function allows to convert a string of the given length *inLen*, whether it is NUL-terminated or not (for wide character strings, unlike for the multibyte ones, a single NUL is always enough). But notice that just as with *cMB2WC* (p. 1039), it is more efficient to pass an already terminated string to this function as otherwise a copy is made internally.

If *outLen* is not-NULL, it receives the length of the converted string.

### **wxMBConv::cMB2WX**

**const char\* cMB2WX(const char\* psz) const**

**const wxWCharBuffer cMB2WX(const char\* psz) const**

Converts from multibyte encoding to the current wxChar type (which depends on whether wxUSE\_UNICODE is set to 1). If wxChar is char, it returns the parameter unaltered. If wxChar is wchar\_t, it returns the result in a wxWCharBuffer. The macro wxMB2WXbuf is defined as the correct return type (without const).

### **wxMBConv::cWX2MB**

**const char\* cWX2MB(const wxChar\* psz) const**

**const wxCharBuffer cWX2MB(const wxChar\* psz) const**

Converts from the current wxChar type to multibyte encoding. If wxChar is char, it returns the parameter unaltered. If wxChar is wchar\_t, it returns the result in a wxCharBuffer. The macro wxWX2MBbuf is defined as the correct return type (without const).

### **wxMBConv::cWC2WX**

**const wchar\_t\* cWC2WX(const wchar\_t\* psz) const**

**const wxCharBuffer cWC2WX(const wchar\_t\* psz) const**

Converts from Unicode to the current wxChar type. If wxChar is wchar\_t, it returns the parameter unaltered. If wxChar is char, it returns the result in a wxCharBuffer. The macro wxWC2WXbuf is defined as the correct return type (without const).

### **wxMBConv::cWX2WC**

**const wchar\_t\* cWX2WC(const wxChar\* psz) const**

**const wxWCharBuffer cWX2WC(const wxChar\* psz) const**

Converts from the current wxChar type to Unicode. If wxChar is wchar\_t, it returns the parameter unaltered. If wxChar is char, it returns the result in a wxWCharBuffer. The macro wxWX2WCbuf is defined as the correct return type (without const).

### **wxMBConv::FromWChar**

**virtual size\_t FromWChar(wchar\_t \*dst, size\_t dstLen, const char \*src, size\_t srcLen = wxNO\_LEN) const**

The most general function for converting a multibyte string to a wide string. The main case is when *dst* is not NULL and *srcLen* is not wxNO\_LEN (which is defined as (size\_t)-1): then the function converts exactly *srcLen* bytes starting at *src* into wide string which it output to *dst*. If the length of the resulting wide string is greater than *dstLen*, an error is returned. Note that if *srcLen* bytes don't include NUL characters, the resulting wide string is not NUL-terminated neither.

If *srcLen* is wxNO\_LEN, the function supposes that the string is properly (i.e. as necessary for the encoding handled by this conversion) NUL-terminated and converts the entire string, including any trailing NUL bytes. In this case the wide string is also NUL-terminated.

Finally, if *dst* is NULL, the function returns the length of the needed buffer.

### **Return value**

The number of characters written to *dst* (or the number of characters which would have been written to it if it were non-NULL) on success or wxCONV\_FAILED on error.

### **wxMBConv::GetMaxMBNulLen**

**const size\_t GetMaxMBNulLen()**

Returns the maximal value which can be returned by *GetMBNulLen* (p. 1042) for any conversion object. Currently this value is 4.

This method can be used to allocate the buffer with enough space for the trailing NUL characters for any encoding.

### **wxMBConv::GetMBNulLen**

**size\_t GetMBNulLen() const**

This function returns 1 for most of the multibyte encodings in which the string is terminated by a single NUL, 2 for UTF-16 and 4 for UTF-32 for which the string is terminated with 2 and 4 NUL characters respectively. The other cases are not currently supported and wxCONV\_FAILED (defined as -1) is returned for them.

### **wxMBConv::ToWChar**

**virtual size\_t ToWChar(char\_t \*dst, size\_t dstLen, const wchar\_t \*src, size\_t srcLen = wxNO\_LEN) const**

This function has the same semantics as *FromWChar* (p. 1042) except that it converts a wide string to multibyte one.

## wxMBConvFile

This class used to define the class instance **wxConvFileName**, but nowadays **wxConvFileName** is either of type **wxConvLibc** (on most platforms) or **wxConvUTF8** (on MacOS X). **wxConvFileName** converts filenames between filesystem multibyte encoding and Unicode. **wxConvFileName** can also be set to a something else at run-time which is used e.g. by wxGTK to use a class which checks the environment variable **G\_FILESYSTEM\_ENCODING** indicating that filenames should not be interpreted as UTF8 and also for converting invalid UTF8 characters (e.g. if there is a filename in iso8859\_1) to strings with octal values.

Since some platforms (such as Win32) use Unicode in the filenames, and others (such as Unix) use multibyte encodings, this class should only be used directly if **wxMBFILES** is defined to 1. A convenience macro, **wxFNCONV**, is defined to **wxConvFileName->cWX2MB** in this case. You could use it like this:

```
wxChar *name = wxT("rawfile.doc");  
FILE *fil = fopen(wxFNCONV(name), "r");
```

(although it would be better to use **wxFopen(name, wxT("r"))** in this case.)

### Derived from

*wxMBConv* (p. 1038)

### Include files

<wx/strconv.h>

### See also

*wxMBConv classes overview* (p. 2059)

## wxMBConvFile::MB2WC

**size\_t MB2WC(wchar\_t\* buf, const char\* psz, size\_t n) const**

Converts from multibyte filename encoding to Unicode. Returns the size of the destination buffer.

## wxMBConvFile::WC2MB

**size\_t WC2MB(char\* buf, const wchar\_t\* psz, size\_t n) const**

Converts from Unicode to multibyte filename encoding. Returns the size of the destination buffer.

## **wxMBConvUTF7**

This class converts between the UTF-7 encoding and Unicode. It has one predefined instance, **wxConvUTF7**.

**WARNING:** this class is not implemented yet.

### **Derived from**

*wxMBConv* (p. 1038)

### **Include files**

<wx/strconv.h>

### **See also**

*wxMBConvUTF8* (p. 1044), *wxMBConv classes overview* (p. 2059)

## **wxMBConvUTF7::MB2WC**

**size\_t MB2WC(wchar\_t\* buf, const char\* psz, size\_t n) const**

Converts from UTF-7 encoding to Unicode. Returns the size of the destination buffer.

## **wxMBConvUTF7::WC2MB**

**size\_t WC2MB(char\* buf, const wchar\_t\* psz, size\_t n) const**

Converts from Unicode to UTF-7 encoding. Returns the size of the destination buffer.

## **wxMBConvUTF8**

This class converts between the UTF-8 encoding and Unicode. It has one predefined instance, **wxConvUTF8**.

### **Derived from**

*wxMBConv* (p. 1038)

### **Include files**

<wx/strconv.h>

### **See also**

*wxMBConvUTF7* (p. 1044), *wxMBConv classes overview* (p. 2059)

### Remarks

UTF-8 is a compatibility encoding used to encode Unicode text into anything that was originally written for 8-bit strings, including (but not limited to) filenames, transfer protocols, and database fields. Notable properties include:

- Variable-length encoding able to encode up to 31 bits per character
- ASCII characters (character values under 128) are encoded as plain ASCII (1 byte per character)
- Null bytes do not occur in the encoding, except when there's an actual Unicode null character
- Preserves sort ordering for plain 8-bit comparison routines like `strcmp()`
- High bit patterns disambiguates character boundaries, and makes it easy to detect whether a string is encoded with UTF-8 or not

All of these properties make UTF-8 a very favorable solution in any situation where full Unicode character support is desired while remaining compatible with code written with only 8-bit extended-ASCII characters in mind.

### **wxMBConvUTF8::MB2WC**

**size\_t MB2WC(wchar\_t\* buf, const char\* psz, size\_t n) const**

Converts from UTF-8 encoding to Unicode. Returns the size of the destination buffer.

### **wxMBConvUTF8::WC2MB**

**size\_t WC2MB(char\* buf, const wchar\_t\* psz, size\_t n) const**

Converts from Unicode to UTF-8 encoding. Returns the size of the destination buffer.

## **wxMBConvUTF16**

This class is used to convert between multibyte encodings and UTF-16 Unicode encoding (also known as UCS-2). Unlike *UTF-8* (p. 1044) encoding, UTF-16 uses words and not bytes and hence depends on the byte ordering: big or little endian. Hence this class is provided in two versions: *wxMBConvUTF16LE* and *wxMBConvUTF16BE* and *wxMBConvUTF16* itself is just a typedef for one of them (native for the given platform, e.g. LE under Windows and BE under Mac).

### Derived from

*wxMBConv* (p. 1038)

**Include files**

<wx/strconv.h>

**See also**

*wxMBConvUTF8* (p. 1044), *wxMBConvUTF32* (p. 1046), *wxMBConv classes overview* (p. 2059)

**wxMBConvUTF16::MB2WC**

**size\_t MB2WC(wchar\_t\* buf, const char\* psz, size\_t n) const**

Converts from UTF-16 encoding to Unicode. Returns the size of the destination buffer.

**wxMBConvUTF16::WC2MB**

**size\_t WC2MB(char\* buf, const wchar\_t\* psz, size\_t n) const**

Converts from Unicode to UTF-16 encoding. Returns the size of the destination buffer.

**wxMBConvUTF32**

This class is used to convert between multibyte encodings and UTF-32 Unicode encoding (also known as UCS-4). Unlike *UTF-8* (p. 1044) encoding, UTF-32 uses (double) words and not bytes and hence depends on the byte ordering: big or little endian. Hence this class is provided in two versions: *wxMBConvUTF32LE* and *wxMBConvUTF32BE* and *wxMBConvUTF32* itself is just a typedef for one of them (native for the given platform, e.g. LE under Windows and BE under Mac).

**Derived from**

*wxMBConv* (p. 1038)

**Include files**

<wx/strconv.h>

**See also**

*wxMBConvUTF8* (p. 1044), *wxMBConvUTF16* (p. 1045), *wxMBConv classes overview* (p. 2059)

**wxMBConvUTF32::MB2WC**

**size\_t MB2WC(wchar\_t\* buf, const char\* psz, size\_t n) const**

Converts from UTF-32 encoding to Unicode. Returns the size of the destination buffer.

**wxMBConvUTF32::WC2MB****size\_t WC2MB(char\* buf, const wchar\_t\* psz, size\_t n) const**

Converts from Unicode to UTF-32 encoding. Returns the size of the destination buffer.

**wxMDIChildFrame**

An MDI child frame is a frame that can only exist on a *wxMDIClientWindow* (p. 1050), which is itself a child of *wxMDIParentFrame* (p. 1051).

**Derived from**

*wxFrame* (p. 682)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/mdi.h>

**Window styles**

**wxCAPTION** Puts a caption on the frame.

**wxDEFAULT\_FRAME\_STYLE** Defined as **wxMINIMIZE\_BOX | wxMAXIMIZE\_BOX | wxTHICK\_FRAME | wxSYSTEM\_MENU | wxCAPTION**.

**wxICONIZE** Display the frame iconized (minimized) (Windows only).

**wxMAXIMIZE** Displays the frame maximized (Windows only).

**wxMAXIMIZE\_BOX** Displays a maximize box on the frame (Windows and Motif only).

**wxMINIMIZE** Identical to **wxICONIZE**.

**wxMINIMIZE\_BOX** Displays a minimize box on the frame (Windows and Motif only).

**wxRESIZE\_BORDER** Displays a resizable border around the window (Motif only; for Windows, it is implicit in **wxTHICK\_FRAME**).

**wxSTAY\_ON\_TOP** Stay on top of other windows (Windows only).

**wxSYSTEM\_MENU** Displays a system menu (Windows and Motif only).

**wxTHICK\_FRAME** Displays a thick frame around the window (Windows and Motif only).

See also *window styles overview* (p. 2089).

## Remarks

Although internally an MDI child frame is a child of the MDI client window, in wxWidgets you create it as a child of *wxMDIParentFrame* (p. 1051). You can usually forget that the client window exists.

MDI child frames are clipped to the area of the MDI client window, and may be iconized on the client window.

You can associate a menubar with a child frame as usual, although an MDI child doesn't display its menubar under its own title bar. The MDI parent frame's menubar will be changed to reflect the currently active child frame. If there are currently no children, the parent frame's own menubar will be displayed.

## See also

*wxMDIClientWindow* (p. 1050), *wxMDIParentFrame* (p. 1051), *wxFrame* (p. 682)

## wxMDIChildFrame::wxMDIChildFrame

### wxMDIChildFrame()

Default constructor.

**wxMDIChildFrame(wxMDIParentFrame\* parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT\_FRAME\_STYLE, const wxString& name = "frame")**

Constructor, creating the window.

## Parameters

*parent*

The window parent. This should not be NULL.

*id*

The window identifier. It may take a value of -1 to indicate a default value.

*title*

The caption to be displayed on the frame's title bar.

*pos*

The window position. A value of (-1, -1) indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform.

*size*

The window size. A value of (-1, -1) indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform.



*style*

The window style. See *wxMDIChildFrame* (p. 1047).

*name*

The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

#### **Remarks**

None.

#### **See also**

*wxMDIChildFrame::Create* (p. 1049)

#### **wxMDIChildFrame::~~wxMDIChildFrame**

**~wxMDIChildFrame()**

Destructor. Destroys all child windows and menu bar if present.

#### **wxMDIChildFrame::Activate**

**void Activate()**

Activates this MDI child frame.

#### **See also**

*wxMDIChildFrame::Maximize* (p. 1049), *wxMDIChildFrame::Restore* (p. 1050)

#### **wxMDIChildFrame::Create**

**bool Create(wxWindow\* parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT\_FRAME\_STYLE, const wxString& name = "frame")**

Used in two-step frame construction. See *wxMDIChildFrame::wxMDIChildFrame* (p. 1048) for further details.

#### **wxMDIChildFrame::Maximize**

**void Maximize(bool maximize)**

Maximizes this MDI child frame.

#### **See also**

*wxMDIChildFrame::Activate* (p. 1049), *wxMDIChildFrame::Restore* (p. 1050)

**wxMDIChildFrame::Restore****void Restore()**

Restores this MDI child frame (unmaximizes).

**See also**

*wxMDIChildFrame::Activate* (p. 1049), *wxMDIChildFrame::Maximize* (p. 1049)

**wxMDIClientWindow**

An MDI client window is a child of *wxMDIParentFrame* (p. 1051), and manages zero or more *wxMDIChildFrame* (p. 1047) objects.

**Derived from**

*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/mdi.h>

**Remarks**

The client window is the area where MDI child windows exist. It doesn't have to cover the whole parent frame; other windows such as toolbars and a help window might coexist with it. There can be scrollbars on a client window, which are controlled by the parent window style.

The **wxMDIClientWindow** class is usually adequate without further derivation, and it is created automatically when the MDI parent frame is created. If the application needs to derive a new class, the function *wxMDIParentFrame::OnCreateClient* (p. 1056) must be overridden in order to give an opportunity to use a different class of client window.

Under Windows 95, the client window will automatically have a sunken border style when the active child is not maximized, and no border style when a child is maximized.

**See also**

*wxMDIChildFrame* (p. 1047), *wxMDIParentFrame* (p. 1051), *wxFrame* (p. 682)

**wxMDIClientWindow::wxMDIClientWindow****wxMDIClientWindow()**

Default constructor.

**wxMDIClientWindow(wxMDIParentFrame\* parent, long style = 0)**

Constructor, creating the window.

### Parameters

*parent*

The window parent.

*style*

The window style. Currently unused.

### Remarks

The second style of constructor is called within *wxMDIParentFrame::OnCreateClient* (p. 1056).

### See also

*wxMDIParentFrame::wxMDIParentFrame* (p. 1053),  
*wxMDIParentFrame::OnCreateClient* (p. 1056)

### **wxMDIClientWindow::~~wxMDIClientWindow**

**~wxMDIClientWindow()**

Destructor.

### **wxMDIClientWindow::CreateClient**

**bool CreateClient(wxMDIParentFrame\* parent, long style = 0)**

Used in two-step frame construction. See *wxMDIClientWindow::wxMDIClientWindow* (p. 1050) for further details.

## **wxMDIParentFrame**

An MDI (Multiple Document Interface) parent frame is a window which can contain MDI child frames in its own 'desktop'. It is a convenient way to avoid window clutter, and is used in many popular Windows applications, such as Microsoft Word(TM).

### Derived from

*wxFrame* (p. 682)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/mdi.h>

## Remarks

There may be multiple MDI parent frames in a single application, but this probably only makes sense within programming development environments.

Child frames may be of class *wxMDIChildFrame* (p. 1047) (contained within the parent frame) or *wxFrame* (p. 682) (shown as a top-level frame).

An MDI parent frame always has a *wxMDIClientWindow* (p. 1050) associated with it, which is the parent for MDI child frames. This client window may be resized to accommodate non-MDI windows, as seen in Microsoft Visual C++ (TM) and Microsoft Publisher (TM), where a documentation window is placed to one side of the workspace.

MDI remains popular despite dire warnings from Microsoft itself that MDI is an obsolete user interface style.

The implementation is native in Windows, and simulated under Motif. Under Motif, the child window frames will often have a different appearance from other frames because the window decorations are simulated.

## Window styles

<b>wxCAPTION</b>	Puts a caption on the frame.
<b>wxDEFAULT_FRAME_STYLE</b>	Defined as <b>wxMINIMIZE_BOX   wxMAXIMIZE_BOX   wxTHICK_FRAME   wxSYSTEM_MENU   wxCAPTION</b> .
<b>wxHSCROLL</b>	Displays a horizontal scrollbar in the <i>client window</i> , allowing the user to view child frames that are off the current view.
<b>wxICONIZE</b>	Display the frame iconized (minimized) (Windows only).
<b>wxMAXIMIZE</b>	Displays the frame maximized (Windows only).
<b>wxMAXIMIZE_BOX</b>	Displays a maximize box on the frame (Windows and Motif only).
<b>wxMINIMIZE</b>	Identical to <b>wxICONIZE</b> .
<b>wxMINIMIZE_BOX</b>	Displays a minimize box on the frame (Windows and Motif only).
<b>wxRESIZE_BORDER</b>	Displays a resizeable border around the window (Motif only; for Windows, it is implicit in <b>wxTHICK_FRAME</b> ).
<b>wxSTAY_ON_TOP</b>	Stay on top of other windows (Windows only).
<b>wxSYSTEM_MENU</b>	Displays a system menu (Windows and Motif only).
<b>wxTHICK_FRAME</b>	Displays a thick frame around the window (Windows and Motif only).
<b>wxVSCROLL</b>	Displays a vertical scrollbar in the <i>client window</i> , allowing the user to view child frames that are off the current view.

**wxFRAME\_NO\_WINDOW\_MENU** Under Windows, removes the Window menu that is normally added automatically.

See also *window styles overview* (p. 2089).

**See also**

*wxMDIChildFrame* (p. 1047), *wxMDIClientWindow* (p. 1050), *wxFrame* (p. 682), *wxDialog* (p. 496)

## **wxMDIParentFrame::wxMDIParentFrame**

### **wxMDIParentFrame()**

Default constructor.

**wxMDIParentFrame(wxWindow\* parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT\_FRAME\_STYLE | wxVSCROLL | wxHSCROLL, const wxString& name = "frame")**

Constructor, creating the window.

### **Parameters**

*parent*

The window parent. This should be NULL.

*id*

The window identifier. It may take a value of -1 to indicate a default value.

*title*

The caption to be displayed on the frame's title bar.

*pos*

The window position. A value of (-1, -1) indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform.

*size*

The window size. A value of (-1, -1) indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform.

*style*

The window style. See *wxMDIParentFrame* (p. 1051).

*name*

The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

**Remarks**

During the construction of the frame, the client window will be created. To use a different class from *wxMDIClientWindow* (p. 1050), override *wxMDIParentFrame::OnCreateClient* (p. 1056).

Under Windows 95, the client window will automatically have a sunken border style when the active child is not maximized, and no border style when a child is maximized.

**See also**

*wxMDIParentFrame::Create* (p. 1055), *wxMDIParentFrame::OnCreateClient* (p. 1056)

**wxMDIParentFrame::~~wxMDIParentFrame**

**~wxMDIParentFrame()**

Destructor. Destroys all child windows and menu bar if present.

**wxMDIParentFrame::ActivateNext**

**void ActivateNext()**

Activates the MDI child following the currently active one.

**See also**

*wxMDIParentFrame::ActivatePrevious* (p. 1054)

**wxMDIParentFrame::ActivatePrevious**

**void ActivatePrevious()**

Activates the MDI child preceding the currently active one.

**See also**

*wxMDIParentFrame::ActivateNext* (p. 1054)

**wxMDIParentFrame::ArrangeIcons**

**void ArrangeIcons()**

Arranges any iconized (minimized) MDI child windows.

**See also**

*wxMDIParentFrame::Cascade* (p. 1055), *wxMDIParentFrame::Tile* (p. 1058)

**wxMDIParentFrame::Cascade****void Cascade()**

Arranges the MDI child windows in a cascade.

**See also**

*wxMDIParentFrame::Tile* (p. 1058), *wxMDIParentFrame::ArrangeIcons* (p. 1054)

**wxMDIParentFrame::Create**

**bool Create**(*wxWindow\* parent*, *wxWindowID id*, **const wxString& title**, **const wxPoint& pos** = *wxDefaultPosition*, **const wxSize& size** = *wxDefaultSize*, **long style** = *wxDEFAULT\_FRAME\_STYLE | wxVSCROLL | wxHSCROLL*, **const wxString& name** = *"frame"*)

Used in two-step frame construction. See *wxMDIParentFrame::wxMDIParentFrame* (p. 1053) for further details.

**wxMDIParentFrame::GetClientSize**

**virtual void GetClientSize**(*int\* width*, *int\* height*) **const**

This gets the size of the frame 'client area' in pixels.

**Parameters**

*width*

Receives the client width in pixels.

*height*

Receives the client height in pixels.

**Remarks**

The client area is the area which may be drawn on by the programmer, excluding title bar, border, status bar, and toolbar if present.

If you wish to manage your own toolbar (or perhaps you have more than one), provide an **OnSize** event handler. Call **GetClientSize** to find how much space there is for your windows and don't forget to set the size and position of the MDI client window as well as your toolbar and other windows (but not the status bar).

If you have set a toolbar with *wxMDIParentFrame::SetToolbar* (p. 1057), the client size returned will have subtracted the toolbar height. However, the available positions for the client window and other windows of the frame do not start at zero - you must add the toolbar height.

The position and size of the status bar and toolbar (if known to the frame) are always managed by **wxMDIParentFrame**, regardless of what behaviour is defined in your

**OnSize** event handler. However, the client window position and size are always set in **OnSize**, so if you override this event handler, make sure you deal with the client window.

You do not have to manage the size and position of MDI child windows, since they are managed automatically by the client window.

**See also**

*wxMDIParentFrame::GetToolBar* (p. 1056), *wxMDIParentFrame::SetToolBar* (p. 1057), *wxMDIClientWindow* (p. 1050)

**wxPython note:** The wxPython version of this method takes no arguments and returns a tuple containing width and height.

**wxMDIParentFrame::GetActiveChild**

**wxMDIChildFrame\* GetActiveChild() const**

Returns a pointer to the active MDI child, if there is one.

**wxMDIParentFrame::GetClientWindow**

**wxMDIClientWindow\* GetClientWindow() const**

Returns a pointer to the client window.

**See also**

*wxMDIParentFrame::OnCreateClient* (p. 1056)

**wxMDIParentFrame::GetToolBar**

**virtual wxWindow\* GetToolBar() const**

Returns the window being used as the toolbar for this frame.

**See also**

*wxMDIParentFrame::SetToolBar* (p. 1057)

**wxMDIParentFrame::GetWindowMenu**

**wxMenu\* GetWindowMenu() const**

Returns the current Window menu (added by wxWidgets to the menubar). This function is available under Windows only.

**wxMDIParentFrame::OnCreateClient**

**virtual wxMDIClientWindow\* OnCreateClient()**

Override this to return a different kind of client window. If you override this function, you



must create your parent frame in two stages, or your function will never be called, due to the way C++ treats virtual functions called from constructors. For example:

```
frame = new MyParentFrame;  
frame->Create(parent, myParentFrameId, wxT("My Parent Frame"));
```

### Remarks

You might wish to derive from *wxMDIClientWindow* (p. 1050) in order to implement different erase behaviour, for example, such as painting a bitmap on the background.

Note that it is probably impossible to have a client window that scrolls as well as painting a bitmap or pattern, since in **OnScroll**, the scrollbar positions always return zero. (Solutions to: [julian.smart@btopenworld.com](mailto:julian.smart@btopenworld.com)).

### See also

*wxMDIParentFrame::GetClientWindow* (p. 1056), *wxMDIClientWindow* (p. 1050)

### **wxMDIParentFrame::SetToolBar**

**virtual void SetToolBar**(*wxWindow\* toolbar*)

Sets the window to be used as a toolbar for this MDI parent window. It saves the application having to manage the positioning of the toolbar MDI client window.

### Parameters

*toolbar*

Toolbar to manage.

### Remarks

When the frame is resized, the toolbar is resized to be the width of the frame client area, and the toolbar height is kept the same.

The parent of the toolbar must be this frame.

If you wish to manage your own toolbar (or perhaps you have more than one), don't call this function, and instead manage your subwindows and the MDI client window by providing an **OnSize** event handler. Call *wxMDIParentFrame::GetClientSize* (p. 1055) to find how much space there is for your windows.

Note that SDI (normal) frames and MDI child windows must always have their toolbars managed by the application.

### See also

*wxMDIParentFrame::GetToolBar* (p. 1056), *wxMDIParentFrame::GetClientSize* (p. 1055)

### **wxMDIParentFrame::SetWindowMenu**

**void SetWindowMenu**(*wxMenu\* menu*)

Call this to change the current Window menu. Ownership of the menu object passes to the frame when you call this function.

This call is available under Windows only.

To remove the window completely, use the `wxFRAME_NO_WINDOW_MENU` window style.

### **`wxMDIParentFrame::Tile`**

**`void Tile(wxOrientation orient = wxHORIZONTAL)`**

Tiles the MDI child windows either horizontally or vertically depending on whether *orient* is `wxHORIZONTAL` or `wxVERTICAL`.

Currently only implemented for MSW, does nothing under the other platforms.

#### **See also**

*`wxMDIParentFrame::Cascade`* (p. 1055), *`wxMDIParentFrame::ArrangeIcons`* (p. 1054)

## **`wxMediaCtrl`**

`wxMediaCtrl` is a class for displaying types of media, such as videos, audio files, natively through native codecs.

`wxMediaCtrl` uses native backends to render media, for example on Windows there is a `ActiveMovie/DirectShow` backend, and on Macintosh there is a `QuickTime` backend.

#### **See also**

*`wxMediaEvent`* (p. 1066)

#### **Derived from**

*`wxControl`* (p. 285)

#### **Include files**

`<wx/mediactrl.h>`

### **Rendering media**

Depending upon the backend, `wxMediaCtrl` can render and display pretty much any kind of media that the native system can - such as an image, mpeg video, or mp3 (without license restrictions - since it relies on native system calls that may not technically have mp3 decoding available, for example, it falls outside the realm of licensing restrictions).

For general operation, all you need to do is call *`wxMediaCtrl::Load`* (p. 1063) to load the file you want to render, catch the `EVT_MEDIA_LOADED` event, and then call

*wxMediaCtrl::Play* (p. 1064) to show the video/audio of the media in that event.

More complex operations are generally more heavily dependant on the capabilities of the backend. For example, QuickTime cannot set the playback rate of certain streaming media - while DirectShow is slightly more flexible in that regard.

## Operation

When *wxMediaCtrl* plays a file, it plays until the stop position is reached (currently the end of the file/stream). Right before it hits the end of the stream, it fires off a *EVT\_MEDIA\_STOP* event to its parent window, at which point the event handler can choose to veto the event, preventing the stream from actually stopping.

```
Example://connect to the media event
this->Connect(wxMY_ID, wxEVT_MEDIA_STOP, (wxObjectEventFunction)
(wxEventFunction)(wxMediaEventFunction) &MyFrame::OnMediaStop);

//...
void MyFrame::OnMediaStop(const wxMediaEvent& evt)
{
    if(bUserWantsToSeek)
    {
        m_mediactrl->SetPosition(
            m_mediactrl->GetDuration() << 1
            );
        evt.Veto();
    }
}
```

When *wxMediaCtrl* stops, either by the *EVT\_MEDIA\_STOP* not being vetoed, or by manually calling *wxMediaCtrl::Stop* (p. 1065), where it actually stops is not at the beginning, rather, but at the beginning of the stream. That is, when it stops and play is called, playback is guaranteed to start at the beginning of the media. This is because some streams are not seekable, and when stop is called on them they return to the beginning, thus *wxMediaCtrl* tries to keep consistant for all types of media.

Note that when changing the state of the media through *Play()* and other methods, the media may not actually be in the *wxMEDIASTATE\_PLAYING*, for example. If you are relying on the media being in certain state catch the event relevant to the state. See *wxMediaEvent* (p. 1066) for the kinds of events that you can catch.

## Video size

By default, *wxMediaCtrl* will scale the size of the video to the requested amount passed to either it's constructor or *Create()*. After calling *Load* or performing an equivalent operation, you can subsequently obtain the "real" size of the video (if there is any) by calling *GetBestSize()*. Note that the actual result on the display will be slightly different when *ShowPlayerControls* is activated and the actual video size will be less then specified due to the extra controls provided by the native toolkit. In addition, the backend may modify *GetBestSize()* to include the size of the extra controls - so if you want the real size of the video just disable *ShowPlayerControls()*.

The idea with setting `GetBestSize` to the size of the video is that `GetBestSize` is a `wxWindow`-derived function that is called when sizers on a window recalculate. What this means is that if you use sizers by default the video will show in its original size without any extra assistance needed from the user.

## Player controls

Normally, when you use `wxMediaCtrl` it is just a window for the video to play in. However, some toolkits have their own media player interface. For example, QuickTime generally has a bar below the video with a slider. A special feature available to `wxMediaCtrl`, you can use the toolkit's interface instead of making your own by using the `ShowPlayerControls()` (p. 1065) function. There are several options for the flags parameter, with the two general flags being `wxMEDIACtrlPLAYERCONTROLS_NONE` which turns off the native interface, and `wxMEDIACtrlPLAYERCONTROLS_DEFAULT` which lets `wxMediaCtrl` decide what native controls on the interface. Be sure to review the caveats outlined in *Video size* (p. 1059) before doing so.

## Choosing a backend

Generally, you should almost certainly leave this part up to `wxMediaCtrl` - but if you need a certain backend for a particular reason, such as QuickTime for playing .mov files, all you need to do to choose a specific backend is to pass the name of the backend class to `wxMediaCtrl::Create` (p. 1062).

The following are valid backend identifiers - **`wxMEDIABACKEND_DIRECTSHOW`**

Use ActiveMovie/DirectShow. Uses the native ActiveMovie (I.E. DirectShow) control. Default backend on Windows and supported by nearly all Windows versions, even some Windows CE versions. May display a windows media player logo while inactive.

**`wxMEDIABACKEND_QUICKTIME`**

Use QuickTime. Mac Only. WARNING: May not working correctly embedded in a `wxNotebook`.

**`wxMEDIABACKEND_GSTREAMER`**

Use GStreamer. Unix Only. Requires GStreamer 0.8 along with at the very least the `xvimagesink`, `xoverlay`, and `gst-play` modules of gstreamer to function. You need the correct modules to play the relevant files, for example the `mad` module to play mp3s, etc.

**`wxMEDIABACKEND_WMP10`**

Uses Windows Media Player 10 (Windows only) - works on mobile machines with Windows Media Player 10 and desktop machines with either Windows Media Player 9 or 10

Note that other backends such as `wxMEDIABACKEND_MCI` can now be found at `wxCode`.

## Creating a backend

Creating a backend for `wxMediaCtrl` is a rather simple process. Simply derive from `wxMediaBackendCommonBase` and implement the methods you want. The methods in `wxMediaBackend` correspond to those in `wxMediaCtrl` except for `CreateControl` which does the actual creation of the control, in cases where a custom control is not needed you may simply call `wxControl::Create`.

You need to make sure to use the `DECLARE_CLASS` and `IMPLEMENT_CLASS` macros.

The only real tricky part is that you need to make sure the file is compiled in, which if there are just backends in there will not happen and you may need to use a force link hack (see <http://www.wxwidgets.org/wiki/index.php/RTTI>).

This is a rather simple example of how to create a backend in the `wxActiveXContainer` (p. 34) documentation.

### **wxMediaCtrl::wxMediaCtrl**

#### **wxMediaCtrl()**

Default constructor - you must call `Create` before calling any other methods of `wxMediaCtrl`.

```
wxMediaCtrl( wxWindow* parent, wxWindowID id, const wxString&  
fileName = wxT(""), const wxPoint& pos = wxDefaultPosition, const wxSize&  
size = wxDefaultSize, long style = 0, const wxString& szBackend = wxT(""),  
const wxValidator& validator = wxDefaultValidator, const wxString& name =  
wxPanelNameStr )
```

Constructor that calls `Create` (p. 1062). You may prefer to call `Create` (p. 1062) directly to check to see if `wxMediaCtrl` is available on the system.

*parent*

parent of this control. Must not be NULL.

*id*

id to use for events

*fileName*

If not empty, the path of a file to open.

*pos*

Position to put control at.

*size*

Size to put the control at and to stretch movie to.

*style*

Optional styles.

*szBackend*

Name of backend you want to use, leave blank to make wxMediaCtrl figure it out.

*validator*

validator to use.

*name*

Window name.

### **wxMediaCtrl::Create**

```
bool Create(    wxWindow* parent,    wxWindowID id,    const wxString&  
fileName = wxT(""),    const wxPoint& pos = wxDefaultPosition,    const wxSize&  
size = wxDefaultSize,    long style = 0,    const wxString& szBackend = wxT(""),  
const wxValidator& validator = wxDefaultValidator,    const wxString& name =  
wxPanelNameStr    )
```

Creates this control. Returns `false` if it can't load the movie located at `fileName` or it cannot load one of its native backends.

If you specify a file to open via `fileName` and you don't specify a backend to use, `wxMediaCtrl` tries each of its backends until one that can render the path referred to by `fileName` can be found.

*parent*

parent of this control. Must not be NULL.

*id*

id to use for events

*fileName*

If not empty, the path of a file to open.

*pos*

Position to put control at.

*size*

Size to put the control at and to stretch movie to.

*style*

Optional styles.

*szBackend*

Name of backend you want to use, leave blank to make wxMediaCtrl figure it out.

*validator*

validator to use.

*name*

Window name.

### **wxMediaCtrl::GetBestSize**

#### **wxSize GetBestSize()**

Obtains the best size relative to the original/natural size of the video, if there is any. See *Video size* (p. 1059) for more information.

### **wxMediaCtrl::GetPlaybackRate**

#### **double GetPlaybackrate()**

Obtains the playback rate, or speed of the media. 1.0 represents normal speed, while 2.0 represents twice the normal speed of the media, for example. Not supported on the GStreamer (Unix) backend. Returns 0 on failure.

### **wxMediaCtrl::GetVolume**

#### **double GetVolume()**

Gets the volume of the media from a 0.0 to 1.0 range. Note that due to rounding and other errors this may not be the exact value sent to `SetVolume`.

### **wxMediaCtrl::GetState**

#### **wxMediaCtrlState GetState()**

Obtains the state the playback of the media is in -

<b>wxMEDIASTATE_STOPPED</b>	The movie has stopped.
<b>wxMEDIASTATE_PAUSED</b>	The movie is paused.
<b>wxMEDIASTATE_PLAYING</b>	The movie is currently playing.

### **wxMediaCtrl::Length**

#### **wxFileOffset Length()**

Obtains the length - the total amount of time the movie has in milliseconds.

### **wxMediaCtrl::Load**

#### **bool Load(const wxString& fileName)**

Loads the file that `fileName` refers to. Returns false if loading fails.

### **wxMediaCtrl::Load**

**bool Load(const wxURI& uri)**

Loads the location that `uri` refers to. Note that this is very implementation-dependant, although HTTP URI/URLs are generally supported, for example. Returns false if loading fails.

### **wxMediaCtrl::Load**

**bool Load(const wxURI& uri, const wxURI& proxy)**

Loads the location that `uri` refers to with the proxy `proxy`. Not implemented on most backends so it should be called with caution. Returns false if loading fails.

### **wxMediaCtrl::LoadURI**

**bool LoadURI(const wxURI& uri)**

Same as *Load* (p. 1064). Kept for wxPython compatability.

### **wxMediaCtrl::LoadURIWithProxy**

**bool LoadURIWithProxy(const wxURI& uri, const wxURI& proxy)**

Same as *Load* (p. 1064). Kept for wxPython compatability.

### **wxMediaCtrl::Pause**

**bool Pause()**

Pauses playback of the movie.

### **wxMediaCtrl::Play**

**bool Play()**

Resumes playback of the movie.

### **wxMediaCtrl::Seek**

**wxFileOffset Seek(wxFileOffset where, wxSeekMode mode)**

Seeks to a position within the movie.

### **wxMediaCtrl::SetPlaybackRate**

**bool SetPlaybackRate(double dRate)**



Sets the playback rate, or speed of the media, to that referred by `dRate`. `1.0` represents normal speed, while `2.0` represents twice the normal speed of the media, for example. Not supported on the GStreamer (Unix) backend. Returns true if successful.

### **wxMediaCtrl::SetVolume**

**bool SetVolume(double dVolume)**

Sets the volume of the media from a `0.0` to `1.0` range to that referred by `dVolume`. `1.0` represents full volume, while `0.5` represents half (50 percent) volume, for example. Note that this may not be exact due to conversion and rounding errors, although setting the volume to full or none is always exact. Returns true if successful.

### **wxMediaCtrl::ShowPlayerControls**

**bool ShowPlayerControls(wxMediaCtrlPlayerControls flags =  
wxMEDIACtrlPLAYERCONTROLS\_DEFAULT)**

A special feature to `wxMediaCtrl`. Applications using native toolkits such as QuickTime usually have a scrollbar, play button, and more provided to them by the toolkit. By default `wxMediaCtrl` does not do this. However, on the directshow and quicktime backends you can show or hide the native controls provided by the underlying toolkit at will using `ShowPlayerControls`. Simply calling the function with default parameters tells `wxMediaCtrl` to use the default controls provided by the toolkit. The function takes `awxMediaCtrlPlayerControls` enumeration as follows:

**wxMEDIACtrlPLAYERCONTROLS\_NONE** No controls. return `wxMediaCtrl` to it's default state.

**wxMEDIACtrlPLAYERCONTROLS\_STEP** Step controls like fastforward, step one frame etc.

**wxMEDIACtrlPLAYERCONTROLS\_VOLUME** Volume controls like the speaker icon, volume slider, etc.

**wxMEDIACtrlPLAYERCONTROLS\_DEFAULT** Default controls for the toolkit. Currently a typedef for `wxMEDIACtrlPLAYERCONTROLS_STEP` and `wxMEDIACtrlPLAYERCONTROLS_VOLUME`.

For more see *Player controls* (p. 1060). Currently only implemented on the QuickTime and DirectShow backends. The function returns true on success.

### **wxMediaCtrl::Stop**

**bool Stop()**

Stops the media.

See *Operation* (p. 1059) for an overview of how stopping works.

## **wxMediaCtrl::Tell**

### **wxFileOffset Tell()**

Obtains the current position in time within the movie in milliseconds.

## **wxMediaEvent**

Event *wxMediaCtrl* (p. 1058) uses.

### **Derived from**

*wxNotifyEvent* (p. 1146)

### **Include files**

<wx/mediactrl.h>

### **Event table macros**

<b>EVT_MEDIA_LOADED(id, func)</b>	Sent when a media has loaded enough data that it can start playing.
<b>EVT_MEDIA_STOP(id, func)</b>	Send when a media has switched to the <code>wxMEDIASTATE_STOPPED</code> state. You may be able to Veto this event to prevent it from stopping, causing it to continue playing - even if it has reached that end of the media (note that this may not have the desired effect - if you want to loop the media, for example, catch the <code>EVT_MEDIA_FINISHED</code> and play there instead).
<b>EVT_MEDIA_FINISHED(id, func)</b>	Sent when a media has finished playing in a <i>wxMediaCtrl</i> (p. 1058).
<b>EVT_MEDIA_STATECHANGED(id, func)</b>	Send when a media has switched its state (from any media state).
<b>EVT_MEDIA_PLAY(id, func)</b>	Send when a media has switched to the <code>wxMEDIASTATE_PLAYING</code> state.
<b>EVT_MEDIA_PAUSE(id, func)</b>	Send when a media has switched to the <code>wxMEDIASTATE_PAUSED</code> state.

## **wxMemoryBuffer**

A **wxMemoryBuffer** is a useful data structure for storing arbitrary sized blocks of memory. `wxMemoryBuffer` guarantees deletion of the memory block when the object is destroyed.

**Derived from**

None

**Include files**

<wx/buffer.h>

**wxMemoryBuffer::wxMemoryBuffer**

**wxMemoryBuffer(const wxMemoryBuffer& src)**

Copy constructor, refcounting is used for performance , but `wxMemoryBuffer` is not a copy-on-write structure so changes made to one buffer effect all copies made from it.

**wxMemoryBuffer(size\_t size)**

Create a new buffer.

*size*

size of new buffer.

**wxMemoryBuffer::GetData**

**void\* GetData()**

Return a pointer to the data in the buffer.

**wxMemoryBuffer::GetBufSize**

**size\_t GetBufSize()**

Returns the size of the buffer.

**wxMemoryBuffer::GetDataLen**

**size\_t GetDataLen()**

Returns the length of the valid data in the buffer.

**wxMemoryBuffer::SetBufSize**

**void SetBufSize(size\_t size)**

Ensures the buffer has at least *size* bytes available.

**wxMemoryBuffer::SetDataLen****void SetDataLen(size\_t size)**

Sets the length of the data stored in the buffer. Mainly useful for truncating existing data.

*size*

New length of the valid data in the buffer. This is distinct from the allocated size

**wxMemoryBuffer::GetWriteBuf****void \* GetWriteBuf(size\_t sizeNeeded)**

Ensure the buffer is big enough and return a pointer to the buffer which can be used to directly write into the buffer up to *sizeNeeded* bytes.

**wxMemoryBuffer::UngetWriteBuf****void UngetWriteBuf(size\_t sizeUsed)**

Update the buffer after completing a direct write, which you must have used `GetWriteBuf()` to initialise.

*sizeUsed*

The amount of data written in to buffer by the direct write

**wxMemoryBuffer::GetAppendBuf****void \* GetAppendBuf(size\_t sizeNeeded)**

Ensure that the buffer is big enough and return a pointer to the start of the empty space in the buffer. This pointer can be used to directly write data into the buffer, this new data will be appended to the existing data.

*sizeNeeded*

Amount of extra space required in the buffer for the append operation

**wxMemoryBuffer::UngetAppendBuf****void UngetAppendBuf(size\_t sizeUsed)**

Update the length after completing a direct append, which you must have used `GetAppendBuf()` to initialise.

*sizeUsed*

This is the amount of new data that has been appended.

**wxMemoryBuffer::AppendByte**

**void AppendByte(char data)**

Append a single byte to the buffer.

*data*

New byte to append to the buffer.

**wxMemoryBuffer::AppendData****void AppendData(void\* data, size\_t len)**

Single call to append a data block to the buffer.

*data*

Pointer to block to append to the buffer.

*len*

Length of data to append.

**wxMemoryDC**

A memory device context provides a means to draw graphics onto a bitmap. When drawing in to a mono-bitmap, using `wxWHITE`, `wxWHITE_PEN` and `wxWHITE_BRUSH` will draw the background colour (i.e. 0) whereas all other colours will draw the foreground colour (i.e. 1).

**Derived from**

*wxDC* (p. 456)

*wxObject* (p. 1148)

**Include files**

<wx/dcmemory.h>

**Remarks**

A bitmap must be selected into the new memory DC before it may be used for anything. Typical usage is as follows:

```
// Create a memory DC
wxMemoryDC temp_dc;
temp_dc.SelectObject(test_bitmap);

// We can now draw into the memory DC...
// Copy from this DC to another DC.
old_dc.Blit(250, 50, BITMAP_WIDTH, BITMAP_HEIGHT, temp_dc, 0, 0);
```

Note that the memory DC *must* be deleted (or the bitmap selected out of it) before a bitmap can be reselected into another memory DC.

**See also**

*wxBitmap* (p. 123), *wxDC* (p. 456)

**wxMemoryDC::wxMemoryDC****wxMemoryDC()**

Constructs a new memory device context.

Use the *IsOk* (p. 472) member to test whether the constructor was successful in creating a usable device context. Don't forget to select a bitmap into the DC before drawing on it.

**wxMemoryDC(wxBitmap& bitmap)**

Constructs a new memory device context and calls *SelectObject* (p. 1070) with the given bitmap. Use the *IsOk* (p. 472) member to test whether the constructor was successful in creating a usable device context.

**wxMemoryDC::SelectObject****void SelectObject(wxBitmap& bitmap)**

Works exactly like *SelectObjectAsSource* (p. 1070) but this is the function you should use when you select a bitmap because you want to modify it, e.g. drawing on this DC.

Be careful to use this function and not *SelectObjectAsSource* (p. 1070) when you want to modify the bitmap you are selecting otherwise you may incur in some problems related to *wxBitmap* being a reference counted object (see *reference counting overview* (p. 2046)).

**See also**

*wxDC::DrawBitmap* (p. 460)

**wxMemoryDC::SelectObjectAsSource****void SelectObjectAsSource(const wxBitmap& bitmap)**

Selects the given bitmap into the device context, to use as the memory bitmap. Selecting the bitmap into a memory DC allows you to draw into the DC (and therefore the bitmap) and also to use *wxDC::Blit* (p. 457) to copy the bitmap to a window. For this purpose, you may find *wxDC::DrawIcon* (p. 461) easier to use instead.

If the argument is *wxNullBitmap* (or some other uninitialised *wxBitmap*) the current bitmap is selected out of the device context, and the original bitmap restored, allowing the current bitmap to be destroyed safely.

**See also**

*wxMemoryDC::SelectObject* (p. 1070)

## wxMemoryFSHandler

This *wxFileSystem* (p. 633) handler can store arbitrary data in memory stream and make them accessible via URL. It is particularly suitable for storing bitmaps from resources or included XPM files so that they can be used with wxHTML.

Filenames are prefixed with "memory:", e.g. "memory:myfile.html".

Example:

```
#ifndef __WXMSW__
#include "logo.xpm"
#endif

void MyFrame::OnAbout(wxCommandEvent&)
{
    wxBusyCursor bcur;
    wxFileSystem::AddHandler(new wxMemoryFSHandler);
    wxMemoryFSHandler::AddFile("logo.pcx", wxBITMAP(logo),
wxBITMAP_TYPE_PCX);
    wxMemoryFSHandler::AddFile("about.htm",
                                "<html><body>About: "
                                "<img
src=\"memory:logo.pcx\"></body></html>");

    wxDialog dlg(this, -1, wxString(_("About")));
    wxBoxSizer *topsizer;
    wxHtmlWindow *html;
    topsizer = new wxBoxSizer(wxVERTICAL);
    html = new wxHtmlWindow(&dlg, -1, wxDefaultPosition,
                            wxSize(380, 160),
wxHW_SCROLLBAR_NEVER);
    html->SetBorders(0);
    html->LoadPage("memory:about.htm");
    html->SetSize(html->GetInternalRepresentation()->GetWidth(),
                  html->GetInternalRepresentation()->GetHeight());
    topsizer->Add(html, 1, wxALL, 10);
    topsizer->Add(new wxStaticLine(&dlg, -1), 0, wxEXPAND | wxLEFT |
wxRIGHT, 10);
    topsizer->Add(new wxButton(&dlg, wxID_OK, "Ok"),
                  0, wxALL | wxALIGN_RIGHT, 15);
    dlg.SetAutoLayout(true);
    dlg.SetSizer(topsizer);
    topsizer->Fit(&dlg);
    dlg.Centre();
    dlg.ShowModal();

    wxMemoryFSHandler::RemoveFile("logo.pcx");
    wxMemoryFSHandler::RemoveFile("about.htm");
}
```

### Derived from

*wxFileSystemHandler* (p. 636)

### Include files

<wx/fs\_mem.h>

## **wxMemoryFSHandler::wxMemoryFSHandler**

**wxMemoryFSHandler()**

Constructor.

## **wxMemoryFSHandler::AddFile**

**static void AddFile(const wxString& filename, wxImage& image, long type)**

**static void AddFile(const wxString& filename, const wxBitmap& bitmap, long type)**

Add file to list of files stored in memory. Stored data (bitmap, text or raw data) will be copied into private memory stream and available under name "memory:" + *filename*.

The *type* argument is one of `wxBITMAP_TYPE_XXX` constants. Note that you must use a *type* value (aka image format) that wxWidgets can save (e.g. JPG, PNG, see *wxImage documentation* (p. 906))!

### **See also**

*AddFileWithMimeType* (p. 1072)

## **wxMemoryFSHandler::AddFileWithMimeType**

**static void AddFileWithMimeType(const wxString& filename, const wxString& textdata, const wxString& mimetype)**

**static void AddFileWithMimeType(const wxString& filename, const void\* binarydata, size\_t size, const wxString& mimetype)**

Like *AddFile* (p. 1072), but lets you explicitly specify added file's MIME type. This version should be used whenever you know the MIME type, because it makes accessing the files faster.

This function is new since wxWidgets version 2.8.5

### **See also**

*AddFile* (p. 1072)

## **wxMemoryFSHandler::RemoveFile**

**static void RemoveFile(const wxString& filename)**

Remove file from memory FS and free occupied memory.



## **wxMemoryInputStream**

### **Derived from**

*wxInputStream* (p. 941)

### **Include files**

<wx/mstream.h>

### **See also**

*wxStreamBuffer* (p. 1547), *wxMemoryOutputStream* (p. 1073)

## **wxMemoryInputStream::wxMemoryInputStream**

**wxMemoryInputStream(const char \* data, size\_t len)**

Initializes a new read-only memory stream which will use the specified buffer *data* of length *len*. The stream does not take ownership of the buffer, i.e. the buffer will not be deleted in its destructor.

**wxMemoryInputStream(const wxMemoryOutputStream& stream)**

Creates a new read-only memory stream, initializing it with the data from the given output stream *stream*.

## **wxMemoryInputStream::~~wxMemoryInputStream**

**~wxMemoryInputStream()**

Destructor.

## **wxMemoryInputStream::GetInputStreamBuffer**

**wxStreamBuffer \* GetInputStreamBuffer() const**

Returns the pointer to the stream object used as an internal buffer for that stream.

## **wxMemoryOutputStream**

### **Derived from**

*wxOutputStream* (p. 1156)

### **Include files**

<wx/mstream.h>

### **See also**

*wxStreamBuffer* (p. 1547)

### **wxMemoryOutputStream::wxMemoryOutputStream**

**wxMemoryOutputStream(char \* data = NULL, size\_t length = 0)**

If *data* is NULL, then it will initialize a new empty buffer which will grow if required.

#### **Warning**

If the buffer is created, it will be destroyed at the destruction of the stream.

### **wxMemoryOutputStream::~wxMemoryOutputStream**

**~wxMemoryOutputStream()**

Destructor.

### **wxMemoryOutputStream::CopyTo**

**size\_t CopyTo(char \*buffer, size\_t len) const**

CopyTo allowed you to transfer data from the internal buffer of wxMemoryOutputStream to an external buffer. *len* specifies the size of the buffer.

#### **Returned value**

CopyTo returns the number of bytes copied to the buffer. Generally it is either *len* or the size of the stream buffer.

### **wxMemoryOutputStream::GetOutputStreamBuffer**

**wxStreamBuffer \* GetOutputStreamBuffer() const**

Returns the pointer to the stream object used as an internal buffer for that stream.

## **wxMenu**

A menu is a popup (or pull down) list of items, one of which may be selected before the menu goes away (clicking elsewhere dismisses the menu). Menus may be used to construct either menu bars or popup menus.

A menu item has an integer ID associated with it which can be used to identify the selection, or to change the menu item in some way. A menu item with a special identifier -1 is a separator item and doesn't have an associated command but just makes a separator line appear in the menu.

**NB:** Please note that *wxID\_ABOUT* and *wxID\_EXIT* are predefined by wxWidgets and have a special meaning since entries using these IDs will be taken out of the normal

menus under MacOS X and will be inserted into the system menu (following the appropriate MacOS X interface guideline). On PalmOS *wxID\_EXIT* is disabled according to Palm OS Companion guidelines.

Menu items may be either normal items, check items or radio items. Normal items don't have any special properties while the check items have a boolean flag associated to them and they show a checkmark in the menu when the flag is set. *wxWidgets* automatically toggles the flag value when the item is clicked and its value may be retrieved using either *IsChecked* (p. 1084) method of *wxMenu* or *wxMenuBar* itself or by using *wxEvent::IsChecked* (p. 255) when you get the menu notification for the item in question.

The radio items are similar to the check items except that all the other items in the same radio group are unchecked when a radio item is checked. The radio group is formed by a contiguous range of radio items, i.e. it starts at the first item of this kind and ends with the first item of a different kind (or the end of the menu). Notice that because the radio groups are defined in terms of the item positions inserting or removing the items in the menu containing the radio items risks to not work correctly. Finally note that radio items are not supported under Motif.

### Allocation strategy

All menus except the popup ones must be created on the heap. All menus attached to a menubar or to another menu will be deleted by their parent when it is deleted. As the frame menubar is deleted by the frame itself, it means that normally all menus used are deleted automatically.

### Derived from

*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/menu.h>

### Event handling

If the menu is part of a menubar, then *wxMenuBar* (p. 1088) event processing is used.

With a popup menu, there is a variety of ways to handle a menu selection event (*wxEVT\_COMMAND\_MENU\_SELECTED*).

1. Derive a new class from *wxMenu* and define event table entries using the *EVT\_MENU* macro.
2. Set a new event handler for *wxMenu*, using an object whose class has *EVT\_MENU* entries.
3. Provide *EVT\_MENU* handlers in the window which pops up the menu, or in an ancestor of this window.
4. Define a callback of type *wxFunction*, which you pass to the *wxMenu* constructor. The callback takes a reference to the menu, and a reference to *awxCommandEvent* (p. 250). This method is deprecated and should not be used

in the new code, it is provided for backwards compatibility only.

**See also**

*wxMenuBar* (p. 1088), *wxWindow::PopupMenu* (p. 1828), *Event handling overview* (p. 2077), *wxFileHistory* (*most recently used files menu*) (p. 605)

**wxMenu::wxMenu**

**wxMenu(const wxString& title = "", long style = 0)**

Constructs a wxMenu object.

**Parameters**

*title*

A title for the popup menu: the empty string denotes no title.

*style*

If set to `wxMENU_TEAROFF`, the menu will be detachable (wxGTK only).

**wxMenu(long style)**

Constructs a wxMenu object.

**Parameters**

*style*

If set to `wxMENU_TEAROFF`, the menu will be detachable (wxGTK only).

**wxMenu::~~wxMenu**

**~wxMenu()**

Destructor, destroying the menu.

Note: under Motif, a popup menu must have a valid parent (the window it was last popped up on) when being destroyed. Therefore, make sure you delete or re-use the popup menu *before* destroying the parent window. Re-use in this context means popping up the menu on a different window from last time, which causes an implicit destruction and recreation of internal data structures.

**wxMenu::Append**

**wxMenuItem\* Append(int id, const wxString& item = "", const wxString& helpString = "", wxItemKind kind = wxITEM\_NORMAL)**

Adds a string item to the end of the menu.

**wxMenuItem\* Append(int id, const wxString& item, wxMenu\* subMenu, const wxString& helpString = "")**

Adds a pull-right submenu to the end of the menu. Append the submenu to the parent menu *after* you have added your menu items, or accelerators may not be registered properly.

**wxMenuItem\* Append(wxMenuItem\* menuItem)**

Adds a menu item object. This is the most generic variant of Append() method because it may be used for both items (including separators) and submenus and because you can also specify various extra properties of a menu item this way, such as bitmaps and fonts.

### Parameters

*id*

The menu command identifier.

*item*

The string to appear on the menu item.

*menu*

Pull-right submenu.

*kind*

May be wxITEM\_SEPARATOR, wxITEM\_NORMAL, wxITEM\_CHECK or wxITEM\_RADIO

*helpString*

An optional help string associated with the item. By default, the handler for the wxEVT\_MENU\_HIGHLIGHT event displays this string in the status line.

*menuItem*

A menuitem object. It will be owned by the wxMenu object after this function is called, so do not delete it yourself.

### Remarks

This command can be used after the menu has been shown, as well as on initial creation of a menu or menubar.

The *item* string for the normal menu items (not submenus or separators) may include the accelerator which can be used to activate the menu item from keyboard. The accelerator string follows the item label and is separated from it by a TAB character ( '\t ' ). Its general syntax is any combination of "CTRL", "ALT" and "SHIFT" strings (case doesn't matter) separated by either ' - ' or ' + ' characters and followed by the accelerator itself. The accelerator may be any alphanumeric character, any function key (from F1 to F12) or one of the special characters listed in the table below (again, case doesn't matter):

DEL or DELETE	Delete key
INS or INSERT	Insert key
ENTER or RETURN	Enter key
PGUP	PageUp key
PGDN	PageDown key
LEFT	Left cursor arrow key
RIGHT	Right cursor arrow key
UP	Up cursor arrow key
DOWN	Down cursor arrow key
HOME	Home key
END	End key
SPACE	Space
TAB	Tab key
ESC or ESCAPE	Escape key (Windows only)

**See also**

*wxMenu::AppendSeparator* (p. 1079), *wxMenu::AppendCheckItem* (p. 1078), *wxMenu::AppendRadioItem* (p. 1079), *wxMenu::AppendSubMenu* (p. 1079), *wxMenu::Insert* (p. 1083), *wxMenu::SetLabel* (p. 1087), *wxMenu::GetHelpString* (p. 1082), *wxMenu::SetHelpString* (p. 1086), *wxMenuItem* (p. 1099)

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

**Append(id, string, helpStr="", checkable=false)**

**AppendMenu(id, string, aMenu, helpStr="")**

**AppendItem(aMenuItem)**

**wxMenu::AppendCheckItem**

**wxMenuItem\* AppendCheckItem(int id, const wxString& item, const wxString& helpString = "")**

Adds a checkable item to the end of the menu.

**See also**

*wxMenu::Append* (p. 1076), *wxMenu::InsertCheckItem* (p. 1084)

### **wxMenu::AppendRadioItem**

**wxMenuItem\* AppendRadioItem(int id, const wxString& item, const wxString& helpString = "")**

Adds a radio item to the end of the menu. All consequent radio items form a group and when an item in the group is checked, all the others are automatically unchecked.

#### **See also**

*wxMenu::Append* (p. 1076), *wxMenu::InsertRadioItem* (p. 1084)

### **wxMenu::AppendSeparator**

**wxMenuItem\* AppendSeparator()**

Adds a separator to the end of the menu.

#### **See also**

*wxMenu::Append* (p. 1076), *wxMenu::InsertSeparator* (p. 1084)

### **wxMenu::AppendSubMenu**

**wxMenuItem \* AppendSubMenu(wxMenu \*submenu, const wxString& text, const wxString& help = wxEmptyString)**

Adds the given *submenu* to this menu. *text* is the text shown in the menu for it and *help* is the help string shown in the status bar when the submenu item is selected.

### **wxMenu::Break**

**void Break()**

Inserts a break in a menu, causing the next appended item to appear in a new column.

### **wxMenu::Check**

**void Check(int id, const bool check)**

Checks or unchecks the menu item.

#### **Parameters**

*id*

The menu item identifier.

*check*

If true, the item will be checked, otherwise it will be unchecked.

**See also**

*wxMenu::IsChecked* (p. 1084)

**wxMenu::Delete**

**void Delete**(int *id*)

**void Delete**(wxMenuItem *\*item*)

Deletes the menu item from the menu. If the item is a submenu, it will **not** be deleted. Use *Destroy* (p. 1080) if you want to delete a submenu.

**Parameters**

*id*

Id of the menu item to be deleted.

*item*

Menu item to be deleted.

**See also**

*wxMenu::FindItem* (p. 1081), *wxMenu::Destroy* (p. 1080), *wxMenu::Remove* (p. 1086)

**wxMenu::Destroy**

**void Destroy**(int *id*)

**void Destroy**(wxMenuItem *\*item*)

Deletes the menu item from the menu. If the item is a submenu, it will be deleted. Use *Remove* (p. 1086) if you want to keep the submenu (for example, to reuse it later).

**Parameters**

*id*

Id of the menu item to be deleted.

*item*

Menu item to be deleted.

**See also**

*wxMenu::FindItem* (p. 1081), *wxMenu::Deletes* (p. 1080), *wxMenu::Remove* (p. 1086)

**wxMenu::Enable**



**void Enable(int *id*, const bool *enable*)**

Enables or disables (greys out) a menu item.

**Parameters**

*id*

The menu item identifier.

*enable*

true to enable the menu item, false to disable it.

**See also**

*wxMenu::IsEnabled* (p. 1085)

**wxMenu::FindItem**

**int FindItem(const wxString& *itemString*) const**

Finds the menu item id for a menu item string.

**wxMenuItem \* FindItem(int *id*, wxMenu \*\**menu* = NULL) const**

Finds the menu item object associated with the given menu item identifier and, optionally, the (sub)menu it belongs to.

**wxPerl note:** In wxPerl this method takes just the *id* parameter; in scalar context it returns the associated `Wx::MenuItem`, in list context it returns a 2-element list ( *item*, *submenu* )

**Parameters**

*itemString*

Menu item string to find.

*id*

Menu item identifier.

*menu*

If the pointer is not NULL, it will be filled with the item's parent menu (if the item was found)

**Return value**

First form: menu item identifier, or `wxNOT_FOUND` if none is found.

Second form: returns the menu item object, or NULL if it is not found.

**Remarks**

Any special menu codes are stripped out of source and target strings before matching.

**wxPython note:** The name of this method in wxPython is `FindItemById` and it does not support the second parameter.

### **wxMenu::FindItemByPosition**

**wxMenuItem\* FindItemByPosition(size\_t position) const**

Returns the wxMenuItem given a position in the menu.

### **wxMenu::GetHelpString**

**wxString GetHelpString(int id) const**

Returns the help string associated with a menu item.

#### **Parameters**

*id*

The menu item identifier.

#### **Return value**

The help string, or the empty string if there is no help string or the item was not found.

#### **See also**

*wxMenu::SetHelpString* (p. 1086), *wxMenu::Append* (p. 1076)

### **wxMenu::GetLabel**

**wxString GetLabel(int id) const**

Returns a menu item label, including any mnemonics and accelerators.

#### **Parameters**

*id*

The menu item identifier.

#### **Return value**

The item label, or the empty string if the item was not found.

#### **See also**

*wxMenu::GetLabelText* (p. 1082), *wxMenu::SetLabel* (p. 1087)

### **wxMenu::GetLabelText**

**wxString GetLabelText(int id) const**

Returns a menu item label, without any of the original mnemonics and accelerators.

**Parameters**

*id*

The menu item identifier.

**Return value**

The item label, or the empty string if the item was not found.

**See also**

*wxMenu::GetLabel* (p. 1082), *wxMenu::SetLabel* (p. 1087)

**wxMenu::GetMenuItemCount****size\_t GetMenuItemCount() const**

Returns the number of items in the menu.

**wxMenu::GetMenuItems****wxMenuItemList& GetMenuItems() const**

Returns the list of items in the menu. *wxMenuItemList* is a pseudo-template list class containing *wxMenuItem* pointers, see *wxList* (p. 966).

**wxMenu::GetTitle****wxString GetTitle() const**

Returns the title of the menu.

**Remarks**

This is relevant only to popup menus, use *wxMenuBar::GetMenuLabel* (p. 1093) for the menus in the menubar.

**See also**

*wxMenu::SetTitle* (p. 1087)

**wxMenu::Insert**

**wxMenuItem\* Insert(size\_t pos, wxMenuItem \*item)**

**wxMenuItem\* Insert(size\_t pos, int id, const wxString& item = "", const wxString& helpString = "", wxItemKind kind = wxITEM\_NORMAL)**

Inserts the given *item* before the position *pos*. Inserting the item at position *GetMenuItemCount* (p. 1083) is the same as appending it.

**See also**

*wxMenu::Append* (p. 1076), *wxMenu::Prepend* (p. 1085)

**wxMenu::InsertCheckItem**

**wxMenuItem\* InsertCheckItem(size\_t pos, int id, const wxString& item, const wxString& helpString = "")**

Inserts a checkable item at the given position.

**See also**

*wxMenu::Insert* (p. 1083), *wxMenu::AppendCheckItem* (p. 1078)

**wxMenu::InsertRadioItem**

**wxMenuItem\* InsertRadioItem(size\_t pos, int id, const wxString& item, const wxString& helpString = "")**

Inserts a radio item at the given position.

**See also**

*wxMenu::Insert* (p. 1083), *wxMenu::AppendRadioItem* (p. 1079)

**wxMenu::InsertSeparator**

**wxMenuItem\* InsertSeparator(size\_t pos)**

Inserts a separator at the given position.

**See also**

*wxMenu::Insert* (p. 1083), *wxMenu::AppendSeparator* (p. 1079)

**wxMenu::IsChecked**

**bool IsChecked(int id) const**

Determines whether a menu item is checked.

**Parameters**

*id*

The menu item identifier.

**Return value**

true if the menu item is checked, false otherwise.

**See also**

*wxMenu::Check* (p. 1079)

**wxMenu::IsEnabled**

**bool IsEnabled(int id) const**

Determines whether a menu item is enabled.

**Parameters**

*id*

The menu item identifier.

**Return value**

true if the menu item is enabled, false otherwise.

**See also**

*wxMenu::Enable* (p. 1080)

**wxMenu::Prepend**

**wxMenuItem\* Prepend(wxMenuItem \*item)**

**wxMenuItem\* Prepend(int id, const wxString& item = "", const wxString& helpString = "", wxItemKind kind = wxITEM\_NORMAL)**

Inserts the given *item* at position 0, i.e. before all the other existing items.

**See also**

*wxMenu::Append* (p. 1076), *wxMenu::Insert* (p. 1083)

**wxMenu::PrependCheckItem**

**wxMenuItem\* PrependCheckItem(int id, const wxString& item, const wxString& helpString = "")**

Inserts a checkable item at position 0.

**See also**

*wxMenu::Prepend* (p. 1085), *wxMenu::AppendCheckItem* (p. 1078)

**wxMenu::PrependRadioItem**

**wxMenuItem\* PrependRadioItem(int id, const wxString& item, const wxString&**

```
helpString = "")
```

Inserts a radio item at position 0.

**See also**

*wxMenu::Prepend* (p. 1085), *wxMenu::AppendRadioItem* (p. 1079)

**wxMenu::PrependSeparator**

**wxMenuItem\* PrependSeparator()**

Inserts a separator at position 0.

**See also**

*wxMenu::Prepend* (p. 1085), *wxMenu::AppendSeparator* (p. 1079)

**wxMenu::Remove**

**wxMenuItem \* Remove(int *id*)**

**wxMenuItem \* Remove(wxMenuItem \**item*)**

Removes the menu item from the menu but doesn't delete the associated C++ object. This allows to reuse the same item later by adding it back to the menu (especially useful with submenus).

**Parameters**

*id*

The identifier of the menu item to remove.

*item*

The menu item to remove.

**Return value**

The item which was detached from the menu.

**wxMenu::SetHelpString**

**void SetHelpString(int *id*, const wxString& *helpString*)**

Sets an item's help string.

**Parameters**

*id*

The menu item identifier.

*helpString*

The help string to set.

**See also**

*wxMenu::GetHelpString* (p. 1082)

**wxMenu::SetLabel**

**void SetLabel(int id, const wxString& label)**

Sets the label of a menu item.

**Parameters**

*id*

The menu item identifier.

*label*

The menu item label to set.

**See also**

*wxMenu::Append* (p. 1076), *wxMenu::GetLabel* (p. 1082)

**wxMenu::SetTitle**

**void SetTitle(const wxString& title)**

Sets the title of the menu.

**Parameters**

*title*

The title to set.

**Remarks**

This is relevant only to popup menus, use *wxMenuBar::SetLabelTop* (p. 1097) for the menus in the menubar.

**See also**

*wxMenu::GetTitle* (p. 1083)

**wxMenu::UpdateUI**

**void UpdateUI(wxEvtHandler\* source = NULL) const**

Sends events to *source* (or owning window if NULL) to update the menu UI. This is called

just before the menu is popped up with `wxWindow::PopupMenu` (p. 1828), but the application may call it at other times if required.

**See also**

`wxUpdateUIEvent` (p. 1752)

## **wxMenuBar**

A menu bar is a series of menus accessible from the top of a frame.

**Derived from**

`wxWindow` (p. 1795)

`wxEvtHandler` (p. 576)

`wxObject` (p. 1148)

**Include files**

`<wx/menu.h>`

**Event handling**

To respond to a menu selection, provide a handler for `EVT_MENU`, in the frame that contains the menu bar. If you have a toolbar which uses the same identifiers as your `EVT_MENU` entries, events from the toolbar will also be processed by your `EVT_MENU` event handlers.

**Tip:** under Windows, if you discover that menu shortcuts (for example, Alt-F to show the file menu) are not working, check any `EVT_CHAR` events you are handling in child windows. If you are not calling `event.Skip()` for events that you don't process in these event handlers, menu shortcuts may cease to work.

**See also**

`wxMenu` (p. 1074), *Event handling overview* (p. 2077)

### **wxMenuBar::wxMenuBar**

**wxMenuBar**(long *style* = 0)

Default constructor.

**wxMenuBar**(size\_t *n*, wxMenu\* *menus*[], const wxString *titles*[], long *style* = 0)

Construct a menu bar from arrays of menus and titles.

**Parameters**

*n*



The number of menus.

*menus*

An array of menus. Do not use this array again - it now belongs to the menu bar.

*titles*

An array of title strings. Deallocate this array after creating the menu bar.

*style*

If `wxMB_DOCKABLE` the menu bar can be detached (wxGTK only).

**wxPython note:** Only the default constructor is supported in wxPython. Use `wxMenuBar::Append` (p. 1089) instead.

**wxPerl note:** wxPerl only supports the first constructor: use `wxMenuBar::Append` (p. 1089) instead.

### **wxMenuBar::~~wxMenuBar**

**void ~wxMenuBar()**

Destructor, destroying the menu bar and removing it from the parent frame (if any).

### **wxMenuBar::Append**

**bool Append(wxMenu \*menu, const wxString& title)**

Adds the item to the end of the menu bar.

#### **Parameters**

*menu*

The menu to add. Do not deallocate this menu after calling **Append**.

*title*

The title of the menu.

#### **Return value**

true on success, false if an error occurred.

#### **See also**

`wxMenuBar::Insert` (p. 1094)

### **wxMenuBar::Check**

**void Check(int id, const bool check)**

Checks or unchecks a menu item.

**Parameters**

*id*

The menu item identifier.

*check*

If true, checks the menu item, otherwise the item is unchecked.

**Remarks**

Only use this when the menu bar has been associated with a frame; otherwise, use the `wxMenu` equivalent call.

**`wxMenuBar::Enable`**

**`void Enable(int id, const bool enable)`**

Enables or disables (greys out) a menu item.

**Parameters**

*id*

The menu item identifier.

*enable*

true to enable the item, false to disable it.

**Remarks**

Only use this when the menu bar has been associated with a frame; otherwise, use the `wxMenu` equivalent call.

**`wxMenuBar::EnableTop`**

**`void EnableTop(int pos, const bool enable)`**

Enables or disables a whole menu.

**Parameters**

*pos*

The position of the menu, starting from zero.

*enable*

true to enable the menu, false to disable it.

**Remarks**

Only use this when the menu bar has been associated with a frame.

### **wxMenuBar::FindMenu**

**int FindMenu(const wxString& *title*) const**

Returns the index of the menu with the given *title* or `wxNOT_FOUND` if no such menu exists in this menubar. The *title* parameter may specify either the menu title (with accelerator characters, i.e. "&File") or just the menu label ("File") indifferently.

### **wxMenuBar::FindMenuItem**

**int FindMenuItem(const wxString& *menuString*, const wxString& *itemString*) const**

Finds the menu item id for a menu name/menu item string pair.

#### **Parameters**

*menuString*

Menu title to find.

*itemString*

Item to find.

#### **Return value**

The menu item identifier, or `wxNOT_FOUND` if none was found.

#### **Remarks**

Any special menu codes are stripped out of source and target strings before matching.

### **wxMenuBar::FindItem**

**wxMenuItem \* FindItem(int *id*, wxMenu \*\**menu* = NULL) const**

Finds the menu item object associated with the given menu item identifier.

#### **Parameters**

*id*

Menu item identifier.

*menu*

If not NULL, menu will get set to the associated menu.

#### **Return value**

The found menu item object, or NULL if one was not found.

**wxMenuBar::GetHelpString****wxString GetHelpString(int *id*) const**

Gets the help string associated with the menu item identifier.

**Parameters***id*

The menu item identifier.

**Return value**

The help string, or the empty string if there was no help string or the menu item was not found.

**See also**

*wxMenuBar::SetHelpString* (p. 1096)

**wxMenuBar::GetLabel****wxString GetLabel(int *id*) const**

Gets the label associated with a menu item.

**Parameters***id*

The menu item identifier.

**Return value**

The menu item label, or the empty string if the item was not found.

**Remarks**

Use only after the menubar has been associated with a frame.

**wxMenuBar::GetLabelTop****wxString GetLabelTop(int *pos*) const**

Returns the label of a top-level menu. Note that the returned string does not include the accelerator characters which could have been specified in the menu title string during its construction.

**Parameters***pos*

Position of the menu on the menu bar, starting from zero.

**Return value**

The menu label, or the empty string if the menu was not found.

**Remarks**

Use only after the menubar has been associated with a frame.

This function is deprecated in favour of *GetMenuLabel* (p. 1093) and *GetMenuLabelText* (p. 1094).

**See also**

*wxMenuBar::SetLabelTop* (p. 1097)

**wxMenuBar::GetMenu**

**wxMenu\* GetMenu(int menuIndex) const**

Returns the menu at *menuIndex* (zero-based).

**wxMenuBar::GetMenuCount**

**int GetMenuCount() const**

Returns the number of menus in this menubar.

**wxMenuBar::GetMenuLabel**

**wxString GetMenuLabel(int pos) const**

Returns the label of a top-level menu. Note that the returned string includes the accelerator characters that have been specified in the menu title string during its construction.

**Parameters**

*pos*

Position of the menu on the menu bar, starting from zero.

**Return value**

The menu label, or the empty string if the menu was not found.

**Remarks**

Use only after the menubar has been associated with a frame.

**See also**

*wxMenuBar::GetMenuLabelText* (p. 1094), *wxMenuBar::SetMenuLabel* (p. 1097)

**wxMenuBar::GetMenuLabelText****wxString GetMenuLabelText(int *pos*) const**

Returns the label of a top-level menu. Note that the returned string does not include any accelerator characters that may have been specified in the menu title string during its construction.

**Parameters***pos*

Position of the menu on the menu bar, starting from zero.

**Return value**

The menu label, or the empty string if the menu was not found.

**Remarks**

Use only after the menubar has been associated with a frame.

**See also**

*wxMenuBar::GetMenuLabel* (p. 1093), *wxMenuBar::SetMenuLabel* (p. 1097)

**wxMenuBar::Insert****bool Insert(size\_t *pos*, wxMenu \**menu*, const wxString& *title*)**

Inserts the menu at the given position into the menu bar. Inserting menu at position 0 will insert it in the very beginning of it, inserting at position *GetMenuCount()* (p. 1093) is the same as calling *Append()* (p. 1089).

**Parameters***pos*

The position of the new menu in the menu bar

*menu*

The menu to add. wxMenuBar owns the menu and will free it.

*title*

The title of the menu.

**Return value**

true on success, false if an error occurred.

**See also**

*wxMenuBar::Append* (p. 1089)

**wxMenuBar::IsChecked****bool IsChecked(int *id*) const**

Determines whether an item is checked.

**Parameters***id*

The menu item identifier.

**Return value**

true if the item was found and is checked, false otherwise.

**wxMenuBar::IsEnabled****bool IsEnabled(int *id*) const**

Determines whether an item is enabled.

**Parameters***id*

The menu item identifier.

**Return value**

true if the item was found and is enabled, false otherwise.

**wxMenuBar::Refresh****void Refresh()**

Redraw the menu bar

**wxMenuBar::Remove****wxMenu \* Remove(size\_t *pos*)**

Removes the menu from the menu bar and returns the menu object - the caller is responsible for deleting it. This function may be used together with *wxMenuBar::Insert* (p. 1094) to change the menubar dynamically.

**See also**

*wxMenuBar::Replace* (p. 1095)

**wxMenuBar::Replace****wxMenu \* Replace(size\_t *pos*, wxMenu \**menu*, const wxString& *title*)**

Replaces the menu at the given position with another one.

**Parameters**

*pos*

The position of the new menu in the menu bar

*menu*

The menu to add.

*title*

The title of the menu.

**Return value**

The menu which was previously at position *pos*. The caller is responsible for deleting it.

**See also**

*wxMenuBar::Insert* (p. 1094), *wxMenuBar::Remove* (p. 1095)

**wxMenuBar::SetHelpString**

**void SetHelpString(int *id*, const wxString& *helpString*)**

Sets the help string associated with a menu item.

**Parameters**

*id*

Menu item identifier.

*helpString*

Help string to associate with the menu item.

**See also**

*wxMenuBar::GetHelpString* (p. 1092)

**wxMenuBar::SetLabel**

**void SetLabel(int *id*, const wxString& *label*)**

Sets the label of a menu item.

**Parameters**

*id*

Menu item identifier.



*label*

Menu item label.

### Remarks

Use only after the menubar has been associated with a frame.

### See also

*wxMenuBar::GetLabel* (p. 1092)

### **wxMenuBar::SetLabelTop**

**void SetLabelTop(int *pos*, const wxString& *label*)**

Sets the label of a top-level menu.

### Parameters

*pos*

The position of a menu on the menu bar, starting from zero.

*label*

The menu label.

### Remarks

Use only after the menubar has been associated with a frame.

This function has been deprecated in favour of *SetMenuLabel* (p. 1097).

### See also

*wxMenuBar::GetLabelTop* (p. 1092)

### **wxMenuBar::SetMenuLabel**

**void SetMenuLabel(int *pos*, const wxString& *label*)**

Sets the label of a top-level menu.

### Parameters

*pos*

The position of a menu on the menu bar, starting from zero.

*label*

The menu label.

### Remarks

Use only after the menubar has been associated with a frame.

**See also**

*wxMenuBar::GetMenuLabel* (p. 1093), *wxMenuBar::GetMenuLabelText* (p. 1094)

## **wxMenuEvent**

This class is used for a variety of menu-related events. Note that these do not include menu command events, which are handled using *wxCommandEvent* (p. 250) objects.

The default handler for `wxEVT_MENU_HIGHLIGHT` displays help text in the first field of the status bar.

**Derived from**

*wxEvent* (p. 572)

*wxObject* (p. 1148)

**Include files**

<wx/event.h>

**Event table macros**

To process a menu event, use these event handler macros to direct input to member functions that take a *wxMenuEvent* argument. They can only be processed by a menubar's frame.

<b>EVT_MENU_OPEN(func)</b>	A menu is about to be opened. On Windows, this is only sent once for each navigation of the menubar (up until all menus have closed).
<b>EVT_MENU_CLOSE(func)</b>	A menu has been just closed.
<b>EVT_MENU_HIGHLIGHT(id, func)</b>	The menu item with the specified id has been highlighted: used to show help prompts in the status bar by <i>wxFrame</i> (p. 682)
<b>EVT_MENU_HIGHLIGHT_ALL(func)</b>	A menu item has been highlighted, i.e. the currently selected menu item has changed.

**See also**

*Command events* (p. 250),

*Event handling overview* (p. 2077)

**wxMenuEvent::wxMenuEvent**

**wxMenuEvent**(WXTYPE *id* = 0, int *id* = 0, **wxMenu\*** *menu* = NULL)

Constructor.

### **wxMenuEvent::GetMenu**

**wxMenu \* GetMenu() const**

Returns the menu which is being opened or closed. This method should only be used with the `OPEN` and `CLOSE` events and even for them the returned pointer may be `NULL` in some ports.

### **wxMenuEvent::GetMenuId**

**int GetMenuId() const**

Returns the menu identifier associated with the event. This method should be only used with the `HIGHLIGHT` events.

### **wxMenuEvent::IsPopup**

**bool IsPopup() const**

Returns `true` if the menu which is being opened or closed is a popup menu, `false` if it is a normal one.

This method should only be used with the `OPEN` and `CLOSE` events.

## **wxMenuItem**

A menu item represents an item in a menu. Note that you usually don't have to deal with it directly as *wxMenu* (p. 1074) methods usually construct an object of this class for you.

Also please note that the methods related to fonts and bitmaps are currently only implemented for Windows and GTK+.

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/menuitem.h>

### **See also**

*wxMenuBar* (p. 1088), *wxMenu* (p. 1074)

### **wxMenuItem::wxMenuItem**

**wxMenuItem(wxMenu\* parentMenu = NULL, int id = wxID\_SEPARATOR, const**

```
wxString& text = "", const wxString& helpString = "", wxItemKind kind =  
wxITEM_NORMAL, wxMenu* subMenu = NULL)
```

Constructs a `wxMenuItem` object.

Menu items can be standard, or "stock menu items", or custom. For the standard menu items (such as commands to open a file, exit the program and so on, see *stock items* (p. 2004) for the full list) it is enough to specify just the stock ID and leave *text* and *helpString* empty. In fact, leaving at least *text* empty for the stock menu items is strongly recommended as they will have appearance and keyboard interface (including standard accelerators) familiar to the user.

For the custom (non-stock) menu items, *text* must be specified and while *helpString* may be left empty, it's recommended to pass the item description (which is automatically shown by the library in the status bar when the menu item is selected) in this parameter.

Finally note that you can e.g. use a stock menu label without using its stock help string:

```
// use all stock properties:  
helpMenu->Append(wxID_ABOUT);  
  
// use the stock label and the stock accelerator but not the stock help  
string:  
helpMenu->Append(wxID_ABOUT, wxEmptyString, wxT("My custom help  
string"));  
  
// use all stock properties except for the bitmap:  
wxMenuItem *mymenu = new wxMenuItem(helpMenu, wxID_ABOUT);  
mymenu->SetBitmap(wxArtProvider::GetBitmap(wxART_WARNING));  
helpMenu->Append(mymenu);
```

that is, stock properties are set independently one from the other.

### Parameters

#### *parentMenu*

Menu that the menu item belongs to.

#### *id*

Identifier for this menu item, or `wxID_SEPARATOR` to indicate a separator.

#### *text*

Text for the menu item, as shown on the menu. An accelerator key can be specified using the ampersand '&' character. In order to embed an ampersand character in the menu item text, the ampersand must be doubled.

#### *helpString*

Optional help string that will be shown on the status bar.

#### *kind*

May be `wxITEM_SEPARATOR`, `wxITEM_NORMAL`, `wxITEM_CHECK` or `wxITEM_RADIO`

*subMenu*

If non-NULL, indicates that the menu item is a submenu.

### **wxMenuItem::~wxMenuItem**

**~wxMenuItem()**

Destructor.

### **wxMenuItem::Check**

**void Check(bool check = true)**

Checks or unchecks the menu item.

Note that this only works when the item is already appended to a menu.

### **wxMenuItem::Enable**

**void Enable(bool enable = true)**

Enables or disables the menu item.

### **wxMenuItem::GetBackgroundColour**

**wxColour& GetBackgroundColour() const**

Returns the background colour associated with the menu item (Windows only).

### **wxMenuItem::GetBitmap**

**wxBitmap& GetBitmap(bool checked = true) const**

Returns the checked or unchecked bitmap (Windows only).

### **wxMenuItem::GetFont**

**wxFont& GetFont() const**

Returns the font associated with the menu item (Windows only).

### **wxMenuItem::GetHelp**

**wxString GetHelp() const**

Returns the help string associated with the menu item.

**wxMenuItem::GetId****int GetId() const**

Returns the menu item identifier.

**wxMenuItem::GetItemLabel****wxString GetItemLabel() const**

Returns the text associated with the menu item including any accelerator characters that were passed to the constructor or SetItemLabel.

**See also**

*GetItemLabelText* (p. 1102), *GetLabelText* (p. 1102)

**wxMenuItem::GetItemLabelText****wxString GetItemLabelText() const**

Returns the text associated with the menu item, without any accelerator characters.

**See also**

*GetItemLabel* (p. 1102), *GetLabelText* (p. 1102)

**wxMenuItem::GetKind****wxItemKind GetKind() const**

Returns the item kind, one of `wxITEM_SEPARATOR`, `wxITEM_NORMAL`, `wxITEM_CHECK` or `wxITEM_RADIO`.

**wxMenuItem::GetLabel****wxString GetLabel() const**

Returns the text associated with the menu item without any accelerator characters it might contain.

This function is deprecated in favour of *GetItemLabelText* (p. 1102).

**See also**

*GetText* (p. 1103), *GetLabelFromText* (p. 1103)

**wxMenuItem::GetLabelText****static wxString GetLabelText(const wxString& text)**

Strips all accelerator characters and mnemonics from the given *text*. For example,

```
wxMenuItem::GetLabelFromText("&Hello\tCtrl-H");
```

will return just "Hello".

**See also**

*GetItemLabelText* (p. 1102), *GetItemLabel* (p. 1102)

**wxMenuItem::GetLabelFromText**

**static wxString GetLabelFromText(const wxString& text)**

Strips all accelerator characters and mnemonics from the given *text*. For example,

```
wxMenuItem::GetLabelFromText("&Hello\tCtrl-H");
```

will return just "Hello".

This function is deprecated; please use *wxMenuItem::GetLabelText* (p. 1102) instead.

**See also**

*GetText* (p. 1103), *GetLabel* (p. 1102)

**wxMenuItem::GetMarginWidth**

**int GetMarginWidth() const**

Gets the width of the menu item checkmark bitmap (Windows only).

**wxMenuItem::GetMenu**

**wxMenu\* GetMenu() const**

Returns the menu this menu item is in, or NULL if this menu item is not attached.

**wxMenuItem::GetName**

**wxString GetName() const**

Returns the text associated with the menu item.

**NB:** this function is deprecated, please use *GetItemLabel* (p. 1102) or *GetItemLabelText* (p. 1102) instead.

**wxMenuItem::GetText**

**wxString GetText() const**

Returns the text associated with the menu item, such as it was passed to the *wxMenuItem* constructor, i.e. with any accelerator characters it may contain.

This function is deprecated in favour of *GetItemLabel* (p. 1102).

**See also**

*GetLabel* (p. 1102), *GetLabelFromText* (p. 1103)

**wxMenuItem::GetSubMenu**

**wxMenu\* GetSubMenu() const**

Returns the submenu associated with the menu item, or NULL if there isn't one.

**wxMenuItem::GetTextColour**

**wxColour& GetTextColour() const**

Returns the text colour associated with the menu item (Windows only).

**wxMenuItem::IsCheckable**

**bool IsCheckable() const**

Returns true if the item is checkable.

**wxMenuItem::IsChecked**

**bool IsChecked() const**

Returns true if the item is checked.

**wxMenuItem::IsEnabled**

**bool IsEnabled() const**

Returns true if the item is enabled.

**wxMenuItem::IsSeparator**

**bool IsSeparator() const**

Returns true if the item is a separator.

**wxMenuItem::IsSubMenu**

**bool IsSubMenu() const**

Returns true if the item is a submenu.

**wxMenuItem::SetBackgroundColour**



**void SetBackgroundColour(const wxColour& colour) const**

Sets the background colour associated with the menu item (Windows only).

**wxMenuItem::SetBitmap**

**void SetBitmap(const wxBitmap& bmp)**

Sets the bitmap for the menu item (Windows and GTK+ only). It is equivalent to *SetBitmaps* (p. 1105)(bmp, wxNullBitmap).

**wxMenuItem::SetBitmaps**

**void SetBitmaps(const wxBitmap& checked, const wxBitmap& unchecked = wxNullBitmap)**

Sets the checked/unchecked bitmaps for the menu item (Windows only). The first bitmap is also used as the single bitmap for uncheckable menu items.

**wxMenuItem::SetFont**

**void SetFont(const wxFont& font)**

Sets the font associated with the menu item (Windows only).

**wxMenuItem::SetHelp**

**void SetHelp(const wxString& helpString)**

Sets the help string.

**wxMenuItem::SetItemLabel**

**void SetItemLabel(const wxString& label)**

Sets the label associated with the menu item.

**wxMenuItem::SetMarginWidth**

**void SetMarginWidth(int width) const**

Sets the width of the menu item checkmark bitmap (Windows only).

**wxMenuItem::SetMenu**

**void SetMenu(const wxMenu\* menu)**

Sets the parent menu which will contain this menu item.

**wxMenuItem::SetSubMenu**

**void SetSubMenu(const wxMenu\* menu)**

Sets the submenu of this menu item.

**wxMenuItem::SetText**

**void SetText(const wxString& text)**

Sets the text associated with the menu item.

This function is deprecated in favour of *SetItemLabel* (p. 1105).

**wxMenuItem::SetTextColour**

**void SetTextColour(const wxColour& colour)**

Sets the text colour associated with the menu item (Windows only).

## wxMessageDialog

This class represents a dialog that shows a single or multi-line message, with a choice of OK, Yes, No and Cancel buttons.

### Derived from

*wxDialog* (p. 496)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### Include files

<wx/msgdlg.h>

### See also

*wxMessageDialog* overview (p. 2131)

## wxMessageDialog::wxMessageDialog

**wxMessageDialog(wxWindow\* parent, const wxString& message, const wxString& caption = "Message box", long style = wxOK | wxCANCEL, const wxPoint& pos = wxDefaultPosition)**

Constructor. Use *wxMessageDialog::ShowModal* (p. 1107) to show the dialog.

### Parameters

*parent*

Parent window.

*message*

Message to show on the dialog.

*caption*

The dialog caption.

*style*

A dialog style (bitlist) containing flags chosen from the following:

<b>wxOK</b>	Show an OK button.
<b>wxCANCEL</b>	Show a Cancel button.
<b>wxYES_NO</b>	Show Yes and No buttons.
<b>wxYES_DEFAULT</b>	Used with <b>wxYES_NO</b> , makes <b>Yes</b> button the default - which is the default behaviour.
<b>wxNO_DEFAULT</b>	Used with <b>wxYES_NO</b> , makes <b>No</b> button the default.
<b>wxICON_EXCLAMATION</b>	Shows an exclamation mark icon.
<b>wxICON_HAND</b>	Shows an error icon.
<b>wxICON_ERROR</b>	Shows an error icon - the same as wxICON_HAND.
<b>wxICON_QUESTION</b>	Shows a question mark icon.
<b>wxICON_INFORMATION</b>	Shows an information (i) icon.
<b>wxSTAY_ON_TOP</b>	The message box stays on top of all other window, even those of the other applications (Windows only).

*pos*

Dialog position. Not Windows.

### **wxMessageDialog::~wxMessageDialog**

**~wxMessageDialog()**

Destructor.

### **wxMessageDialog::ShowModal**

**int ShowModal()**

Shows the dialog, returning one of wxID\_OK, wxID\_CANCEL, wxID\_YES, wxID\_NO.

## wxMetafile

A **wxMetafile** represents the MS Windows metafile object, so metafile operations have no effect in X. In wxWidgets, only sufficient functionality has been provided for copying a graphic to the clipboard; this may be extended in a future version. Presently, the only way of creating a metafile is to use a wxMetafileDC.

### Derived from

*wxObject* (p. 1148)

### Include files

<wx/metafile.h>

### See also

*wxMetafileDC* (p. 1109)

## wxMetafile::wxMetafile

**wxMetafile**(const wxString& filename = "")

Constructor. If a filename is given, the Windows disk metafile is read in. Check whether this was performed successfully by using the *wxMetafile::IsOk* (p. 1108) member.

## wxMetafile::~~wxMetafile

**~wxMetafile**()

Destructor. See *reference-counted object destruction* (p. 2046) for more info.

## wxMetafile::IsOk

**bool** IsOk()

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 a example of metafile, metafile device context and clipboard use from the `hello.cpp` example. Note the way the metafile dimensions are passed to the clipboard, making use of the device context's ability to keep track of the maximum extent of drawing commands.

```
wxMetafileDC dc;
if (dc.Ok())
{
    Draw(dc, false);
    wxMetafile *mf = dc.Close();
    if (mf)
    {
        bool success = mf->SetClipboard((int)(dc.MaxX() + 10),
(int)(dc.MaxY() + 10));
        delete mf;
    }
}
```

## wxMetafileDC

This is a type of device context that allows a metafile object to be created (Windows only), and has most of the characteristics of a normal **wxDC**. The `wxMetafileDC::Close` (p. 1110) member must be called after drawing into the device context, to return a metafile. The only purpose for this at present is to allow the metafile to be copied to the clipboard (see `wxMetafile` (p. 1108)).

Adding metafile capability to an application should be easy if you already write to a `wxDC`; simply pass the `wxMetafileDC` to your drawing function instead. You may wish to conditionally compile this code so it is not compiled under X (although no harm will result if you leave it in).

Note that a metafile saved to disk is in standard Windows metafile format, and cannot be imported into most applications. To make it importable, call the function `::wxMakeMetafilePlaceable` (p. 1946) after closing your disk-based metafile device context.

### Derived from

`wxDC` (p. 456)

`wxObject` (p. 1148)

### Include files

`<wx/metafile.h>`

### See also

`wxMetafile` (p. 1108), `wxDC` (p. 456)

**wxMetafileDC::wxMetafileDC****wxMetafileDC**(const wxString& filename = "")

Constructor. If no filename is passed, the metafile is created in memory.

**wxMetafileDC::~~wxMetafileDC****~wxMetafileDC**()

Destructor.

**wxMetafileDC::Close****wxMetafile \* Close**()

This must be called after the device context is finished with. A metafile is returned, and ownership of it passes to the calling application (so it should be destroyed explicitly).

**wxMimeTypeManager**

This class allows the application to retrieve the information about all known MIME types from a system-specific location and the filename extensions to the MIME types and vice versa. After initialization the functions *wxMimeTypeManager::GetFileTypeFromMimeType* (p. 1112) and *wxMimeTypeManager::GetFileTypeFromExtension* (p. 1112) may be called: they will return a *wxFileType* (p. 639) object which may be further queried for file description, icon and other attributes.

**Windows:** MIME type information is stored in the registry and no additional initialization is needed.

**Unix:** MIME type information is stored in the files *mailcap* and *mime.types* (system-wide) and *.mailcap* and *.mime.types* in the current user's home directory: all of these files are searched for and loaded if found by default. However, additional functions *wxMimeTypeManager::ReadMailcap* (p. 1113) and *wxMimeTypeManager::ReadMimeType* (p. 1113) are provided to load additional files.

If GNOME or KDE desktop environment is installed, then *wxMimeTypeManager* gathers MIME information from respective files (e.g. *.kdeInk* files under KDE).

NB: Currently, *wxMimeTypeManager* is limited to reading MIME type information but it will support modifying it as well in future versions.

**Global objects**

Global instance of *wxMimeTypeManager* is always available. It is defined as follows:

```
wxMimeTypeManager *wxTheMimeTypeManager;
```

It is recommended to use this instance instead of creating your own because gathering

MIME information may take quite a long time on Unix systems.

### **Derived from**

No base class.

### **Include files**

<wx/mimetype.h>

### **See also**

*wxFileType* (p. 639)

### **Helper functions**

All of these functions are static (i.e. don't need a *wxMimeTypesManager* object to call them) and provide some useful operations for string representations of MIME types. Their usage is recommended instead of directly working with MIME types using *wxString* functions.

*IsOfType* (p. 1112)

### **Constructor and destructor**

NB: You won't normally need to use more than one *wxMimeTypesManager* object in a program.

*wxMimeTypesManager* (p. 1112)

*~wxMimeTypesManager* (p. 1112)

### **Query database**

These functions are the heart of this class: they allow to find a *file type* (p. 639) object from either file extension or MIME type. If the function is successful, it returns a pointer to the *wxFileType* object which **must** be deleted by the caller, otherwise `NULL` will be returned.

*GetFileTypeFromMimeType* (p. 1112)

*GetFileTypeFromExtension* (p. 1112)

### **Initialization functions**

**Unix:** These functions may be used to load additional files (except for the default ones which are loaded automatically) containing MIME information in either `mailcap(5)` or `mime.types(5)` format.

*ReadMailcap* (p. 1113)

*ReadMimeTypes* (p. 1113)

*AddFallbacks* (p. 1112)

**wxMimeTypeManager::wxMimeTypeManager****wxMimeTypeManager()**

Constructor puts the object in the "working" state, no additional initialization are needed - but *ReadXXX* (p. 1111) may be used to load additional mailcap/mime.types files.

**wxMimeTypeManager::~~wxMimeTypeManager****~wxMimeTypeManager()**

Destructor is not virtual, so this class should not be derived from.

**wxMimeTypeManager::AddFallbacks****void AddFallbacks(const wxFileTypeInfo \*fallbacks)**

This function may be used to provide hard-wired fallbacks for the MIME types and extensions that might not be present in the system MIME database.

Please see the *typetest* sample for an example of using it.

**wxMimeTypeManager::GetFileTypeFromExtension****wxFileType\* GetFileTypeFromExtension(const wxString& extension)**

Gather information about the files with given extension and return the corresponding *wxFileType* (p. 639) object or `NULL` if the extension is unknown.

The *extension* parameter may have, or not, the leading dot, if it has it, it is stripped automatically. It must not however be empty.

**wxMimeTypeManager::GetFileTypeFromMimeType****wxFileType\* GetFileTypeFromMimeType(const wxString& mimeType)**

Gather information about the files with given MIME type and return the corresponding *wxFileType* (p. 639) object or `NULL` if the MIME type is unknown.

**wxMimeTypeManager::IsOfType****bool IsOfType(const wxString& mimeType, const wxString& wildcard)**

This function returns true if either the given *mimeType* is exactly the same as *wildcard* or if it has the same category and the subtype of *wildcard* is `'*'`. Note that the `'*'` wildcard is not allowed in *mimeType* itself.

The comparison don by this function is case insensitive so it is not necessary to convert the strings to the same case before calling it.



### **wxMimeTypeManager::ReadMailcap**

**bool ReadMailcap(const wxString& filename, bool fallback = false)**

Load additional file containing information about MIME types and associated information in mailcap format. See `metamail(1)` and `mailcap(5)` for more information.

*fallback* parameter may be used to load additional mailcap files without overriding the settings found in the standard files: normally, entries from files loaded with `ReadMailcap` will override the entries from files loaded previously (and the standard ones are loaded in the very beginning), but this will not happen if this parameter is set to true (default is false).

The return value is true if there were no errors in the file or false otherwise.

### **wxMimeTypeManager::ReadMimeTypes**

**bool ReadMimeTypes(const wxString& filename)**

Load additional file containing information about MIME types and associated information in `mime.types` file format. See `metamail(1)` and `mailcap(5)` for more information.

The return value is true if there were no errors in the file or false otherwise.

## **wxMiniFrame**

A miniframe is a frame with a small title bar. It is suitable for floating toolbars that must not take up too much screen area.

### **Derived from**

*wxFrame* (p. 682)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### **Include files**

<wx/minifram.h>

### **Window styles**

<b>wxICONIZE</b>	Display the frame iconized (minimized) (Windows only).
<b>wxCAPTION</b>	Puts a caption on the frame.
<b>wxMINIMIZE</b>	Identical to <b>wxICONIZE</b> .
<b>wxMINIMIZE_BOX</b>	Displays a minimize box on the frame (Windows and Motif only).
<b>wxMAXIMIZE</b>	Displays the frame maximized (Windows only).
<b>wxMAXIMIZE_BOX</b>	Displays a maximize box on the frame (Windows and Motif

	only).
<b>wxCLOSE_BOX</b>	Displays a close box on the frame.
<b>wxSTAY_ON_TOP</b>	Stay on top of other windows (Windows only).
<b>wxSYSTEM_MENU</b>	Displays a system menu (Windows and Motif only).
<b>wxTINY_CAPTION_HORIZ</b>	This style is obsolete and not used any longer.
<b>wxTINY_CAPTION_VERT</b>	This style is obsolete and not used any longer.
<b>wxRESIZE_BORDER</b>	Displays a resizable border around the window.

### Remarks

This class has miniframe functionality under Windows and GTK, i.e. the presence of mini frame will not be noted in the task bar and focus behaviour is different. On other platforms, it behaves like a normal frame.

### See also

*wxMDIParentFrame* (p. 1051), *wxMDIChildFrame* (p. 1047), *wxFrame* (p. 682), *wxDialog* (p. 496)

## **wxMiniFrame::wxMiniFrame**

### **wxMiniFrame()**

Default constructor.

**wxMiniFrame**(*wxWindow\** *parent*, *wxWindowID* *id*, **const** *wxString&* *title*, **const** *wxPoint&* *pos* = *wxDefaultPosition*, **const** *wxSize&* *size* = *wxDefaultSize*, **long** *style* = *wxCAPTION* | *wxRESIZE\_BORDER*, **const** *wxString&* *name* = "frame")

Constructor, creating the window.

### Parameters

*parent*

The window parent. This may be NULL. If it is non-NULL, the frame will always be displayed on top of the parent window on Windows.

*id*

The window identifier. It may take a value of -1 to indicate a default value.

*title*

The caption to be displayed on the frame's title bar.

*pos*

The window position. A value of (-1, -1) indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform.

*size*

The window size. A value of (-1, -1) indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform.

*style*

The window style. See *wxMiniFrame* (p. 1113).

*name*

The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

### Remarks

The frame behaves like a normal frame on non-Windows platforms.

### See also

*wxMiniFrame::Create* (p. 1115)

### **wxMiniFrame::~~wxMiniFrame**

**void ~wxMiniFrame()**

Destructor. Destroys all child windows and menu bar if present.

### **wxMiniFrame::Create**

**bool Create**(wxWindow\* *parent*, wxWindowID *id*, const wxString& *title*, const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size* = wxDefaultSize, long *style* = wxCAPTION | wxRESIZE\_BORDER, const wxString& *name* = "frame")

Used in two-step frame construction. See *wxMiniFrame::wxMiniFrame* (p. 1114) for further details.

## **wxMirrorDC**

wxMirrorDC is a simple wrapper class which is always associated with a real wxDC (p. 456) object and either forwards all of its operations to it without changes (no mirroring takes place) or exchanges x and y coordinates which makes it possible to reuse the same code to draw a figure and its mirror -- i.e. reflection related to the diagonal line  $x == y$ .

wxMirrorDC has been added in wxWidgets version 2.5.0.

### Derived from

wxDC (p. 456)

**Include files**

<wx/dcmirror.h>

**wxMirrorDC::wxMirrorDC**

**wxMirrorDC(wxDC& dc, bool mirror)**

Creates a (maybe) mirrored DC associated with the real *dc*. Everything drawn on *wxMirrorDC* will appear (and maybe mirrored) on *dc*.

*mirror* specifies if we do mirror (if it is `true`) or not (if it is `false`).

**wxModule**

The module system is a very simple mechanism to allow applications (and parts of *wxWidgets* itself) to define initialization and cleanup functions that are automatically called on *wxWidgets* startup and exit.

To define a new kind of module, derive a class from *wxModule*, override the *OnInit* (p. 1118) and *OnExit* (p. 1117) functions, and add the `DECLARE_DYNAMIC_CLASS` and `IMPLEMENT_DYNAMIC_CLASS` to header and implementation files (which can be the same file). On initialization, *wxWidgets* will find all classes derived from *wxModule*, create an instance of each, and call each *OnInit* function. On exit, *wxWidgets* will call the *OnExit* function for each module instance.

Note that your module class does not have to be in a header file.

For example:

```
// A module to allow DDE initialization/cleanup
// without calling these functions from app.cpp or from
// the user's application.
class wxDDEModule: public wxModule
{
public:
    wxDDEModule() { }
    virtual bool OnInit() { wxDDEInitialize(); return true; };
    virtual void OnExit() { wxDDECleanUp(); };

private:
    DECLARE_DYNAMIC_CLASS(wxDDEModule)
};

IMPLEMENT_DYNAMIC_CLASS(wxDDEModule, wxModule)

// Another module which uses DDE in its OnInit()
class MyModule: public wxModule
{
public:
```

```
wxDDEModule() { AddDependency(CLASSINFO(wxDDEModule)); }  
virtual bool OnInit() { ... code using DDE ... }  
virtual void OnExit() { ... }  
  
private:  
    DECLARE_DYNAMIC_CLASS(wxDDEModule)  
};
```

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/module.h>

**wxModule::wxModule****wxModule()**

Constructs a wxModule object.

**wxModule::~~wxModule****~wxModule()**

Destructor.

**wxModule::AddDependency****void AddDependency(wxClassInfo \* dep)**

Call this function from the constructor of the derived class. *dep* must be the *CLASSINFO* (p. 1964) of a wxModule-derived class and the corresponding module will be loaded *before* and unloaded *after* this module.

Note that circular dependencies are detected and result in a fatal error.

**Parameters**

*dep*

The class information object for the dependent module.

**wxModule::OnExit****virtual void OnExit()**

Provide this function with appropriate cleanup for your module.

**wxModule::OnInit****virtual bool OnInit()**

Provide this function with appropriate initialization for your module. If the function returns false, wxWidgets will exit immediately.

**wxMouseCaptureChangedEvent**

An mouse capture changed event is sent to a window that loses its mouse capture. This is called even if `wxWindow::ReleaseCapture` was called by the application code. Handling this event allows an application to cater for unexpected capture releases which might otherwise confuse mouse handling code.

This event is implemented under Windows only.

**Derived from**

*wxEvent* (p. 572)

*wxObject* (p. 1148)

**Include files**

<wx/event.h>

**Event table macros**

To process this event, use the following event handler macro to direct input to a member function that takes a `wxMouseCaptureChangedEvent` argument.

**EVT\_MOUSE\_CAPTURE\_CHANGED(func)**      Process a  
   wxEVT\_MOUSE\_CAPTURE\_CHANGED  
   event.

**See also**

*wxMouseCaptureLostEvent* (p. 1119) *Event handling overview* (p. 2077),  
*wxWindow::CaptureMouse* (p. 1799), *wxWindow::ReleaseMouse* (p. 1831),  
*wxWindow::GetCapture* (p. 1810)

**wxMouseCaptureChangedEvent::wxMouseCaptureChangedEvent**

**wxMouseCaptureChangedEvent(wxWindowID windowId = 0, wxWindow\* gainedCapture = NULL)**

Constructor.

**wxActivateEvent::GetCapturedWindow**

**wxWindow\* GetCapturedWindow() const**

Returns the window that gained the capture, or NULL if it was a non-wxWidgets window.

## wxMouseCaptureLostEvent

An mouse capture lost event is sent to a window that obtained mouse capture, which was subsequently loss due to "external" event, for example when a dialog box is shown or if another application captures the mouse.

If this happens, this event is sent to all windows that are on capture stack (i.e. called CaptureMouse, but didn't call ReleaseMouse yet). The event is *not* sent if the capture changes because of a call to CaptureMouse or ReleaseMouse.

This event is currently emitted under Windows only.

### Derived from

*wxEvent* (p. 572)  
*wxObject* (p. 1148)

### Include files

<wx/event.h>

### Event table macros

To process this event, use the following event handler macro to direct input to a member function that takes a wxMouseCaptureLostEvent argument.

**EVT\_MOUSE\_CAPTURE\_LOST(func)** Process a wxEVT\_MOUSE\_CAPTURE\_LOST event.

### See also

*wxMouseCaptureChangedEvent* (p. 1118) *Event handling overview* (p. 2077),  
*wxWindow::CaptureMouse* (p. 1799), *wxWindow::ReleaseMouse* (p. 1831),  
*wxWindow::GetCapture* (p. 1810)

## wxMouseCaptureLostEvent::wxMouseCaptureLostEvent

**wxMouseCaptureLostEvent(wxWindowID windowId = 0)**

Constructor.

## wxMouseEvent

This event class contains information about the events generated by the mouse: they include mouse buttons press and release events and mouse move events.

All mouse events involving the buttons use wxMOUSE\_BTN\_LEFT for the left mouse button,

`wxMOUSE_BTN_MIDDLE` for the middle one and `wxMOUSE_BTN_RIGHT` for the right one. Note that not all mice have a middle button so a portable application should avoid relying on the events from it.

**NB:** Note that under Windows CE mouse enter and leave events are not natively supported by the system but are generated by `wxWidgets` itself. This has several drawbacks: the `LEAVE_WINDOW` event might be received some time after the mouse left the window and the state variables for it may have changed during this time.

**NB:** Note the difference between methods like `LeftDown` (p. 1126) and `LeftIsDown` (p. 1126): the former returns `true` when the event corresponds to the left mouse button click while the latter returns `true` if the left mouse button is currently being pressed. For example, when the user is dragging the mouse you can use `LeftIsDown` (p. 1126) to test whether the left mouse button is (still) depressed. Also, by convention, if `LeftDown` (p. 1126) returns `true`, `LeftIsDown` (p. 1126) will also return `true` in `wxWidgets` whatever the underlying GUI behaviour is (which is platform-dependent). The same applies, of course, to other mouse buttons as well.

#### Derived from

`wxEvent` (p. 572)

#### Include files

<wx/event.h>

#### Event table macros

To process a mouse event, use these event handler macros to direct input to member functions that take a `wxMouseEvent` argument.

<b>EVT_LEFT_DOWN(func)</b>	Process a <code>wxEVT_LEFT_DOWN</code> event. The handler of this event should normally call <code>event.Skip()</code> (p. 575) to allow the default processing to take place as otherwise the window under mouse wouldn't get the focus.
<b>EVT_LEFT_UP(func)</b>	Process a <code>wxEVT_LEFT_UP</code> event.
<b>EVT_LEFT_DCLICK(func)</b>	Process a <code>wxEVT_LEFT_DCLICK</code> event.
<b>EVT_MIDDLE_DOWN(func)</b>	Process a <code>wxEVT_MIDDLE_DOWN</code> event.
<b>EVT_MIDDLE_UP(func)</b>	Process a <code>wxEVT_MIDDLE_UP</code> event.
<b>EVT_MIDDLE_DCLICK(func)</b>	Process a <code>wxEVT_MIDDLE_DCLICK</code> event.
<b>EVT_RIGHT_DOWN(func)</b>	Process a <code>wxEVT_RIGHT_DOWN</code> event.
<b>EVT_RIGHT_UP(func)</b>	Process a <code>wxEVT_RIGHT_UP</code> event.
<b>EVT_RIGHT_DCLICK(func)</b>	Process a <code>wxEVT_RIGHT_DCLICK</code> event.
<b>EVT_MOTION(func)</b>	Process a <code>wxEVT_MOTION</code> event.



<b>EVT_ENTER_WINDOW(func)</b>	Process a wxEVT_ENTER_WINDOW event.
<b>EVT_LEAVE_WINDOW(func)</b>	Process a wxEVT_LEAVE_WINDOW event.
<b>EVT_MOUSEWHEEL(func)</b>	Process a wxEVT_MOUSEWHEEL event.
<b>EVT_MOUSE_EVENTS(func)</b>	Process all mouse events.

**wxMouseEvent::m\_altDown****bool m\_altDown**

true if the Alt key is pressed down.

**wxMouseEvent::m\_controlDown****bool m\_controlDown**

true if control key is pressed down.

**wxMouseEvent::m\_leftDown****bool m\_leftDown**

true if the left mouse button is currently pressed down.

**wxMouseEvent::m\_middleDown****bool m\_middleDown**

true if the middle mouse button is currently pressed down.

**wxMouseEvent::m\_rightDown****bool m\_rightDown**

true if the right mouse button is currently pressed down.

**wxMouseEvent::m\_metaDown****bool m\_metaDown**

true if the Meta key is pressed down.

**wxMouseEvent::m\_shiftDown****bool m\_shiftDown**

true if shift is pressed down.

**wxMouseEvent::m\_x****long m\_x**

X-coordinate of the event.

**wxMouseEvent::m\_y****long m\_y**

Y-coordinate of the event.

**wxMouseEvent::m\_wheelRotation****int m\_wheelRotation**

The distance the mouse wheel is rotated.

**wxMouseEvent::m\_wheelDelta****int m\_wheelDelta**

The wheel delta, normally 120.

**wxMouseEvent::m\_linesPerAction****int m\_linesPerAction**

The configured number of lines (or whatever) to be scrolled per wheel action.

**wxMouseEvent::wxMouseEvent****wxMouseEvent(WXTYPE *mouseEventType* = 0)**

Constructor. Valid event types are:

- **wxEVT\_ENTER\_WINDOW**
- **wxEVT\_LEAVE\_WINDOW**
- **wxEVT\_LEFT\_DOWN**
- **wxEVT\_LEFT\_UP**
- **wxEVT\_LEFT\_DCLICK**
- **wxEVT\_MIDDLE\_DOWN**
- **wxEVT\_MIDDLE\_UP**
- **wxEVT\_MIDDLE\_DCLICK**

- **wxEVT\_RIGHT\_DOWN**
- **wxEVT\_RIGHT\_UP**
- **wxEVT\_RIGHT\_DCLICK**
- **wxEVT\_MOTION**
- **wxEVT\_MOUSEWHEEL**

**wxMouseEvent::AltDown****bool AltDown()**

Returns true if the Alt key was down at the time of the event.

**wxMouseEvent::Button****bool Button(int *button*)**

Returns true if the identified mouse button is changing state. Valid values of *button* are:

wxMOUSE_BTN_LEFT	check if left button was pressed
wxMOUSE_BTN_MIDDLE	check if middle button was pressed
wxMOUSE_BTN_RIGHT	check if right button was pressed
wxMOUSE_BTN_ANY	check if any button was pressed

**wxMouseEvent::ButtonDClick****bool ButtonDClick(int *but* = wxMOUSE\_BTN\_ANY)**

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 (see *Button* (p. 1123) for the possible values).

**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 (see *Button* (p. 1123) for the possible values).

**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 (see *Button* (p. 1123) for the possible values).

### **wxMouseEvent::CmdDown**

**bool CmdDown() const**

Same as *MetaDown* (p. 1127) under Mac, same as *ControlDown* (p. 1124) elsewhere.

#### **See also**

*wxKeyEvent::CmdDown* (p. 959)

### **wxMouseEvent::ControlDown**

**bool ControlDown()**

Returns true if the control key was down at the time of the event.

### **wxMouseEvent::Dragging**

**bool Dragging()**

Returns true if this was a dragging event (motion while a button is depressed).

#### **See also**

*Moving* (p. 1127)

### **wxMouseEvent::Entering**

**bool Entering()**

Returns true if the mouse was entering the window.

See also *wxMouseEvent::Leaving* (p. 1126).

### **wxMouseEvent::GetButton**

**int GetButton() const**

Returns the mouse button which generated this event or `wxMOUSE_BTN_NONE` if no button is involved (for mouse move, enter or leave event, for example). Otherwise `wxMOUSE_BTN_LEFT` is returned for the left button down, up and double click events, `wxMOUSE_BTN_MIDDLE` and `wxMOUSE_BTN_RIGHT` for the same events for the middle and the right buttons respectively.

### **wxMouseEvent::GetPosition**

**wxPoint GetPosition() const**

**void GetPosition(wxCoord\* x, wxCoord\* y) const**

**void GetPosition(long\* x, long\* y) const**

Sets \*x and \*y to the position at which the event occurred.

Returns the physical mouse position in pixels.

Note that if the mouse event has been artificially generated from a special keyboard combination (e.g. under Windows when the "menu" key is pressed), the returned position is wxDefaultPosition.

**wxMouseEvent::GetLogicalPosition**

**wxPoint GetLogicalPosition(const wxDC& dc) const**

Returns the logical mouse position in pixels (i.e. translated according to the translation set for the DC, which usually indicates that the window has been scrolled).

**wxMouseEvent::GetLinesPerAction**

**int GetLinesPerAction() const**

Returns the configured number of lines (or whatever) to be scrolled per wheel action. Defaults to three.

**wxMouseEvent::GetWheelRotation**

**int GetWheelRotation() const**

Get wheel rotation, positive or negative indicates direction of rotation. Current devices all send an event when rotation is at least +/-WheelDelta, but finer resolution devices can be created in the future. Because of this you shouldn't assume that one event is equal to 1 line, but you should be able to either do partial line scrolling or wait until several events accumulate before scrolling.

**wxMouseEvent::GetWheelDelta**

**int GetWheelDelta() const**

Get wheel delta, normally 120. This is the threshold for action to be taken, and one such action (for example, scrolling one increment) should occur for each delta.

**wxMouseEvent::GetX**

**long GetX() const**

Returns X coordinate of the physical mouse event position.

**wxMouseEvent::GetY**

**long GetY()**

Returns Y coordinate of the physical mouse event position.

**wxMouseEvent::IsButton****bool IsButton() const**

Returns true if the event was a mouse button event (not necessarily a button down event - that may be tested using *ButtonDown*).

**wxMouseEvent::IsPageScroll****bool IsPageScroll() const**

Returns true if the system has been setup to do page scrolling with the mouse wheel instead of line scrolling.

**wxMouseEvent::Leaving****bool Leaving() const**

Returns true if the mouse was leaving the window.

See also *wxMouseEvent::Entering* (p. 1124).

**wxMouseEvent::LeftDClick****bool LeftDClick() const**

Returns true if the event was a left double click.

**wxMouseEvent::LeftDown****bool LeftDown() const**

Returns true if the left mouse button changed to down.

**wxMouseEvent::LeftIsDown****bool LeftIsDown() const**

Returns true if the left mouse button is currently down, independent of the current event type.

Please notice that it is *not* the same as *LeftDown* (p. 1126) which returns `true` if the event was generated by the left mouse button being pressed. Rather, it simply describes the state of the left mouse button at the time when the event was generated (so while it will be true for a left click event, it can also be true for a right click if it happened while the left mouse button was pressed).

This event is usually used in the mouse event handlers which process "move mouse" messages to determine whether the user is (still) dragging the mouse.

**wxMouseEvent::LeftUp****bool LeftUp() const**

Returns true if the left mouse button changed to up.

**wxMouseEvent::MetaDown****bool MetaDown() const**

Returns true if the Meta key was down at the time of the event.

**wxMouseEvent::MiddleDClick****bool MiddleDClick() const**

Returns true if the event was a middle double click.

**wxMouseEvent::MiddleDown****bool MiddleDown() const**

Returns true if the middle mouse button changed to down.

**wxMouseEvent::MiddleIsDown****bool MiddleIsDown() const**

Returns true if the middle mouse button is currently down, independent of the current event type.

**wxMouseEvent::MiddleUp****bool MiddleUp() const**

Returns true if the middle mouse button changed to up.

**wxMouseEvent::Moving****bool Moving() const**

Returns true if this was a motion event and no mouse buttons were pressed. If any mouse button is held pressed, then this method returns *false* and *Dragging* (p. 1124) returns *true*.

**wxMouseEvent::RightDClick**

**bool RightDClick() const**

Returns true if the event was a right double click.

**wxMouseEvent::RightDown****bool RightDown() const**

Returns true if the right mouse button changed to down.

**wxMouseEvent::RightIsDown****bool RightIsDown() const**

Returns true if the right mouse button is currently down, independent of the current event type.

**wxMouseEvent::RightUp****bool RightUp() const**

Returns true if the right mouse button changed to up.

**wxMouseEvent::ShiftDown****bool ShiftDown() const**

Returns true if the shift key was down at the time of the event.

**wxMoveEvent**

A move event holds information about move change events.

**Derived from**

*wxEvent* (p. 572)

*wxObject* (p. 1148)

**Include files**

<wx/event.h>

**Event table macros**

To process a move event, use this event handler macro to direct input to a member function that takes a *wxMoveEvent* argument.

**EVT\_MOVE(func)**

Process a *wxEVT\_MOVE* event, which is generated when a window is moved.

**See also**



*wxPoint* (p. 1193), *Event handling overview* (p. 2077)

### **wxMoveEvent::wxMoveEvent**

**wxMoveEvent(const wxPoint& pt, int id = 0)**

Constructor.

### **wxMoveEvent::GetPosition**

**wxPoint GetPosition() const**

Returns the position of the window generating the move change event.

## **wxMultiChoiceDialog**

This class represents a dialog that shows a list of strings, and allows the user to select one or more.

### **Derived from**

*wxDialog* (p. 496)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### **Include files**

<wx/choicdlg.h>

### **See also**

*wxMultiChoiceDialog overview* (p. 2131), *wxSingleChoiceDialog* (p. 1435)

### **wxMultiChoiceDialog::wxMultiChoiceDialog**

**wxMultiChoiceDialog(wxWindow\* parent, const wxString& message, const wxString& caption, int n, const wxString\* choices, long style = wxCHOICEDLG\_STYLE, const wxPoint& pos = wxDefaultPosition)**

**wxMultiChoiceDialog(wxWindow\* parent, const wxString& message, const wxString& caption, const wxStringArray& choices, long style = wxCHOICEDLG\_STYLE, const wxPoint& pos = wxDefaultPosition)**

Constructor taking an array of wxString choices.

### **Parameters**

*parent*

Parent window.

*message*

Message to show on the dialog.

*caption*

The dialog caption.

*n*

The number of choices.

*choices*

An array of strings, or a string list, containing the choices.

*style*

A dialog style (bitlist) containing flags chosen from standard dialog styles and the following:

<b>wxOK</b>	Show an OK button.
<b>wxCANCEL</b>	Show a Cancel button.
<b>wxCENTRE</b>	Centre the message. Not Windows.

The default value is equivalent to **wxDEFAULT\_DIALOG\_STYLE | wxRESIZE\_BORDER | wxOK | wxCANCEL | wxCENTRE**.

*pos*

Dialog position. Not Windows.

### Remarks

Use *wxMultiChoiceDialog::ShowModal* (p. 1131) to show the dialog.

**wxPython note:** For Python the two parameters *n* and *choices* are collapsed into a multi parameter *choices* which is expected to be a Python list of strings.

**wxPerl note:** In wxPerl there is just an array reference in place of *n*.

### **wxMultiChoiceDialog::GetSelections**

**wxArrayInt GetSelections() const**

Returns array with indexes of selected items.

### **wxMultiChoiceDialog::SetSelections**

**void SetSelections(const wxArrayInt& selections) const**

Sets selected items from the array of selected items' indexes.

**wxMultiChoiceDialog::ShowModal**

**int ShowModal()**

Shows the dialog, returning either wxID\_OK or wxID\_CANCEL.

## wxMutex

A mutex object is a synchronization object whose state is set to signaled when it is not owned by any thread, and nonsignaled when it is owned. Its name comes from its usefulness in coordinating mutually-exclusive access to a shared resource as only one thread at a time can own a mutex object.

Mutexes may be recursive in the sense that a thread can lock a mutex which it had already locked before (instead of dead locking the entire process in this situation by starting to wait on a mutex which will never be released while the thread is waiting) but using them is not recommended and they are **not** recursive by default. The reason for this is that recursive mutexes are not supported by all Unix flavours and, worse, they cannot be used with *wxCondition* (p. 259).

For example, when several threads use the data stored in the linked list, modifications to the list should only be allowed to one thread at a time because during a new node addition the list integrity is temporarily broken (this is also called *program invariant*).

### Example

```
// this variable has an "s_" prefix because it is static: seeing
an "s_" in
// a multithreaded program is in general a good sign that you should
use a
// mutex (or a critical section)
static wxMutex *s_mutexProtectingTheGlobalData;

// we store some numbers in this global array which is presumably
used by
// several threads simultaneously
wxArrayInt s_data;

void MyThread::AddNewNode(int num)
{
    // ensure that no other thread accesses the list
    s_mutexProtectingTheGlobalList->Lock();

    s_data.Add(num);

    s_mutexProtectingTheGlobalList->Unlock();
}

// return true the given number is greater than all array elements
```

```
bool MyThread::IsGreater(int num)
{
    // before using the list we must acquire the mutex
    wxMutexLocker lock(s_mutexProtectingTheGlobalData);

    size_t count = s_data.Count();
    for ( size_t n = 0; n < count; n++ )
    {
        if ( s_data[n] > num )
            return false;
    }

    return true;
}
```

Notice how `wxMutexLocker` was used in the second function to ensure that the mutex is unlocked in any case: whether the function returns true or false (because the destructor of the local object *lock* is always called). Using this class instead of directly using `wxMutex` is, in general safer and is even more so if your program uses C++ exceptions.

### Constants

```
enum wxMutexType
{
    // normal mutex: try to always use this one
    wxMUTEX_DEFAULT,

    // recursive mutex: don't use these ones with wxCondition
    wxMUTEX_RECURSIVE
};
```

### Derived from

None.

### Include files

<wx/thread.h>

### See also

*wxThread* (p. 1670), *wxCondition* (p. 259), *wxMutexLocker* (p. 1133), *wxCriticalSection* (p. 294)

### wxMutex::wxMutex

**wxMutex**(wxMutexType *type* = *wxMUTEX\_DEFAULT*)

Default constructor.

**wxMutex::~~wxMutex****~wxMutex()**

Destroys the wxMutex object.

**wxMutex::Lock****wxMutexError Lock()**

Locks the mutex object.

**Return value**

One of:

**wxMUTEX\_NO\_ERROR**

There was no error.

**wxMUTEX\_DEAD\_LOCK**

A deadlock situation was detected.

**wxMutex::TryLock****wxMutexError TryLock()**

Tries to lock the mutex object. If it can't, returns immediately with an error.

**Return value**

One of:

**wxMUTEX\_NO\_ERROR**

There was no error.

**wxMUTEX\_BUSY**

The mutex is already locked by another thread.

**wxMutex::Unlock****wxMutexError Unlock()**

Unlocks the mutex object.

**Return value**

One of:

**wxMUTEX\_NO\_ERROR**

There was no error.

**wxMUTEX\_UNLOCKED**

The calling thread doesn't own the mutex.

**wxMutexLocker**

This is a small helper class to be used with *wxMutex* (p. 1131) objects. A *wxMutexLocker* acquires a mutex lock in the constructor and releases (or unlocks) the mutex in the

destructor making it much more difficult to forget to release a mutex (which, in general, will promptly lead to serious problems). See *wxMutex* (p. 1131) for an example of *wxMutexLocker* usage.

**Derived from**

None.

**Include files**

<wx/thread.h>

**See also**

*wxMutex* (p. 1131), *wxCriticalSectionLocker* (p. 295)

**wxMutexLocker::wxMutexLocker**

**wxMutexLocker(wxMutex& mutex)**

Constructs a *wxMutexLocker* object associated with *mutex* and locks it. Call *IsLocked* (p. 1134) to check if the mutex was successfully locked.

**wxMutexLocker::~~wxMutexLocker**

**~wxMutexLocker()**

Destructor releases the mutex if it was successfully acquired in the ctor.

**wxMutexLocker::IsOk**

**bool IsOk() const**

Returns true if mutex was acquired in the constructor, false otherwise.

**wxNode**

*wxNodeBase* is the node structure used in linked lists (see *wxList* (p. 966)) and derived classes. You should never use *wxNodeBase* class directly, however, because it works with untyped (`void *`) data and this is unsafe. Use *wxNodeBase*-derived classes which are automatically defined by `WX_DECLARE_LIST` and `WX_DEFINE_LIST` macros instead as described in *wxList* (p. 966) documentation (see example there). Also note that although there is a class called *wxNode*, it is defined for backwards compatibility only and usage of this class is strongly deprecated.

In the documentation below, the type *T* should be thought of as a "template" parameter: this is the type of data stored in the linked list or, in other words, the first argument of `WX_DECLARE_LIST` macro. Also, *wxNode* is written as *wxNode<T>* even though it isn't really a template class -- but it helps to think of it as if it were.

**Derived from**

None.

**Include files**

<wx/list.h>

**See also**

*wxList* (p. 966), *wxHashTable* (p. 807)

**wxNode<T>::GetData**

**T \* GetData() const**

Retrieves the client data pointer associated with the node.

**wxNode<T>::GetNext**

**wxNode<T> \* GetNext() const**

Retrieves the next node or NULL if this node is the last one.

**wxNode<T>::GetPrevious**

**wxNode<T> \* GetPrevious()**

Retrieves the previous node or NULL if this node is the first one in the list.

**wxNode<T>::SetData**

**void SetData(T \*data)**

Sets the data associated with the node (usually the pointer will have been set when the node was created).

**wxNode<T>::IndexOf**

**int IndexOf()**

Returns the zero-based index of this node within the list. The return value will be `wxNOT_FOUND` if the node has not been added to a list yet.

**wxNotebook**

This class represents a notebook control, which manages multiple windows with associated tabs.

To use the class, create a `wxNotebook` object and call *AddPage* (p. 1138) or *InsertPage* (p. 1142), passing a window to be used as the page. Do not explicitly delete the window for a page that is currently managed by `wxNotebook`.

**wxNotebookPage** is a typedef for `wxWindow`.

### Derived from

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/notebook.h>

### Window styles

<b>wxNB_TOP</b>	Place tabs on the top side.
<b>wxNB_LEFT</b>	Place tabs on the left side.
<b>wxNB_RIGHT</b>	Place tabs on the right side.
<b>wxNB_BOTTOM</b>	Place tabs under instead of above the notebook pages.
<b>wxNB_FIXEDWIDTH</b>	(Windows only) All tabs will have same width.
<b>wxNB_MULTILINE</b>	(Windows only) There can be several rows of tabs.
<b>wxNB_NOPAGETHEME</b>	(Windows only) Display a solid colour on notebook pages, and not a gradient, which can reduce performance.
<b>wxNB_FLAT</b>	(Windows CE only) Show tabs in a flat style.

The styles `wxNB_LEFT`, `RIGHT` and `BOTTOM` are not supported under Microsoft Windows XP when using visual themes.

See also *window styles overview* (p. 2089).

### Event handling

To process input from a notebook control, use the following event handler macros to direct input to member functions that take a *wxNotebookEvent* (p. 1144) argument.

**EVT\_NOTEBOOK\_PAGE\_CHANGED(id, func)** The page selection was changed. Processes a `wxEVT_COMMAND_NOTEBOOK_PAGE_CHANGED` event.

**EVT\_NOTEBOOK\_PAGE\_CHANGING(id, func)** The page selection is about to be changed. Processes a `wxEVT_COMMAND_NOTEBOOK_PAGE_CHANGING` event. This event can be *vetoed* (p.



1147).

### Page backgrounds

On Windows XP, the default theme paints a gradient on the notebook's pages. If you wish to suppress this theme, for aesthetic or performance reasons, there are three ways of doing it. You can use `wxNB_NOPAGETHEME` to disable themed drawing for a particular notebook, you can call `wxSystemOptions::SetOption` to disable it for the whole application, or you can disable it for individual pages by using `SetBackgroundColour`.

To disable themed pages globally:

```
wxSystemOptions::SetOption(wxT("msw.notebook.themed-background"),
0);
```

Set the value to 1 to enable it again.

To give a single page a solid background that more or less fits in with the overall theme, use:

```
wxColour col = notebook->GetThemeBackgroundColour();
if (col.Ok())
{
    page->SetBackgroundColour(col);
}
```

On platforms other than Windows, or if the application is not using Windows themes, `GetThemeBackgroundColour` will return an uninitialised colour object, and the above code will therefore work on all platforms.

### See also

*wxBookCtrl* (p. 2127), *wxNotebookEvent* (p. 1144), *wxImageList* (p. 933), *notebook sample* (p. 2036)

## **wxNotebook::wxNotebook**

### **wxNotebook()**

Default constructor.

**wxNotebook**(**wxWindow\*** *parent*, **wxWindowID** *id*, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = 0, **const wxString&** *name* = *wxNotebookNameStr*)

Constructs a notebook control.

Note that sometimes you can reduce flicker by passing the `wxCLIP_CHILDREN` window style.

### Parameters

*parent*

The parent window. Must be non-NULL.

*id*

The window identifier.

*pos*

The window position.

*size*

The window size.

*style*

The window style. See *wxNotebook* (p. 1135).

*name*

The name of the control (used only under Motif).

### **wxNotebook::~~wxNotebook**

**~wxNotebook()**

Destroys the *wxNotebook* object.

### **wxNotebook::AddPage**

**bool AddPage(wxNotebookPage\* page, const wxString& text, bool select = false, int imageld = -1)**

Adds a new page.

The call to this function may generate the page changing events.

#### **Parameters**

*page*

Specifies the new page.

*text*

Specifies the text for the new page.

*select*

Specifies whether the page should be selected.

*imageld*

Specifies the optional image index for the new page.

**Return value**

true if successful, false otherwise.

**Remarks**

Do not delete the page, it will be deleted by the notebook.

**See also**

*wxNotebook::InsertPage* (p. 1142)

**wxNotebook::AdvanceSelection**

**void AdvanceSelection**(bool *forward* = true)

Cycles through the tabs.

The call to this function generates the page changing events.

**wxNotebook::AssignImageList**

**void AssignImageList**(wxImageList\* *imageList*)

Sets the image list for the page control and takes ownership of the list.

**See also**

*wxImageList* (p. 933), *SetImageList* (p. 1143)

**wxNotebook::Create**

**bool Create**(wxWindow\* *parent*, wxWindowID *id*, const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size*, long *style* = 0, const wxString& *name* = wxNotebookNameStr)

Creates a notebook control. See *wxNotebook::wxNotebook* (p. 1137) for a description of the parameters.

**wxNotebook::DeleteAllPages**

**bool DeleteAllPages**()

Deletes all pages.

**wxNotebook::DeletePage**

**bool DeletePage**(size\_t *page*)

Deletes the specified page, and the associated window.

The call to this function generates the page changing events.

**wxNotebook::GetCurrentPage****wxWindow \* GetCurrentPage() const**

Returns the currently selected notebook page or `NULL`.

**wxNotebook::GetImageList****wxImageList\* GetImageList() const**

Returns the associated image list.

**See also**

*wxImageList* (p. 933), *wxNotebook::SetImageList* (p. 1143)

**wxNotebook::GetPage****wxNotebookPage\* GetPage(size\_t page)**

Returns the window at the given page position.

**wxNotebook::GetPageCount****size\_t GetPageCount() const**

Returns the number of pages in the notebook control.

**wxNotebook::GetPageImage****int GetPageImage(size\_t nPage) const**

Returns the image index for the given page.

**wxNotebook::GetPageText****wxString GetPageText(size\_t nPage) const**

Returns the string for the given page.

**wxNotebook::GetRowCount****int GetRowCount() const**

Returns the number of rows in the notebook control.

**wxNotebook::GetSelection**

**int GetSelection() const**

Returns the currently selected page, or -1 if none was selected.

Note that this method may return either the previously or newly selected page when called from the `EVT_NOTEBOOK_PAGE_CHANGED` handler depending on the platform and so `wxNotebookEvent::GetSelection` (p. 1145) should be used instead in this case.

**wxNotebook::GetThemeBackgroundColour****wxColour GetThemeBackgroundColour() const**

If running under Windows and themes are enabled for the application, this function returns a suitable colour for painting the background of a notebook page, and can be passed to `SetBackgroundColour`. Otherwise, an uninitialised colour will be returned.

**wxNotebook::HitTest****int HitTest(const wxPoint& pt, long \*flags = NULL)**

Returns the index of the tab at the specified position or `wxNOT_FOUND` if none. If *flags* parameter is non-NULL, the position of the point inside the tab is returned as well.

**Parameters**

*pt*

Specifies the point for the hit test.

*flags*

Return value for detailed information. One of the following values: **wxBK\_HITTEST\_NOWHERE**  
There was no tab under this point.

**wxBK\_HITTEST\_ONICON**                      The point was over an icon (currently wxMSW only).

**wxBK\_HITTEST\_ONLABEL**                    The point was over a label (currently wxMSW only).

**wxBK\_HITTEST\_ONITEM**                    The point was over an item, but not on the label or icon.

**wxBK\_HITTEST\_ONPAGE**                    The point was over a currently selected page, not over any tab. Note that this flag is present only if `wxNOT_FOUND` is returned.

**Return value**

Returns the zero-based tab index or `wxNOT_FOUND` if there is no tab is at the specified

position.

### **wxNotebook::InsertPage**

**bool InsertPage**(size\_t *index*, wxNotebookPage\* *page*, const wxString& *text*, bool *select* = false, int *imageId* = -1)

Inserts a new page at the specified position.

#### **Parameters**

*index*

Specifies the position for the new page.

*page*

Specifies the new page.

*text*

Specifies the text for the new page.

*select*

Specifies whether the page should be selected.

*imageId*

Specifies the optional image index for the new page.

#### **Return value**

true if successful, false otherwise.

#### **Remarks**

Do not delete the page, it will be deleted by the notebook.

#### **See also**

*wxNotebook::AddPage* (p. 1138)

### **wxNotebook::OnSelChange**

**void OnSelChange**(wxNotebookEvent& *event*)

An event handler function, called when the page selection is changed.

#### **See also**

*wxNotebookEvent* (p. 1144)

### **wxNotebook::RemovePage**

**bool RemovePage(size\_t page)**

Deletes the specified page, without deleting the associated window.

**wxNotebook::SetImageList**

**void SetImageList(wxImageList\* imageList)**

Sets the image list for the page control. It does not take ownership of the image list, you must delete it yourself.

**See also**

*wxImageList* (p. 933), *AssignImageList* (p. 1139)

**wxNotebook::SetPadding**

**void SetPadding(const wxSize& padding)**

Sets the amount of space around each page's icon and label, in pixels.

**NB:** The vertical padding cannot be changed in wxGTK.

**wxNotebook::SetPageSize**

**void SetPageSize(const wxSize& size)**

Sets the width and height of the pages.

**NB:** This method is currently not implemented for wxGTK.

**wxNotebook::SetPageImage**

**bool SetPageImage(size\_t page, int image)**

Sets the image index for the given page. *image* is an index into the image list which was set with *wxNotebook::SetImageList* (p. 1143).

**wxNotebook::SetPageText**

**bool SetPageText(size\_t page, const wxString& text)**

Sets the text for the given page.

**wxNotebook::SetSelection**

**int SetSelection(size\_t page)**

Sets the selection for the given page, returning the previous selection.

The call to this function generates the page changing events.

This function is deprecated and should not be used in new code. Please use the *ChangeSelection* (p. 1144) function instead.

**See also**

*wxNotebook::GetSelection* (p. 1140)

**wxNotebook::ChangeSelection**

**int ChangeSelection(size\_t page)**

Changes the selection for the given page, returning the previous selection.

The call to this function *does not* generate the page changing events. This is the only difference with *SetSelection* (p. 1143). See *this topic* (p. 2081) for more info.

**wxNotebookEvent**

This class represents the events generated by a notebook control: currently, there are two of them. The `PAGE_CHANGING` event is sent before the current page is changed. It allows the program to examine the current page (which can be retrieved with *GetOldSelection()* (p. 1145)) and to veto the page change by calling *Veto()* (p. 1147) if, for example, the current values in the controls of the old page are invalid.

The second event - `PAGE_CHANGED` - is sent after the page has been changed and the program cannot veto it any more, it just informs it about the page change.

To summarize, if the program is interested in validating the page values before allowing the user to change it, it should process the `PAGE_CHANGING` event, otherwise `PAGE_CHANGED` is probably enough. In any case, it is probably unnecessary to process both events at once.

**Derived from**

*wxNotifyEvent* (p. 1146)  
*wxCommandEvent* (p. 250)  
*wxEvent* (p. 572)  
*wxObject* (p. 1148)

**Include files**

<wx/notebook.h>

**Event handling**

To process input from a notebook control, use the following event handler macros to direct input to member functions that take a *wxNotebookEvent* (p. 1144) argument.

**EVT\_NOTEBOOK\_PAGE\_CHANGED(id, func)**    The page selection was changed.  
Processes a  
wxEVT\_COMMAND\_NOTEBOOK\_PAGE\_CHANGED event.



**EVT\_NOTEBOOK\_PAGE\_CHANGING(id, func)** The page selection is about to be changed. Processes a `wxEVT_COMMAND_NOTEBOOK_PAGE_CHANGING` event. This event can be *vetoed* (p. 1147).

**See also**

*wxNotebook* (p. 1135)

**wxNotebookEvent::wxNotebookEvent**

**wxNotebookEvent(wxEventType eventType = wxEVT\_NULL, int id = 0, int sel = -1, int oldSel = -1)**

Constructor (used internally by wxWidgets only).

**wxNotebookEvent::GetOldSelection**

**int GetOldSelection() const**

Returns the page that was selected before the change, -1 if none was selected.

**wxNotebookEvent::GetSelection**

**int GetSelection() const**

Returns the currently selected page, or -1 if none was selected.

**NB:** under Windows, `GetSelection()` will return the same value as `GetOldSelection()` (p. 1145) when called from `EVT_NOTEBOOK_PAGE_CHANGING` handler and not the page which is going to be selected. Also note that the values of selection and old selection returned for an event generated in response to a call to `wxNotebook::SetSelection` (p. 1143) shouldn't be trusted as they are currently inconsistent under different platforms (but in this case you presumably don't need them anyhow as you already have the corresponding information).

**wxNotebookEvent::SetOldSelection**

**void SetOldSelection(int page)**

Sets the id of the page selected before the change.

**wxNotebookEvent::SetSelection**

**void SetSelection(int page)**

Sets the selection member variable.

**See also**

*wxNotebookEvent::GetSelection* (p. 1145)

## **wxNotebookSizer**

**This class is deprecated and should not be used in new code! It is no longer needed, *wxNotebook* (p. 1135) control can be inserted into any sizer class and its minimal size will be determined correctly. See *wxSizer overview* (p. 2098) for more information.**

*wxNotebookSizer* is a specialized sizer to make sizers work in connection with using notebooks. This sizer is different from any other sizer as you must not add any children to it - instead, it queries the notebook class itself. The only thing this sizer does is to determine the size of the biggest page of the notebook and report an adjusted minimal size to a more toplevel sizer.

**Derived from**

*wxSizer* (p. 1444)  
*wxObject* (p. 1148)

**Include files**

<wx/sizer.h>

**See also**

*wxSizer* (p. 1444), *wxNotebook* (p. 1135), *Sizer overview* (p. 2098)

### **wxNotebookSizer::wxNotebookSizer**

**wxNotebookSizer(wxNotebook\* notebook)**

Constructor. It takes an associated notebook as its only parameter.

### **wxNotebookSizer::GetNotebook**

**wxNotebook\* GetNotebook()**

Returns the notebook associated with the sizer.

## **wxNotifyEvent**

This class is not used by the event handlers by itself, but is a base class for other event classes (such as *wxNotebookEvent* (p. 1144)).

It (or an object of a derived class) is sent when the controls state is being changed and allows the program to *Veto()* (p. 1147) this change if it wants to prevent it from happening.

**Derived from**

*wxCommandEvent* (p. 250)

*wxEvent* (p. 572)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

**Include files**

<wx/event.h>

**Event table macros**

None

**See also**

*wxNotebookEvent* (p. 1144)

**wxNotifyEvent::wxNotifyEvent**

**wxNotifyEvent(wxEvtHandler \*eventHandler, wxEventType eventType = wxEVT\_NULL, int id = 0)**

Constructor (used internally by wxWidgets only).

**wxNotifyEvent::Allow**

**void Allow()**

This is the opposite of *Veto()* (p. 1147): it explicitly allows the event to be processed. For most events it is not necessary to call this method as the events are allowed anyhow but some are forbidden by default (this will be mentioned in the corresponding event description).

**wxNotifyEvent::IsAllowed**

**bool IsAllowed() const**

Returns true if the change is allowed (*Veto()* (p. 1147) hasn't been called) or false otherwise (if it was).

**wxNotifyEvent::Veto**

**void Veto()**

Prevents the change announced by this event from happening.

It is in general a good idea to notify the user about the reasons for vetoing the change because otherwise the applications behaviour (which just refuses to do what the user wants) might be quite surprising.

## wxObject

This is the root class of all wxWidgets classes. It declares a virtual destructor which ensures that destructors get called for all derived class objects where necessary.

wxObject is the hub of a dynamic object creation scheme, enabling a program to create instances of a class only knowing its string class name, and to query the class hierarchy.

The class contains optional debugging versions of **new** and **delete**, which can help trace memory allocation and deallocation problems.

wxObject can be used to implement *reference counted* (p. 2046) objects, such as wxPen, wxBitmap and others (see *this list* (p. 2046)).

### See also

*wxClassInfo* (p. 190), *Debugging overview* (p. 2072), *wxObjectRefData* (p. 1151)

## wxObject::wxObject

**wxObject()**

**wxObject(const wxObject& other)**

Default and copy constructors.

## wxObject::~~wxObject

**wxObject()**

Destructor. Performs dereferencing, for those objects that use reference counting.

## wxObject::m\_refData

**wxObjectRefData\* m\_refData**

Pointer to an object which is the object's reference-counted data.

### See also

*wxObject::Ref* (p. 1150), *wxObject::UnRef* (p. 1150), *wxObject::SetRefData* (p. 1150), *wxObject::GetRefData* (p. 1149), *wxObjectRefData* (p. 1151)

## wxObject::Dump

**void Dump(ostream& stream)**

A virtual function that may be redefined by derived classes to allow dumping of memory states.

This function is only defined in debug build and doesn't exist at all if `__WXDEBUG__` is not defined.

### Parameters

*stream*

Stream on which to output dump information.

### Remarks

Currently wxWidgets 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 in wxObject just writes the class name of the object.

### **wxObject::GetClassInfo**

**wxClassInfo \* GetClassInfo()**

This virtual function is redefined for every class that requires run-time type information, when using DECLARE\_CLASS macros.

### **wxObject::GetRefData**

**wxObjectRefData\* GetRefData() const**

Returns the `m_refData` pointer.

### See also

*wxObject::Ref* (p. 1150), *wxObject::UnRef* (p. 1150), *wxObject::m\_refData* (p. 1148), *wxObject::SetRefData* (p. 1150), *wxObjectRefData* (p. 1151)

### **wxObject::IsKindOf**

**bool IsKindOf(wxClassInfo \*info)**

Determines whether this class is a subclass of (or the same class as) the given class.

### Parameters

*info*

A pointer to a class information object, which may be obtained by using the CLASSINFO macro.

### Return value

true if the class represented by *info* is the same class as this one or is derived from it.

### Example

```
bool tmp = obj->IsKindOf(CLASSINFO(wxFrame));
```

### **wxObject::IsSameAs**

**bool IsSameAs(const wxObject& obj)**

Returns `true` if this object has the same data pointer as *obj*. Notice that `true` is returned if the data pointers are `NULL` in both objects.

This function only does a *shallow* comparison, i.e. it doesn't compare the objects pointed to by the data pointers of these objects.

### **wxObject::Ref**

**void Ref(const wxObject& clone)**

Makes this object refer to the data in *clone*.

#### **Parameters**

*clone*

The object to 'clone'.

#### **Remarks**

First this function calls *wxObject::UnRef* (p. 1150) on itself to decrement (and perhaps free) the data it is currently referring to.

It then sets its own `m_refData` to point to that of *clone*, and increments the reference count inside the data.

#### **See also**

*wxObject::UnRef* (p. 1150), *wxObject::m\_refData* (p. 1148), *wxObject::SetRefData* (p. 1150), *wxObject::GetRefData* (p. 1149), *wxObjectRefData* (p. 1151)

### **wxObject::SetRefData**

**void SetRefData(wxObjectRefData\* data)**

Sets the `m_refData` pointer.

#### **See also**

*wxObject::Ref* (p. 1150), *wxObject::UnRef* (p. 1150), *wxObject::m\_refData* (p. 1148), *wxObject::GetRefData* (p. 1149), *wxObjectRefData* (p. 1151)

### **wxObject::UnRef**

**void UnRef()**

Decrements the reference count in the associated data, and if it is zero, deletes the data. The **m\_refData** member is set to NULL.

**See also**

*wxObject::Ref* (p. 1150), *wxObject::m\_refData* (p. 1148), *wxObject::SetRefData* (p. 1150), *wxObject::GetRefData* (p. 1149), *wxObjectRefData* (p. 1151)

**wxObject::UnShare****void UnShare()**

Ensure that this object's data is not shared with any other object.

if we have no data, it is created using *CreateRefData()* below, if we have shared data it is copied using *CloneRefData()*, otherwise nothing is done.

**wxObject::operator new**

**void \* new**(*size\_t size*, **const wxString& filename** = NULL, **int lineNum** = 0)

The *new* operator is defined for debugging versions of the library only, when the identifier `__WXDEBUG__` is defined. 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 `__WXDEBUG__` is defined. It takes over memory deallocation, allowing *wxDebugContext* operations.

**wxObjectRefData**

This class is used to store reference-counted data. Derive classes from this to store your own data. When retrieving information from a **wxObject**'s reference data, you will need to cast to your own derived class.

**Friends**

*wxObject* (p. 1148)

**See also**

*wxObject* (p. 1148)

**wxObjectRefData::wxObjectRefData**

**wxObjectRefData()**

Default constructor. Initialises the **m\_count** member to 1.

**wxObjectRefData::~~wxObjectRefData****wxObjectRefData()**

Destructor.

**wxObjectRefData::GetRefCount****int GetRefCount() const**

Returns the reference count associated with this shared data. When this goes to zero during a *wxObject::UnRef* (p. 1150), an object can delete this **wxObjectRefData** object.

**wxOwnerDrawnComboBox**

*wxOwnerDrawnComboBox* is a combobox with owner-drawn list items. In essence, it is a *wxComboCtrl* (p. 232) with *wxVListBox* (p. 1783) popup and *wxControlWithItems* (p. 286) interface.

Implementing item drawing and measuring is similar to *wxVListBox* (p. 1783). Application needs to subclass *wxOwnerDrawnComboBox* and implement *OnDrawItem()* (p. 1155), *OnMeasureItem()* (p. 1156) and *OnMeasureItemWidth()* (p. 1156).

**Derived from**

*wxComboCtrl* (p. 232)  
*wxControlWithItems* (p. 286)  
*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<odcombo.h>

**Window styles****wxODCB\_DCLICK\_CYCLES**

Double-clicking cycles item if *wxCB\_READONLY* is also used. Synonymous with *wxCC\_SPECIAL\_DCLICK*.

**wxODCB\_STD\_CONTROL\_PAINT**

Control itself is not custom painted using *OnDrawItem*. Even if this style is not used, writable *wxOwnerDrawnComboBox* is never custom painted unless *SetCustomPaintWidth* is called.



See also *wxComboCtrl* window styles (p. 232) and base *window* styles overview (p. 2089).

### Event handling

<b>EVT_COMBOBOX(id, func)</b>	Process a wxEVT_COMMAND_COMBOBOX_SELECTED event, when an item on the list is selected. Note that calling <i>GetValue</i> (p. 240) returns the new value of selection.
-------------------------------	--

See also events emitted by *wxComboCtrl* (p. 232).

### See also

*wxComboCtrl* (p. 232), *wxComboBox* (p. 225), *wxVListBox* (p. 1783), *wxCommandEvent* (p. 250)

## wxOwnerDrawnComboBox::wxOwnerDrawnComboBox

**wxOwnerDrawnComboBox()**

Default constructor.

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

**wxOwnerDrawnComboBox(wxWindow\* parent, wxWindowID id, const wxString& value, const wxPoint& pos, const wxSize& size, const wxString& choices, long style = 0, const wxValidator& validator = wxDefaultValidator, const wxString& name = "comboBox")**

Constructor, creating and showing a owner-drawn combobox.

### Parameters

*parent*

Parent window. Must not be NULL.

*id*

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

*value*

Initial selection string. An empty string indicates no selection.

*pos*

Window position.

*size*

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

*n*

Number of strings with which to initialise the control.

*choices*

An array of strings with which to initialise the control.

*style*

Window style. See *wxOwnerDrawnComboBox* (p. 1152).

*validator*

Window validator.

*name*

Window name.

### See also

*wxOwnerDrawnComboBox::Create* (p. 1154), *wxValidator* (p. 1767)

### **wxOwnerDrawnComboBox::~wxOwnerDrawnComboBox**

**~wxOwnerDrawnComboBox()**

Destructor, destroying the owner-drawn combobox.

### **wxOwnerDrawnComboBox::Create**

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

**bool Create(wxWindow\* parent, wxWindowID id, const wxString& value, const wxPoint& pos, const wxSize& size, const wxString& choices, long style = 0, const wxValidator& validator = wxDefaultValidator, const wxString& name = "comboBox")**

Creates the combobox for two-step construction. Derived classes should call or replace this function. See *wxOwnerDrawnComboBox::wxOwnerDrawnComboBox* (p. 1153) for further details.

### **wxOwnerDrawnComboBox::GetWidestItem**

**int GetWidestItem() const**

Returns index to the widest item in the list.

**wxOwnerDrawnComboBox::GetWidestItemWidth****int GetWidestItemWidth() const**

Returns width of the widest item in the list.

**wxOwnerDrawnComboBox::OnDrawBackground****void OnDrawBackground(wxDC& dc, const wxRect& rect, int item, int flags) const**

This method is used to draw the items background and, maybe, a border around it.

The base class version implements a reasonable default behaviour which consists in drawing the selected item with the standard background colour and drawing a border around the item if it is either selected or current.

**Remarks**

*flags* has the same meaning as with *OnDrawItem* (p. 1155).

**wxOwnerDrawnComboBox::OnDrawItem****void OnDrawItem(wxDC& dc, const wxRect& rect, int item, int flags) const**

The derived class may implement this function to actually draw the item with the given index on the provided DC. If function is not implemented, the item text is simply drawn, as if the control was a normal combobox.

**Parameters**

*dc*

The device context to use for drawing

*rect*

The bounding rectangle for the item being drawn (DC clipping region is set to this rectangle before calling this function)

*item*

The index of the item to be drawn

*flags*

Combines any of the following flag values:

wxODCB\_PAINTING\_CONTROL

Combo control is being painted, instead of a list item. Argument item may be wxNOT\_FOUND in

this case.

`wxODCB_PAINTING_SELECTED`

An item with selection background is being painted. DC text colour should already be correct.

### **wxOwnerDrawnComboBox::OnMeasureItem**

**wxCoord OnMeasureItem(size\_t item) const**

The derived class may implement this method to return the height of the specified item (in pixels).

The default implementation returns text height, as if this control was a normal combobox.

### **wxOwnerDrawnComboBox::OnMeasureItemWidth**

**wxCoord OnMeasureItemWidth(size\_t item) const**

The derived class may implement this method to return the width of the specified item (in pixels). If -1 is returned, then the item text width is used.

The default implementation returns -1.

## **wxOutputStream**

wxOutputStream is an abstract base class which may not be used directly.

### **Derived from**

*wxStreamBase* (p. 1544)

### **Include files**

<wx/stream.h>

### **wxOutputStream::wxOutputStream**

**wxOutputStream()**

Creates a dummy wxOutputStream object.

### **wxOutputStream::~~wxOutputStream**

**~wxOutputStream()**

Destructor.

### **wxOutputStream::Close**

**bool Close()**

Closes the stream, returning `false` if an error occurs. The stream is closed implicitly in the destructor if `Close()` is not called explicitly.

If this stream wraps another stream or some other resource such as a file, then the underlying resource is closed too if it is owned by this stream, or left open otherwise.

**wxOutputStream::LastWrite****size\_t LastWrite() const**

Returns the number of bytes written during the last `Write()` (p. 1157). It may return 0 even if there is no error on the stream if it is only temporarily impossible to write to it.

**wxOutputStream::PutC****void PutC(char c)**

Puts the specified character in the output queue and increments the stream position.

**wxOutputStream::SeekO****off\_t SeekO(off\_t pos, wxSeekMode mode = wxFromStart)**

Changes the stream current position.

**Parameters**

*pos*

Offset to seek to.

*mode*

One of `wxFromStart`, `wxFromEnd`, `wxFromCurrent`.

**Return value**

The new stream position or `wxInvalidOffset` on error.

**wxOutputStream::TellO****off\_t TellO() const**

Returns the current stream position.

**wxOutputStream::Write****wxOutputStream& Write(const void \*buffer, size\_t size)**

Writes up to the specified amount of bytes using the data of *buffer*. Note that not all data

can always be written so you must check the number of bytes really written to the stream using *LastWrite()* (p. 1157) when this function returns. In some cases (for example a write end of a pipe which is currently full) it is even possible that there is no errors and zero bytes have been written.

This function returns a reference on the current object, so the user can test any states of the stream right away.

#### **wxOutputStream& Write(wxInputStream& stream\_in)**

Reads data from the specified input stream and stores them in the current stream. The data is read until an error is raised by one of the two streams.

## **wxPageSetupDialog**

This class represents the page setup common dialog. In MSW, the page setup dialog is standard from Windows 95 on, replacing the print setup dialog (which is retained in Windows and wxWidgets 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.

On Macintosh, the native page setup dialog is used, which lets you select paper size and orientation but it does not let you change the page margins.

On other platforms, a generic dialog is used.

When the dialog has been closed, you need to query the *wxPageSetupDialogData* (p. 1159) object associated with the dialog.

Note that the OK and Cancel buttons do not destroy the dialog; this must be done by the application.

#### **Derived from**

*wxDialog* (p. 496)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

#### **Include files**

<wx/printdlg.h>

#### **See also**

*Printing framework overview* (p. 2145), *wxPrintDialog* (p. 1206), *wxPageSetupDialogData* (p. 1159)

**wxPageSetupDialog::wxPageSetupDialog****wxPageSetupDialog(wxWindow\* parent, wxPageSetupDialogData\* 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****~wxPageSetupDialog()**

Destructor.

**wxPageSetupDialog::GetPageSetupData****wxPageSetupDialogData& GetPageSetupData()**

Returns the *page setup data* (p. 1159) associated with the dialog.

**wxPageSetupDialog::ShowModal****int ShowModal()**

Shows the dialog, returning wxID\_OK if the user pressed OK, and wxID\_CANCEL otherwise.

**wxPageSetupDialogData**

This class holds a variety of information related to *wxPageSetupDialog* (p. 1158).

It contains a *wxPrintData* (p. 1200) member which is used to hold basic printer configuration data (as opposed to the user-interface configuration settings stored by *wxPageSetupDialogData*).

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/cmndata.h>

**See also**

*Printing framework overview* (p. 2145), *wxPageSetupDialog* (p. 1158)

**wxPageSetupDialogData::wxPageSetupDialogData****wxPageSetupDialogData()**

Default constructor.

**wxPageSetupDialogData(wxPageSetupDialogData& data)**

Copy constructor.

**wxPageSetupDialogData(wxPrintData& printData)**

Construct an object from a print data object.

**wxPageSetupDialogData::~~wxPageSetupDialogData**

**~wxPageSetupDialogData()**

Destructor.

**wxPageSetupDialogData::EnableHelp**

**void EnableHelp(bool flag)**

Enables or disables the 'Help' button (Windows only).

**wxPageSetupDialogData::EnableMargins**

**void EnableMargins(bool flag)**

Enables or disables the margin controls (Windows only).

**wxPageSetupDialogData::EnableOrientation**

**void EnableOrientation(bool flag)**

Enables or disables the orientation control (Windows only).

**wxPageSetupDialogData::EnablePaper**

**void EnablePaper(bool flag)**

Enables or disables the paper size control (Windows only).

**wxPageSetupDialogData::EnablePrinter**

**void EnablePrinter(bool flag)**

Enables or disables the **Printer** button, which invokes a printer setup dialog.

**wxPageSetupDialogData::GetDefaultMinMargins**

**bool GetDefaultMinMargins() const**

Returns true if the page setup dialog will take its minimum margin values from the currently



selected printer properties. Windows only.

**wxPageSetupDialogData::GetEnableMargins****bool GetEnableMargins() const**

Returns true if the margin controls are enabled (Windows only).

**wxPageSetupDialogData::GetEnableOrientation****bool GetEnableOrientation() const**

Returns true if the orientation control is enabled (Windows only).

**wxPageSetupDialogData::GetEnablePaper****bool GetEnablePaper() const**

Returns true if the paper size control is enabled (Windows only).

**wxPageSetupDialogData::GetEnablePrinter****bool GetEnablePrinter() const**

Returns true if the printer setup button is enabled.

**wxPageSetupDialogData::GetEnableHelp****bool GetEnableHelp() const**

Returns true if the printer setup button is enabled.

**wxPageSetupDialogData::GetDefaultInfo****bool GetDefaultInfo() const**

Returns true if the dialog will simply return default printer information (such as orientation) instead of showing a dialog. Windows only.

**wxPageSetupDialogData::GetMarginTopLeft****wxPoint GetMarginTopLeft() const**

Returns the left (x) and top (y) margins in millimetres.

**wxPageSetupDialogData::GetMarginBottomRight****wxPoint GetMarginBottomRight() const**

Returns the right (x) and bottom (y) margins in millimetres.

**wxPageSetupDialogData::GetMinMarginTopLeft****wxPoint GetMinMarginTopLeft() const**

Returns the left (x) and top (y) minimum margins the user can enter (Windows only). Units are in millimetres

**wxPageSetupDialogData::GetMinMarginBottomRight****wxPoint GetMinMarginBottomRight() const**

Returns the right (x) and bottom (y) minimum margins the user can enter (Windows only). Units are in millimetres

**wxPageSetupDialogData::GetPaperId****wxPaperSize GetPaperId() const**

Returns the paper id (stored in the internal `wxPrintData` object).

For further information, see `wxPrintData::SetPaperId` (p. 1204).

**wxPageSetupDialogData::GetPaperSize****wxSize GetPaperSize() const**

Returns the paper size in millimetres.

**wxPageSetupDialogData::GetPrintData****wxPrintData& GetPrintData()**

Returns a reference to the *print data* (p. 1200) associated with this object.

**wxPageSetupDialogData::IsOk****bool IsOk() const**

Returns true if the print data associated with the dialog data is valid. This can return false on Windows if the current printer is not set, for example. On all other platforms, it returns true.

**wxPageSetupDialogData::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.

**wxPageSetupDialogData::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. Units are in millimetres

**wxPageSetupDialogData::SetMarginTopLeft****void SetMarginTopLeft(const wxPoint& *pt*)**

Sets the left (x) and top (y) margins in millimetres.

**wxPageSetupDialogData::SetMarginBottomRight****void SetMarginBottomRight(const wxPoint& *pt*)**

Sets the right (x) and bottom (y) margins in millimetres.

**wxPageSetupDialogData::SetMinMarginTopLeft****void SetMinMarginTopLeft(const wxPoint& *pt*)**

Sets the left (x) and top (y) minimum margins the user can enter (Windows only). Units are in millimetres.

**wxPageSetupDialogData::SetMinMarginBottomRight****void SetMinMarginBottomRight(const wxPoint& *pt*)**

Sets the right (x) and bottom (y) minimum margins the user can enter (Windows only). Units are in millimetres.

**wxPageSetupDialogData::SetPaperId****void SetPaperId(wxPaperSize& *id*)**

Sets the paper size id. For further information, see *wxPrintData::SetPaperId* (p. 1204).

Calling this function overrides the explicit paper dimensions passed in *wxPageSetupDialogData::SetPaperSize* (p. 1163).

**wxPageSetupDialogData::SetPaperSize****void SetPaperSize(const wxSize& *size*)**

Sets the paper size in millimetres. If a corresponding paper id is found, it will be set in the internal *wxPrintData* object, otherwise the paper size overrides the paper id.

**wxPageSetupDialogData::SetPrintData****void SetPrintData(const wxPrintData& *printData*)**

Sets the *print data* (p. 1200) associated with this object.

**wxPageSetupDialogData::operator =**

**void operator =(const wxPrintData& data)**

Assigns print data to this object.

**void operator =(const wxPageSetupDialogData& data)**

Assigns page setup data to this object.

## wxPaintDC

A wxPaintDC must be constructed if an application wishes to paint on the client area of a window from within an **OnPaint** event. This should normally be constructed as a temporary stack object; don't store a wxPaintDC object. If you have an OnPaint handler, you *must* create a wxPaintDC object within it even if you don't actually use it.

Using wxPaintDC within OnPaint is important because it automatically sets the clipping area to the damaged area of the window. Attempts to draw outside this area do not appear.

To draw on a window from outside **OnPaint**, construct a wxClientDC (p. 193) object.

To draw on the whole window including decorations, construct a wxWindowDC (p. 1855) object (Windows only).

### Derived from

wxWindowDC (p. 1855)

wxDC (p. 456)

### Include files

<wx/dcclient.h>

### See also

wxDC (p. 456), wxMemoryDC (p. 1069), wxPaintDC (p. 1164), wxWindowDC (p. 1855), wxScreenDC (p. 1407)

## wxPaintDC::wxPaintDC

**wxPaintDC(wxWindow\* window)**

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

## wxPaintEvent

A paint event is sent when a window's contents needs to be repainted.

Please notice that in general it is impossible to change the drawing of a standard control (such as *wxBUTTON* (p. 164)) and so you shouldn't attempt to handle paint events for them as even if it might work on some platforms, this is inherently not portable and won't work everywhere.

### Derived from

*wxEvent* (p. 572)

*wxObject* (p. 1148)

### Include files

<wx/event.h>

### Event table macros

To process a paint event, use this event handler macro to direct input to a member function that takes a *wxPaintEvent* argument.

**EVT\_PAINT(func)**                      Process a *wxEVT\_PAINT* event.

### See also

*Event handling overview* (p. 2077)

### Remarks

Note that In a paint event handler, the application must *always* create a *wxPaintDC* (p. 1164) object, even if you do not use it. Otherwise, under MS Windows, refreshing for this and other windows will go wrong.

For example:

```
void MyWindow::OnPaint(wxPaintEvent& event)
{
    wxPaintDC dc(this);

    DrawMyDocument(dc);
}
```

You can optimize painting by retrieving the rectangles that have been damaged and only repainting these. The rectangles are in terms of the client area, and are unscrolled, so you will need to do some calculations using the current view position to obtain logical, scrolled units.

Here is an example of using the *wxRegionIterator* (p. 1269) class:

```
// Called when window needs to be repainted.
void MyWindow::OnPaint(wxPaintEvent& event)
{
    wxPaintDC dc(this);

    // Find Out where the window is scrolled to
    int vbX,vbY;                      // Top left corner of client
```

```
GetViewStart(&vbX,&vbY);

int vX,vY,vW,vH;           // Dimensions of client area in pixels
wxRegionIterator upd(GetUpdateRegion()); // get the update rect
list

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

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

    // Repaint this rectangle
    ...some code...

    upd ++ ;
}
}
```

## **wxPaintEvent::wxPaintEvent**

**wxPaintEvent**(int *id* = 0)

Constructor.

## **wxPalette**

A palette is a table that maps pixel values to RGB colours. It allows the colours of a low-depth bitmap, for example, to be mapped to the available colours in a display. The notion of palettes is becoming more and more obsolete nowadays and only the MSW port is still using a native palette. All other ports use generic code which is basically just an array of colours.

It is likely that in the future the only use for palettes within wxWidgets will be for representing colour indices from images (such as GIF or PNG). The image handlers for these formats have been modified to create a palette if there is such information in the original image file (usually 256 or less colour images). See *wxImage* (p. 906) for more information.

### **Derived from**

*wxGDIObject* (p. 709)  
*wxObject* (p. 1148)

### **Include files**

<wx/palette.h>

## Predefined objects

Objects:

### **wxNullPalette**

### **See also**

*wxDC::SetPalette* (p. 475), *wxBitmap* (p. 123)

## **wxPalette::wxPalette**

### **wxPalette()**

Default constructor.

### **wxPalette(const wxPalette& palette)**

Copy constructor, uses *reference counting* (p. 2046).

### **wxPalette(int n, const unsigned char\* red, const unsigned char\* green, const unsigned char\* blue)**

Creates a palette from arrays of size *n*, one for each red, blue or green component.

### **Parameters**

*palette*

A pointer or reference to the palette to copy.

*n*

The number of indices in the palette.

*red*

An array of red values.

*green*

An array of green values.

*blue*

An array of blue values.

### **See also**

*wxPalette::Create* (p. 1168)

**wxPerl note:** In wxPerl the third constructor form takes as parameters 3 array references ( they must be of the same length ).

**wxPalette::~~wxPalette****~wxPalette()**

Destructor. See *reference-counted object destruction* (p. 2046) for more info.

**wxPalette::Create****bool Create(int *n*, const unsigned char\* *red*, const unsigned char\* *green*, const unsigned char\* *blue*)**

Creates a palette from arrays of size *n*, one for each red, blue or green component.

**Parameters***n*

The number of indices in the palette.

*red*

An array of red values.

*green*

An array of green values.

*blue*

An array of blue values.

**Return value**

true if the creation was successful, false otherwise.

**See also**

*wxPalette::wxPalette* (p. 1167)

**wxPalette::GetColoursCount****int GetColoursCount() const**

Returns number of entries in palette.

**wxPalette::GetPixel****int GetPixel(unsigned char *red*, unsigned char *green*, unsigned char *blue*) const**

Returns a pixel value (index into the palette) for the given RGB values.

**Parameters***red*



Red value.

*green*

Green value.

*blue*

Blue value.

### **Return value**

The nearest palette index or `wxNOT_FOUND` for unexpected errors.

### **See also**

`wxPalette::GetRGB` (p. 1169)

### **wxPalette::GetRGB**

**bool** `GetRGB`(int *pixel*, const unsigned char\* *red*, const unsigned char\* *green*, const unsigned char\* *blue*) const

Returns RGB values for a given palette index.

### **Parameters**

*pixel*

The palette index.

*red*

Receives the red value.

*green*

Receives the green value.

*blue*

Receives the blue value.

### **Return value**

true if the operation was successful.

### **See also**

`wxPalette::GetPixel` (p. 1168)

**wxPerl note:** In wxPerl this method takes only the `pixel` parameter and returns a 3-element list ( or the empty list upon failure ).

### **wxPalette::IsOk**

**bool IsOk() const**

Returns true if palette data is present.

**wxPalette::operator =**

**wxPalette& operator =(const wxPalette& palette)**

Assignment operator, using *reference counting* (p. 2046).

## **wxPanel**

A panel is a window on which controls are placed. It is usually placed within a frame. It contains minimal extra functionality over and above its parent class `wxWindow`; its main purpose is to be similar in appearance and functionality to a dialog, but with the flexibility of having any window as a parent.

*Note:* if not all characters are being intercepted by your `OnKeyDown` or `OnChar` handler, it may be because you are using the `wxTAB_TRAVERSAL` style, which grabs some keypresses for use by child controls.

**Derived from**

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

**Include files**

<wx/panel.h>

**Window styles**

There are no specific styles for this window.

See also *window styles overview* (p. 2089).

**Remarks**

By default, a panel has the same colouring as a dialog.

**See also**

*wxDialog* (p. 496)

**wxPanel::wxPanel****wxPanel()**

Default constructor.

```
wxPanel(wxWindow* parent, wxWindowID id = wxID_ANY, const wxPoint& pos =  
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxTAB_TRAVERSAL, const wxString& name = "panel")
```

Constructor.

### Parameters

*parent*

The parent window.

*id*

An identifier for the panel. A value of -1 is taken to mean a default.

*pos*

The panel position. A value of (-1, -1) indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform.

*size*

The panel size. A value of (-1, -1) indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform.

*style*

The window style. See *wxPanel* (p. 1170).

*name*

Used to associate a name with the window, allowing the application user to set Motif resource values for individual dialog boxes.

### See also

*wxPanel::Create* (p. 1171)

### **wxPanel::~~wxPanel**

**~wxPanel()**

Destructor. Deletes any child windows before deleting the physical window.

### **wxPanel::Create**

```
bool Create(wxWindow* parent, wxWindowID id = wxID_ANY, const wxPoint& pos =  
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxTAB_TRAVERSAL, const wxString& name = "panel")
```

Used for two-step panel construction. See *wxPanel::wxPanel* (p. 1170) for details.

**wxPanel::InitDialog****void InitDialog()**

Sends a *wxInitDialogEvent* (p. 941), which in turn transfers data to the dialog via validators.

**See also**

*wxInitDialogEvent* (p. 941)

**wxPanel::OnSysColourChanged****void OnSysColourChanged(wxSysColourChangedEvent& event)**

The default handler for `wxEVT_SYS_COLOUR_CHANGED`.

**Parameters**

*event*

The colour change event.

**Remarks**

Changes the panel's colour to conform to the current settings (Windows only). Add an event table entry for your panel class if you wish the behaviour to be different (such as keeping a user-defined background colour). If you do override this function, call `wxEvent::Skip` to propagate the notification to child windows and controls.

**See also**

*wxSysColourChangedEvent* (p. 1590)

**wxPanel::SetFocus****virtual void SetFocus()**

Overrides *wxWindow::SetFocus* (p. 1839). This method uses the (undocumented) mix-in class *wxControlContainer* which manages the focus and TAB logic for controls which usually have child controls. In practice, if you call this method and the control has at least one child window, the focus will be given to the child window.

**See also**

*wxFocusEvent* (p. 655) *wxWindow::SetFocus* (p. 1839)

**wxPanel::SetFocusIgnoringChildren****virtual void SetFocusIgnoringChildren()**

In contrast to *wxPanel::SetFocus* (p. 1172) (see above) this will set the focus to the panel even if there are child windows in the panel. This is only rarely needed.

**See also**

*wxFocusEvent* (p. 655) *wxPanel::SetFocus* (p. 1172)

## **wxPasswordEntryDialog**

This class represents a dialog that requests a one-line password string from the user. It is implemented as a generic *wxWidgets* dialog.

**Derived from**

*wxTextEntryDialog* (p. 1655)  
*wxDialog* (p. 496)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/textdlg.h>

**See also**

*wxPassowrdEntryDialog* overview (p. 2131)

### **wxPasswordEntryDialog::wxPasswordEntryDialog**

**wxPasswordEntryDialog**(*wxWindow\** *parent*, **const** *wxString&* *message*, **const** *wxString&* *caption* = "Enter password", **const** *wxString&* *defaultValue* = "", **long** *style* = *wxOK* | *wxCANCEL* | *wxCENTRE*, **const** *wxPoint&* *pos* = *wxDefaultPosition*)

Constructor. Use *wxTextEntryDialog::ShowModal* (p. 1657) to show the dialog.

**Parameters**

*parent*

Parent window.

*message*

Message to show on the dialog.

*defaultValue*

The default value, which may be the empty string.

*style*

A dialog style, specifying the buttons (*wxOK*, *wxCANCEL*) and an optional *wxCENTRE* style. You do not need to specify the **wxTE\_PASSWORD** style, it is

always applied.

*pos*

Dialog position.

## **wxPathList**

The path list is a convenient way of storing a number of directories, and when presented with a filename without a directory, searching for an existing file in those directories.

Be sure to look also at *wxStandardPaths* (p. 1522) if you only want to search files in some standard paths.

### **Derived from**

*wxArrayString* (p. 83)

### **Include files**

<wx/filefn.h>

### **See also**

*wxArrayString* (p. 83), *wxStandardPaths* (p. 1522), *wxFileName* (p. 609)

## **wxPathList::wxPathList**

**wxPathList()**

Empty constructor.

**wxPathList(const wxArrayString& arr)**

Constructs the object calling the *Add* (p. 1174) function.

## **wxPathList::AddEnvList**

**void AddEnvList(const wxString& env\_variable)**

Finds the value of the given environment variable, and adds all paths to the path list. Useful for finding files in the `PATH` variable, for example.

## **wxPathList::Add**

**bool Add(const wxString& path)**

**void Add(const wxArrayString& arr)**

The first form adds the given directory to the path list, if the path is not already in the list. If

the path cannot be normalized for some reason, it returns `false`.

The second form just calls the first form on all elements of the given array.

The *path* is always considered a directory but no existence checks will be done on it (because if it doesn't exist, it could be created later and thus result a valid path when *FindValidPath* (p. 1175) is called).

**Note:** if the given path is relative, it won't be made absolute before adding it (this is why *FindValidPath* (p. 1175) may return relative paths).

### **wxPathList::EnsureFileAccessible**

**bool EnsureFileAccessible(const wxString& filename)**

Given a full filename (with path), calls *Add* (p. 1174) with the path of the file.

### **wxPathList::FindAbsolutePath**

**wxString FindAbsolutePath(const wxString& file) const**

Like *FindValidPath* (p. 1175) but this function always returns an absolute path (eventually prepending the current working directory to the value returned *FindValidPath* (p. 1175)) or an empty string.

### **wxPathList::FindValidPath**

**wxString FindValidPath(const wxString& file) const**

Searches the given file in all paths stored in this class. The first path which concatenated to the given string points to an existing file (see *wxFileExists* (p. 595)) is returned.

If the file wasn't found in any of the stored paths, an empty string is returned.

The given string must be a file name, eventually with a path prefix (if the path prefix is absolute, only its name will be searched); i.e. it must not end with a directory separator (see *wxFileName::GetPathSeparator* (p. 618)) otherwise an assertion will fail.

The returned path may be relative to the current working directory. Note in fact that *wxPathList* can be used to store both relative and absolute paths so that if you *Add()* (p. 1174)ed relative paths, then the current working directory (see *wxGetCwd* (p. 1923) and *wxSetWorkingDirectory* (p. 1924)) may affect the value returned by this function!

## **wxPen**

A pen is a drawing tool for drawing outlines. It is used for drawing lines and painting the outline of rectangles, ellipses, etc. It has a colour, a width and a style.

### **Derived from**

*wxGDIObject* (p. 709)

*wxObject* (p. 1148)

### Include files

<wx/pen.h>

### Predefined objects

Objects:

**wxNullPen**

Pointers:

**wxRED\_PEN**

**wxCYAN\_PEN**

**wxGREEN\_PEN**

**wxBLACK\_PEN**

**wxWHITE\_PEN**

**wxTRANSPARENT\_PEN**

**wxBLACK\_DASHED\_PEN**

**wxGREY\_PEN**

**wxMEDIUM\_GREY\_PEN**

**wxLIGHT\_GREY\_PEN**

### Remarks

On a monochrome display, *wxWidgets* shows all non-white pens as black.

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

An application may wish to dynamically create pens with different characteristics, and there is the consequent danger that a large number of duplicate pens will be created. Therefore an application may wish to get a pointer to a pen by using the global list of pens **wxThePenList**, and calling the member function **FindOrCreatePen**. See the entry for *wxPenList* (p. 1182).

This class uses *reference counting and copy-on-write* (p. 2046) internally so that assignments between two instances of this class are very cheap. You can therefore use actual objects instead of pointers without efficiency problems. If an instance of this class is changed it will create its own data internally so that other instances, which previously shared the data using the reference counting, are not affected.

### See also

*wxPenList* (p. 1182), *wxDC* (p. 456), *wxDC::SetPen* (p. 475)

### **wxPen::wxPen**

**wxPen()**



Default constructor. The pen will be uninitialised, and `wxPen::IsOk` (p. 1180) will return false.

**wxPen(const wxColour& colour, int width = 1, int style = wxSOLID)**

Constructs a pen from a colour object, pen width and style.

**wxPen(const wxString& colourName, int width, int style)**

Constructs a pen from a colour name, pen width and style.

**wxPen(const wxBitmap& stipple, int width)**

Constructs a stippled pen from a stipple bitmap and a width.

**wxPen(const wxPen& pen)**

Copy constructor, uses *reference counting* (p. 2046).

### Parameters

*colour*

A colour object.

*colourName*

A colour name.

*width*

Pen width. Under Windows, the pen width cannot be greater than 1 if the style is `wxDOT`, `wxLONG_DASH`, `wxSHORT_DASH`, `wxDOT_DASH`, or `wxUSER_DASH`.

*stipple*

A stipple bitmap.

*pen*

A pointer or reference to a pen to copy.

*style*

The style may be one of the following:

<b>wxSOLID</b>	Solid style.
<b>wxTRANSPARENT</b>	No pen is used.
<b>wxDOT</b>	Dotted style.
<b>wxLONG_DASH</b>	Long dashed style.
<b>wxSHORT_DASH</b>	Short dashed style.

<b>wxDOT_DASH</b>	Dot and dash style.
<b>wxSTIPPLE</b>	Use the stipple bitmap.
<b>wxUSER_DASH</b>	Use the user dashes: see <i>wxPen::SetDashes</i> (p. 1180).
<b>wxBDIAGONAL_HATCH</b>	Backward diagonal hatch.
<b>wxCROSSDIAG_HATCH</b>	Cross-diagonal hatch.
<b>wxFDIAGONAL_HATCH</b>	Forward diagonal hatch.
<b>wxCROSS_HATCH</b>	Cross hatch.
<b>wxHORIZONTAL_HATCH</b>	Horizontal hatch.
<b>wxVERTICAL_HATCH</b>	Vertical hatch.

### Remarks

Different versions of Windows and different versions of other platforms support very different subsets of the styles above - there is no similarity even between Windows95 and Windows98 - so handle with care.

If the named colour form is used, an appropriate **wxColour** structure is found in the colour database.

### See also

*wxPen::SetStyle* (p. 1181), *wxPen::SetColour* (p. 1180), *wxPen::SetWidth* (p. 1181), *wxPen::SetStipple* (p. 1181)

**wxPerl note:** Constructors supported by wxPerl are:

- `Pen->new( colour, width, style )`
- `Pen->new( colourName, width, style )`
- `Pen->new( stipple, width )`

### **wxPen::~~wxPen**

**~wxPen()**

Destructor. See *reference-counted object destruction* (p. 2046) for more info.

### Remarks

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

**wxPen::GetCap****int GetCap() const**

Returns the pen cap style, which may be one of **wxCAP\_ROUND**, **wxCAP\_PROJECTING** and **wxCAP\_BUTT**. The default is **wxCAP\_ROUND**.

**See also**

*wxPen::SetCap* (p. 1180)

**wxPen::GetColour****wxColour& GetColour() const**

Returns a reference to the pen colour.

**See also**

*wxPen::SetColour* (p. 1180)

**wxPen::GetDashes****int GetDashes(wxDash\*\* dashes) const**

Gets an array of dashes (defined as char in X, DWORD under Windows). *dashes* is a pointer to the internal array. Do not deallocate or store this pointer. The function returns the number of dashes associated with this pen.

**See also**

*wxPen::SetDashes* (p. 1180)

**wxPen::GetJoin****int GetJoin() const**

Returns the pen join style, which may be one of **wxJOIN\_BEVEL**, **wxJOIN\_ROUND** and **wxJOIN\_MITER**. The default is **wxJOIN\_ROUND**.

**See also**

*wxPen::SetJoin* (p. 1181)

**wxPen::GetStipple****wxBitmap\* GetStipple() const**

Gets a pointer to the stipple bitmap.

**See also**

*wxPen::SetStipple* (p. 1181)

**wxPen::GetStyle****int GetStyle() const**

Returns the pen style.

**See also**

*wxPen::wxPen* (p. 1176), *wxPen::SetStyle* (p. 1181)

**wxPen::GetWidth****int GetWidth() const**

Returns the pen width.

**See also**

*wxPen::SetWidth* (p. 1181)

**wxPen::IsOk****bool IsOk() const**

Returns true if the pen is initialised.

**wxPen::SetCap****void SetCap(int capStyle)**

Sets the pen cap style, which may be one of **wxCAP\_ROUND**, **wxCAP\_PROJECTING** and **wxCAP\_BUTT**. The default is **wxCAP\_ROUND**.

**See also**

*wxPen::GetCap* (p. 1179)

**wxPen::SetColour****void SetColour(wxColour& colour)****void SetColour(const wxString& colourName)****void SetColour(unsigned char red, unsigned char green, unsigned char blue)**

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

**See also**

*wxPen::GetColour* (p. 1179)

**wxPen::SetDashes**

**void SetDashes(int *n*, wxDash\* *dashes*)**

Associates an array of pointers to dashes (defined as char in X, DWORD under Windows) with the pen. The array is not deallocated by wxPen, but neither must it be deallocated by the calling application until the pen is deleted or this function is called with a NULL array.

**See also**

*wxPen::GetDashes* (p. 1179)

**wxPen::SetJoin**

**void SetJoin(int *join\_style*)**

Sets the pen join style, which may be one of **wxJOIN\_BEVEL**, **wxJOIN\_ROUND** and **wxJOIN\_MITER**. The default is **wxJOIN\_ROUND**.

**See also**

*wxPen::GetJoin* (p. 1179)

**wxPen::SetStipple**

**void SetStipple(wxBitmap\* *stipple*)**

Sets the bitmap for stippling.

**See also**

*wxPen::GetStipple* (p. 1179)

**wxPen::SetStyle**

**void SetStyle(int *style*)**

Set the pen style.

**See also**

*wxPen::wxPen* (p. 1176)

**wxPen::SetWidth**

**void SetWidth(int *width*)**

Sets the pen width.

**See also**

*wxPen::GetWidth* (p. 1180)

**wxPen::operator =**

**wxPen& operator =(const wxPen& pen)**

Assignment operator, using *reference counting* (p. 2046).

**wxPen::operator ==**

**bool operator ==(const wxPen& pen)**

Equality operator. See *reference-counted object comparison* (p. 2046) for more info.

**wxPen::operator !=**

**bool operator !=(const wxPen& pen)**

Inequality operator. See *reference-counted object comparison* (p. 2046) for more info.

## wxPenList

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

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

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

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

### See also

*wxPen* (p. 1175)

**wxPenList::wxPenList**

**void wxPenList()**

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

### **wxPenList::FindOrCreatePen**

**wxPen\*** FindOrCreatePen(const wxColour& colour, int width, int style)

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

**wxPen\*** FindOrCreatePen(const wxString& colourName, int width, int style)

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

#### **Parameters**

*colour*

Colour object.

*colourName*

Colour name, which should be in the *colour database* (p. 219).

*width*

Width of pen.

*style*

Pen style. See *wxPen::wxPen* (p. 1176) for a list of styles.

## **wxPickerBase**

Base abstract class for all pickers which support an auxiliary text control. This class handles all positioning and sizing of the text control like a horizontal *wxBoxSizer* (p. 149) would do, with the text control on the left of the picker button. The proportion (see *wxSizer* (p. 1444) documentation for more info about proportion values) of the picker control defaults to 1 when there isn't a text control associated (see *wxPB\_USE\_TEXTCTRL* style) and to 0 otherwise.

#### **Derived from**

*wxControl* (p. 285)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

#### **Include files**

<wx/pickerbase.h>

**Window styles**

**wxPB\_USE\_TEXTCTRL**      Creates a text control to the left of the picker which is completely managed by this *wxPickerBase* (p. 1183) class.

**See also**

*wxColourPickerCtrl* (p. 222)

**wxPickerBase::SetInternalMargin**

**void SetInternalMargin(int margin)**

Sets the margin (in pixel) between the picker and the text control. This function can be used only when *HasTextCtrl* (p. 1185) returns *true*.

**wxPickerBase::GetInternalMargin**

**int GetInternalMargin() const**

Returns the margin (in pixel) between the picker and the text control. This function can be used only when *HasTextCtrl* (p. 1185) returns *true*.

**wxPickerBase::SetTextCtrlProportion**

**void SetTextCtrlProportion(int prop)**

Sets the proportion value of the text control. Look at the overview of *wxPickerBase* for more details about this. This function can be used only when *HasTextCtrl* (p. 1185) returns *true*.

**wxPickerBase::SetPickerCtrlProportion**

**void SetPickerCtrlProportion(int prop)**

Sets the proportion value of the picker. Look at the overview of *wxPickerBase* for more details about this.

**wxPickerBase::GetTextCtrlProportion**

**int GetTextCtrlProportion() const**

Returns the proportion value of the text control. This function can be used only when *HasTextCtrl* (p. 1185) returns *true*.

**wxPickerBase::GetPickerCtrlProportion**

**int GetPickerCtrlProportion() const**



Returns the proportion value of the picker.

### **wxPickerBase::HasTextCtrl**

**bool HasTextCtrl() const**

Returns true if this window has a valid text control (i.e. if the **wxPB\_USE\_TEXTCTRL** style was given when creating this control).

### **wxPickerBase::GetTextCtrl**

**wxTextCtrl \* GetTextCtrl()**

Returns a pointer to the text control handled by this window or `NULL` if the **wxPB\_USE\_TEXTCTRL** style was not specified when this control was created. Very important: the contents of the text control could be containing an invalid representation of the entity which can be chosen through the picker (e.g. the user entered an invalid colour syntax because of a typo). Thus you should never parse the content of the `textctrl` to get the user's input; rather use the derived-class getter (e.g. `wxColourPickerCtrl::GetColour` (p. 224), `wxFilePickerCtrl::GetPath` (p. 632), etc).

### **wxPickerBase::IsTextCtrlGrowable**

**bool IsTextCtrlGrowable() const**

Returns `true` if the text control is growable. This function can be used only when *HasTextCtrl* (p. 1185) returns `true`.

### **wxPickerBase::SetPickerCtrlGrowable**

**void SetPickerCtrlGrowable(bool grow = true)**

Sets the picker control as growable when `grow` is `true`.

### **wxPickerBase::SetTextCtrlGrowable**

**void SetTextCtrlGrowable(bool grow = true)**

Sets the text control as growable when `grow` is `true`. This function can be used only when *HasTextCtrl* (p. 1185) returns `true`.

### **wxPickerBase::IsPickerCtrlGrowable**

**bool IsPickerCtrlGrowable() const**

Returns `true` if the picker control is growable.

## **wxPlatformInfo**

This class holds informations about the operating system and the toolkit that the application is running under and some basic architecture info of the machine where it's running.

### Derived from

*wxObject* (p. 1148)

### Include files

<wx/platinfo.h>

### See also

*wxGetOSVersion* (p. 1927), *wxIsPlatformLittleEndian* (p. 1928), *wxIsPlatform64Bit* (p. 1928), *wxAppTraits* (p. 56)

### Data structures

The following are the operating systems which are recognized by *wxWidgets* and whose version can be detected at run-time. The values of the constants are chosen so that they can be combined as flags; this allows to check for operating system *families* like e.g. *wxOS\_MAC* and *wxOS\_UNIX*.

```
enum wxOperatingSystemId
{
    wxOS_UNKNOWN = 0,                // returned on error

    wxOS_MAC_OS      = 1 << 0,      // Apple Mac OS 8/9/X with Mac
    paths
    wxOS_MAC_OSX_DARWIN = 1 << 1,    // Apple Mac OS X with Unix paths
    wxOS_MAC = wxOS_MAC_OS | wxOS_MAC_OSX_DARWIN,

    wxOS_WINDOWS_9X   = 1 << 2,      // Windows 9x family (95/98/ME)
    wxOS_WINDOWS_NT   = 1 << 3,      // Windows NT family (NT/2000/XP)
    wxOS_WINDOWS_MICRO = 1 << 4,      // MicroWindows
    wxOS_WINDOWS_CE    = 1 << 5,      // Windows CE (Window Mobile)
    wxOS_WINDOWS = wxOS_WINDOWS_9X   |
                  wxOS_WINDOWS_NT   |
                  wxOS_WINDOWS_MICRO |
                  wxOS_WINDOWS_CE,

    wxOS_UNIX_LINUX    = 1 << 6,      // Linux
    wxOS_UNIX_FREEBSD  = 1 << 7,      // FreeBSD
    wxOS_UNIX_OPENBSD  = 1 << 8,      // OpenBSD
    wxOS_UNIX_NETBSD   = 1 << 9,      // NetBSD
    wxOS_UNIX_SOLARIS  = 1 << 10,     // SunOS
    wxOS_UNIX_AIX       = 1 << 11,     // AIX
    wxOS_UNIX_HPUX      = 1 << 12,     // HP/UX
    wxOS_UNIX = wxOS_UNIX_LINUX      |
                wxOS_UNIX_FREEBSD   |
                wxOS_UNIX_OPENBSD    |
                wxOS_UNIX_NETBSD     |
                wxOS_UNIX_SOLARIS    |
                wxOS_UNIX_AIX         |
                wxOS_UNIX_HPUX,
```

```
    wxOS_DOS          = 1 << 15,      // Microsoft DOS
    wxOS_OS2          = 1 << 16      // OS/2
};
```

The list of wxWidgets ports. Some of them can be used with more than a single (native) toolkit; e.g. wxWinCE port sources can be used with smartphones, pocket PCs and handheld devices SDKs.

```
enum wxPortId
{
    wxPORT_UNKNOWN    = 0,            // returned on error

    wxPORT_BASE       = 1 << 0,      // wxBase, no native toolkit used

    wxPORT_MSW        = 1 << 1,      // wxMSW, native toolkit is Windows
API
    wxPORT_MOTIF      = 1 << 2,      // wxMotif, using [Open]Motif or
Lesstif
    wxPORT_GTK        = 1 << 3,      // wxGTK, using GTK+ 1.x, 2.x, GPE
or Maemo
    wxPORT_MGL        = 1 << 4,      // wxMGL, using wxUniversal
    wxPORT_X11        = 1 << 5,      // wxX11, using wxUniversal
    wxPORT_OS2        = 1 << 6,      // wxOS2, using OS/2 Presentation
Manager
    wxPORT_MAC        = 1 << 7,      // wxMac, using Carbon or Classic
Mac API
    wxPORT_COCOA      = 1 << 8,      // wxCocoa, using Cocoa NextStep/Mac
API
    wxPORT_WINCE      = 1 << 9,      // wxWinCE, toolkit is WinCE SDK
API
    wxPORT_PALMOS     = 1 << 10,     // wxPalmOS, toolkit is PalmOS API
    wxPORT_DFB        = 1 << 11     // wxDFB, using wxUniversal
};
```

The architecture of the operating system (regardless of the build environment of wxWidgets library - see *wx/sPlatform64bit* (p. 1928) documentation for more info).

```
enum wxArchitecture
{
    wxARCH_INVALID    = -1,          // returned on error

    wxARCH_32,
    wxARCH_64,

    wxARCH_MAX

}
```

The endian-ness of the machine.

```
enum wxEndianness
{
    wxENDIAN_INVALID  = -1,          // returned on error

    wxENDIAN_BIG,
    wxENDIAN_LITTLE,
    // 4321
    // 1234
}
```

```
        wxENDIAN_PDP,                // 3412
    wxENDIAN_MAX
}
```

## **wxPlatformInfo::wxPlatformInfo**

### **wxPlatformInfo()**

Initializes the instance with the values corresponding to the currently running platform. This is a fast operation because it only requires to copy the values internally cached for the currently running platform (see also *Get* (p. 1188)).

**wxPlatformInfo(wxPortId pid = wxPORT\_UNKNOWN, int tkMajor = -1, int tkMinor = -1, wxOperatingSystemId id = wxOS\_UNKNOWN, int osMajor = -1, int osMinor = -1, wxArchitecture arch = wxARCH\_INVALID, wxEndianness endian = wxENDIAN\_INVALID)**

Initializes the object using given values.

## **wxPlatformInfo::CheckOSVersion**

### **bool CheckOSVersion(int major, int minor) const**

Returns `true` if the OS version is at least `major.minor`.

#### **See also**

*GetOSMajorVersion* (p. 1189), *GetOSMinorVersion* (p. 1190), *CheckToolkitVersion* (p. 1188)

## **wxPlatformInfo::CheckToolkitVersion**

### **bool CheckToolkitVersion(int major, int minor) const**

Returns `true` if the toolkit version is at least `major.minor`.

#### **See also**

*GetToolkitMajorVersion* (p. 1191), *GetToolkitMinorVersion* (p. 1191), *CheckOSVersion* (p. 1188)

## **wxPlatformInfo::Get**

### **static const wxPlatformInfo& Get()**

Returns the global `wxPlatformInfo` object, initialized with the values for the currently running platform.

## **wxPlatformInfo::GetArch**

**static wxArchitecture GetArch(const wxString& arch)**

Converts the given string to a wxArchitecture enum value or to wxARCH\_INVALID if the given string is not a valid architecture string (i.e. does not contain nor 32 nor 64 strings).

**wxPlatformInfo::GetArchName**

**static wxString GetArchName(wxArchitecture arch)**

Returns the name for the given wxArchitecture enumeration value.

**wxString GetArchName() const**

Returns the name for the architecture of this wxPlatformInfo instance.

**wxPlatformInfo::GetArchitecture**

**wxArchitecture GetArchitecture() const**

Returns the architecture ID of this wxPlatformInfo instance.

**wxPlatformInfo::GetEndianness**

**static wxEndianness GetEndianness(const wxString& end)**

Converts the given string to a wxEndianness enum value or to wxENDIAN\_INVALID if the given string is not a valid endianness string (i.e. does not contain nor little nor big strings).

**wxEndianness GetEndianness() const**

Returns the endianness ID of this wxPlatformInfo instance.

**wxPlatformInfo::GetEndiannessName**

**static wxString GetEndiannessName(wxEndianness end)**

Returns name for the given wxEndianness enumeration value.

**wxString GetEndiannessName() const**

Returns the name for the endianness of this wxPlatformInfo instance.

**wxPlatformInfo::GetOSMajorVersion**

**int GetOSMajorVersion() const**

Returns the run-time major version of the OS associated with this wxPlatformInfo instance. See *wxGetOsVersion* (p. 1927) for more info.

**See also**

*CheckOSVersion* (p. 1188)

### **wxPlatformInfo::GetOSMinorVersion**

**int GetOSMinorVersion() const**

Returns the run-time minor version of the OS associated with this *wxPlatformInfo* instance. See *wxGetOsVersion* (p. 1927) for more info.

#### **See also**

*CheckOSVersion* (p. 1188)

### **wxPlatformInfo::GetOperatingSystemFamilyName**

**static wxString GetOperatingSystemFamilyName(wxOperatingSystemId os)**

Returns the operating system family name for the given *wxOperatingSystemId* enumeration value: *Unix* for *wxOS\_UNIX*, *Macintosh* for *wxOS\_MAC*, *Windows* for *wxOS\_WINDOWS*, *DOS* for *wxOS\_DOS*, *OS/2* for *wxOS\_OS2*.

**wxString GetOperatingSystemFamilyName() const**

Returns the operating system family name of the OS associated with this *wxPlatformInfo* instance.

### **wxPlatformInfo::GetOperatingSystemId**

**static wxOperatingSystemId GetOperatingSystemId(const wxString& name)**

Converts the given string to a *wxOperatingSystemId* enum value or to *wxOS\_UNKNOWN* if the given string is not a valid operating system name.

**wxOperatingSystemId GetOperatingSystemId() const**

Returns the operating system ID of this *wxPlatformInfo* instance.

### **wxPlatformInfo::GetOperatingSystemIdName**

**static wxString GetOperatingSystemIdName(wxOperatingSystemId os)**

Returns the name for the given operating system ID value. This can be a long name (e.g. *Microsoft Windows NT*); use *GetOperatingSystemFamilyName* (p. 1190) to retrieve a short, generic name.

**wxString GetOperatingSystemIdName() const**

Returns the operating system name of the OS associated with this *wxPlatformInfo* instance.

### **wxPlatformInfo::GetPortId**

**static wxPortId GetPortId(const wxString& portname)**

Converts the given string to a wxWidgets port ID value or to wxPORT\_UNKNOWN if the given string does not match any of the wxWidgets canonical name ports ("wxGTK", "wxMSW", etc) nor any of the short wxWidgets name ports ("gtk", "msw", etc).

**wxPortId GetPortId() const**

Returns the wxWidgets port ID associated with this wxPlatformInfo instance.

**wxPlatformInfo::GetPortIdName**

**static wxString GetPortIdName(wxPortId port, bool usingUniversal)**

Returns the name of the given wxWidgets port ID value. The *usingUniversal* argument specifies whether the port is in its native or wxUniversal variant.

The returned string always starts with the "wx" prefix and is a mixed-case string.

**wxString GetPortIdName() const**

Returns the name of the wxWidgets port ID associated with this wxPlatformInfo instance.

**wxPlatformInfo::GetPortIdShortName**

**static wxString GetPortIdShortName(wxPortId port, bool usingUniversal)**

Returns the short name of the given wxWidgets port ID value. The *usingUniversal* argument specifies whether the port is in its native or wxUniversal variant.

The returned string does not start with the "wx" prefix and is always lower case.

**wxString GetPortIdShortName() const**

Returns the short name of the wxWidgets port ID associated with this wxPlatformInfo instance.

**wxPlatformInfo::GetToolkitMajorVersion**

**int GetToolkitMajorVersion() const**

Returns the run-time major version of the toolkit associated with this wxPlatformInfo instance. Note that if *GetPortId* (p. 1190) returns wxPORT\_BASE, then this value is zero (unless externally modified with *SetToolkitVersion* (p. 1193)); that is, no native toolkit is in use.

See *wxAppTraits::GetToolkitVersion* (p. 57) for more info.

**See also**

*CheckToolkitVersion* (p. 1188)

**wxPlatformInfo::GetToolkitMinorVersion**

**int GetToolkitMinorVersion() const**

Returns the run-time minor version of the toolkit associated with this `wxPlatformInfo` instance. Note that if *GetPortId* (p. 1190) returns `wxPORT_BASE`, then this value is zero (unless externally modified with *SetToolkitVersion* (p. 1193)); that is, no native toolkit is in use.

See *wxAppTraits::GetToolkitVersion* (p. 57) for more info.

**See also**

*CheckToolkitVersion* (p. 1188)

**wxPlatformInfo::IsOk****bool IsOk() const**

Returns `true` if this instance is fully initialized with valid values.

**wxPlatformInfo::IsUsingUniversalWidgets****bool IsUsingUniversalWidgets() const**

Returns `true` if this `wxPlatformInfo` describes `wxUniversal` build.

**wxPlatformInfo::SetArchitecture****void SetArchitecture(wxArchitecture *n*)**

Sets the architecture enum value associated with this `wxPlatformInfo` instance.

**wxPlatformInfo::SetEndianness****void SetEndianness(wxEndianness *n*)**

Sets the endianness enum value associated with this `wxPlatformInfo` instance.

**wxPlatformInfo::SetOSVersion****void SetOSVersion(int *major*, int *minor*)**

Sets the version of the operating system associated with this `wxPlatformInfo` instance.

**wxPlatformInfo::SetOperatingSystemId****void SetOperatingSystemId(wxOperatingSystemId *n*)**

Sets the operating system associated with this `wxPlatformInfo` instance.

**wxPlatformInfo::SetPortId**



**void SetPortId(wxPortId n)**

Sets the wxWidgets port ID associated with this wxPlatformInfo instance.

**wxPlatformInfo::SetToolkitVersion**

**void SetToolkitVersion(int major, int minor)**

Sets the version of the toolkit associated with this wxPlatformInfo instance.

**wxPlatformInfo::operator!=**

**bool operator!=(const wxPlatformInfo& t) const**

Inequality operator. Tests all class' internal variables.

**wxPlatformInfo::operator==**

**bool operator==(const wxPlatformInfo& t) const**

Equality operator. Tests all class' internal variables.

## **wxPoint**

A **wxPoint** is a useful data structure for graphics operations. It simply contains integer *x* and *y* members.

See also *wxRealPoint* (p. 1251) for a floating point version.

### **Derived from**

None

### **Include files**

<wx/gdicmn.h>

### **See also**

*wxRealPoint* (p. 1251)

**wxPoint::wxPoint**

**wxPoint()**

**wxPoint(int x, int y)**

Create a point.

**wxPoint::x****int x**

x member.

**wxPoint::y****int y**

y member.

**Operators****void operator =(const wxPoint& pt)**

Assignment operator.

**bool operator ==(const wxPoint& pt)****bool operator !=(const wxPoint& pt)****wxPoint operator +(const wxPoint& pt)****wxPoint operator -(const wxPoint& pt)****wxPoint& operator +=(const wxPoint& pt)****wxPoint& operator -=(const wxPoint& pt)**Operators for comparison, sum and subtraction between *wxPoint* (p. 1193) objects.**wxPoint operator +(const wxSize& sz)****wxPoint operator -(const wxSize& sz)****wxPoint& operator +=(const wxSize& sz)****wxPoint& operator -=(const wxSize& sz)** Operators for sum and subtraction between a *wxPoint* (p. 1193) object and a *wxSize* (p. 1440) object.**wxPostScriptDC**

This defines the wxWidgets Encapsulated PostScript device context, which can write PostScript files on any platform. See *wxDC* (p. 456) for descriptions of the member functions.

**Derived from***wxDC* (p. 456)*wxObject* (p. 1148)**Include files**

<wx/dcps.h>

### **wxPostScriptDC::wxPostScriptDC**

**wxPostScriptDC(const wxPrintData& printData)**

Constructs a PostScript printer device context from a *wxPrintData* (p. 1200) object.

**wxPostScriptDC(const wxString& output, bool interactive = true,  
wxWindow \*parent)**

Constructor. *output* is an optional file for printing to, and if *interactive* is true a dialog box will be displayed for adjusting various parameters. *parent* is the parent of the printer dialog box.

Use the *Ok* member to test whether the constructor was successful in creating a usable device context.

See *Printer settings* (p. 1947) for functions to set and get PostScript printing settings.

This constructor and the global printer settings are now deprecated; use the *wxPrintData* constructor instead.

### **wxPostScriptDC::SetResolution**

**static void SetResolution(int ppi)**

Set resolution (in pixels per inch) that will be used in PostScript output. Default is 720ppi.

### **wxPostScriptDC::GetResolution**

**static int GetResolution()**

Return resolution used in PostScript output. See *SetResolution* (p. 1195).

## **wxPowerEvent**

The power events are generated when the system power state changes, e.g. the system is suspended, hibernated, plugged into or unplugged from the wall socket and so on.

Notice that currently only suspend and resume events are generated and only under MS Windows platform. To avoid the need to change the code using this event later when these events are implemented on the other platforms please use the test `ifdef wxHAS_POWER_EVENTS` instead of directly testing for the platform in your code: this symbol will be defined for all platforms supporting the power events.

### **Event table macros**

To process power events, use these macros to handle them in member functions that take

a `wxPowerEvent` argument.

**EVT\_POWER\_SUSPENDING(func)**

System is about to be suspended, this event can be vetoed to prevent suspend from taking place.

**EVT\_POWER\_SUSPENDED(func)**

System is about to suspend: normally the application should quickly (i.e. without user intervention) close all the open files and network connections here, possibly remembering them to reopen them later when the system is resumed.

**EVT\_POWER\_SUSPEND\_CANCEL(func)**

System suspension was cancelled because some application vetoed it.

**EVT\_POWER\_RESUME(func)**

System resumed from suspend: normally the application should restore the state in which it had been before the suspension.

**Derived from**

*wxEvent* (p. 572)

*wxObject* (p. 1148)

**Include files**

<wx/power.h>

**See also**

*wxGetPowerType* (p. 1956), *wxGetBatteryState* (p. 1956)

**wxPowerEvent::Veto****void Veto()**

Call this to prevent suspend from taking place in `wxEVT_POWER_SUSPENDING` handler (it is ignored for all the others).

**wxPreviewCanvas**

A preview canvas is the default canvas used by the print preview system to display the preview.

**Derived from**

*wxScrolledWindow* (p. 1414)

*wxWindow* (p. 1795)  
*wxevthandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/print.h>

**See also**

*wxPreviewFrame* (p. 1199), *wxPreviewControlBar* (p. 1197), *wxPrintPreview* (p. 1221)

**wxPreviewCanvas::wxPreviewCanvas**

**wxPreviewCanvas**(*wxPrintPreview\** preview, *wxWindow\** parent, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = 0, **const wxString&** name = "canvas")

Constructor.

**wxPreviewCanvas::~~wxPreviewCanvas**

**~wxPreviewCanvas**()

Destructor.

**wxPreviewCanvas::OnPaint**

**void OnPaint**(*wxPaintEvent&* event)

Calls *wxPrintPreview::PaintPage* (p. 1223) to refresh the canvas.

**wxPreviewControlBar**

This is the default implementation of the preview control bar, a panel with buttons and a zoom control. You can derive a new class from this and override some or all member functions to change the behaviour and appearance; or you can leave it as it is.

**Derived from**

*wxPanel* (p. 1170)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/print.h>

**See also**

*wxPreviewFrame* (p. 1199), *wxPreviewCanvas* (p. 1196), *wxPrintPreview* (p. 1221)

### **wxPreviewControlBar::wxPreviewControlbar**

**wxPreviewControlBar**(*wxPrintPreview\** preview, **long** buttons, **wxWindow\*** parent, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = 0, **const wxString&** name = "panel")

Constructor.

The buttons parameter may be a combination of the following, using the bitwise 'or' operator.

<code>wxPREVIEW_PRINT</code>	Create a print button.
<code>wxPREVIEW_NEXT</code>	Create a next page button.
<code>wxPREVIEW_PREVIOUS</code>	Create a previous page button.
<code>wxPREVIEW_ZOOM</code>	Create a zoom control.
<code>wxPREVIEW_DEFAULT</code>	Equivalent to a combination of <code>wxPREVIEW_PREVIOUS</code> , <code>wxPREVIEW_NEXT</code> and <code>wxPREVIEW_ZOOM</code> .

### **wxPreviewControlBar::~~wxPreviewControlBar**

**~wxPreviewControlBar**()

Destructor.

### **wxPreviewControlBar::CreateButtons**

**void CreateButtons**()

Creates buttons, according to value of the button style flags.

### **wxPreviewControlBar::GetPrintPreview**

**wxPrintPreview \* GetPrintPreview**()

Gets the print preview object associated with the control bar.

### **wxPreviewControlBar::GetZoomControl**

**int GetZoomControl**()

Gets the current zoom setting in percent.

**wxPreviewControlBar::SetZoomControl****void SetZoomControl**(int *percent*)

Sets the zoom control.

**wxPreviewFrame**

This class provides the default method of managing the print preview interface. Member functions may be overridden to replace functionality, or the class may be used without derivation.

**Derived from**

*wxFrame* (p. 682)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/print.h>

**See also**

*wxPreviewCanvas* (p. 1196), *wxPreviewControlBar* (p. 1197), *wxPrintPreview* (p. 1221)

**wxPreviewFrame::wxPreviewFrame**

**wxPreviewFrame**(*wxPrintPreview\** *preview*, *wxWindow\** *parent*, **const wxString& title**, **const wxPoint& pos** = *wxDefaultPosition*, **const wxSize& size** = *wxDefaultSize*, **long style** = *wxDEFAULT\_FRAME\_STYLE*, **const wxString& name** = "frame")

Constructor. Pass a print preview object plus other normal frame arguments. The print preview object will be destroyed by the frame when it closes.

**wxPreviewFrame::~~wxPreviewFrame**

**~wxPreviewFrame**()

Destructor.

**wxPreviewFrame::CreateControlBar**

**void CreateControlBar**()

Creates a *wxPreviewControlBar*. Override this function to allow a user-defined preview control bar object to be created.

**wxPreviewFrame::CreateCanvas****void CreateCanvas()**

Creates a wxPreviewCanvas. Override this function to allow a user-defined preview canvas object to be created.

**wxPreviewFrame::Initialize****void Initialize()**

Creates the preview canvas and control bar, and calls wxWindow::MakeModal(true) to disable other top-level windows in the application.

This function should be called by the application prior to showing the frame.

**wxPreviewFrame::OnCloseWindow****void OnCloseWindow(wxCloseEvent& event)**

Enables the other frames in the application, and deletes the print preview object, implicitly deleting any printout objects associated with the print preview object.

**wxPrintData**

This class holds a variety of information related to printers and printer device contexts. This class is used to create a wxPrinterDC and a wxPostScriptDC. It is also used as a data member of wxPrintDialogData and wxPageSetupDialogData, as part of the mechanism for transferring data between the print dialogs and the application.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/cmndata.h>

**See also**

*Printing framework overview* (p. 2145), *wxPrintDialog* (p. 1206), *wxPageSetupDialog* (p. 1158), *wxPrintDialogData* (p. 1207), *wxPageSetupDialogData* (p. 1159), *wxPrintDialog Overview* (p. 2130), *wxPrinterDC* (p. 1213), *wxPostScriptDC* (p. 1194)

**Remarks**

The following functions are specific to PostScript printing and have not yet been documented:

```
const wxString& GetPrinterCommand() const ;
const wxString& GetPrinterOptions() const ;
const wxString& GetPreviewCommand() const ;
```



```
const wxString& GetFilename() const ;
const wxString& GetFontMetricPath() const ;
double GetPrinterScaleX() const ;
double GetPrinterScaleY() const ;
long GetPrinterTranslateX() const ;
long GetPrinterTranslateY() const ;
// wxPRINT_MODE_PREVIEW, wxPRINT_MODE_FILE, wxPRINT_MODE_PRINTER
wxPrintMode GetPrintMode() const ;

void SetPrinterCommand(const wxString& command) ;
void SetPrinterOptions(const wxString& options) ;
void SetPreviewCommand(const wxString& command) ;
void SetFilename(const wxString& filename) ;
void SetFontMetricPath(const wxString& path) ;
void SetPrinterScaleX(double x) ;
void SetPrinterScaleY(double y) ;
void SetPrinterScaling(double x, double y) ;
void SetPrinterTranslateX(long x) ;
void SetPrinterTranslateY(long y) ;
void SetPrinterTranslation(long x, long y) ;
void SetPrintMode(wxPrintMode printMode) ;
```

## **wxPrintData::wxPrintData**

### **wxPrintData()**

Default constructor.

### **wxPrintData(const wxPrintData& data)**

Copy constructor.

## **wxPrintData::~~wxPrintData**

### **~wxPrintData()**

Destructor.

## **wxPrintData::GetCollate**

### **bool GetCollate() const**

Returns true if collation is on.

## **wxPrintData::GetBin**

### **wxPrintBin GetBin() const**

Returns the current bin (papersource). By default, the system is left to select the bin (wxPRINTBIN\_DEFAULT is returned).

See *SetBin()* (p. 1203) for the full list of bin values.

### **wxPrintData::GetColour**

**bool GetColour() const**

Returns true if colour printing is on.

### **wxPrintData::GetDuplex**

**wxDuplexMode GetDuplex() const**

Returns the duplex mode. One of `wxDUPLEX_SIMPLEX`, `wxDUPLEX_HORIZONTAL`, `wxDUPLEX_VERTICAL`.

### **wxPrintData::GetNoCopies**

**int GetNoCopies() const**

Returns the number of copies requested by the user.

### **wxPrintData::GetOrientation**

**int GetOrientation() const**

Gets the orientation. This can be `wxLANDSCAPE` or `wxPORTRAIT`.

### **wxPrintData::GetPaperId**

**wxPaperSize GetPaperId() const**

Returns the paper size id. For more information, see *wxPrintData::SetPaperId* (p. 1204).

### **wxPrintData::GetPrinterName**

**const wxString& GetPrinterName() const**

Returns the printer name. If the printer name is the empty string, it indicates that the default printer should be used.

### **wxPrintData::GetQuality**

**wxPrintQuality GetQuality() const**

Returns the current print quality. This can be a positive integer, denoting the number of dots per inch, or one of the following identifiers:

```
wxPRINT_QUALITY_HIGH  
wxPRINT_QUALITY_MEDIUM  
wxPRINT_QUALITY_LOW  
wxPRINT_QUALITY_DRAFT
```

On input you should pass one of these identifiers, but on return you may get back a positive integer indicating the current resolution setting.

### **wxPrintData::IsOk**

#### **bool IsOk() const**

Returns true if the print data is valid for using in print dialogs. This can return false on Windows if the current printer is not set, for example. On all other platforms, it returns true.

### **wxPrintData::SetBin**

#### **void SetBin(wxPrintBin *flag*)**

Sets the current bin. Possible values are:

```
enum wxPrintBin
{
    wxPRINTBIN_DEFAULT,

    wxPRINTBIN_ONLYONE,
    wxPRINTBIN_LOWER,
    wxPRINTBIN_MIDDLE,
    wxPRINTBIN_MANUAL,
    wxPRINTBIN_ENVELOPE,
    wxPRINTBIN_ENVMANUAL,
    wxPRINTBIN_AUTO,
    wxPRINTBIN_TRACTOR,
    wxPRINTBIN_SMALLFMT,
    wxPRINTBIN_LARGEfmt,
    wxPRINTBIN_LARGECAPACITY,
    wxPRINTBIN_CASSETTE,
    wxPRINTBIN_FORMSOURCE,

    wxPRINTBIN_USER,
};
```

### **wxPrintData::SetCollate**

#### **void SetCollate(bool *flag*)**

Sets collation to on or off.

### **wxPrintData::SetColour**

#### **void SetColour(bool *flag*)**

Sets colour printing on or off.

**wxPrintData::SetDuplex****void SetDuplex(wxDuplexMode mode)**

Returns the duplex mode. One of wxDUPLEX\_SIMPLEX, wxDUPLEX\_HORIZONTAL, wxDUPLEX\_VERTICAL.

**wxPrintData::SetNoCopies****void SetNoCopies(int n)**

Sets the default number of copies to be printed out.

**wxPrintData::SetOrientation****void SetOrientation(int orientation)**

Sets the orientation. This can be wxLANDSCAPE or wxPORTRAIT.

**wxPrintData::SetPaperId****void SetPaperId(wxPaperSize paperId)**

Sets the paper id. This indicates the type of paper to be used. For a mapping between paper id, paper size and string name, see wxPrintPaperDatabase in `paper.h` (not yet documented).

*paperId* can be one of:

wxPAPER_NONE,	// Use specific dimensions
wxPAPER_LETTER,	// Letter, 8 1/2 by 11 inches
wxPAPER_LEGAL,	// Legal, 8 1/2 by 14 inches
wxPAPER_A4,	// A4 Sheet, 210 by 297 millimeters
wxPAPER_CSHEET,	// C Sheet, 17 by 22 inches
wxPAPER_DSHEET,	// D Sheet, 22 by 34 inches
wxPAPER_ESHEET,	// E Sheet, 34 by 44 inches
wxPAPER_LETTERSMALL,	// Letter Small, 8 1/2 by 11 inches
wxPAPER_TABLOID,	// Tabloid, 11 by 17 inches
wxPAPER_LEDGER,	// Ledger, 17 by 11 inches
wxPAPER_STATEMENT,	// Statement, 5 1/2 by 8 1/2 inches
wxPAPER_EXECUTIVE,	// Executive, 7 1/4 by 10 1/2 inches
wxPAPER_A3,	// A3 sheet, 297 by 420 millimeters
wxPAPER_A4SMALL,	// A4 small sheet, 210 by 297 millimeters
wxPAPER_A5,	// A5 sheet, 148 by 210 millimeters
wxPAPER_B4,	// B4 sheet, 250 by 354 millimeters
wxPAPER_B5,	// B5 sheet, 182-by-257-millimeter paper
wxPAPER_FOLIO,	// Folio, 8-1/2-by-13-inch paper
wxPAPER_QUARTO,	// Quarto, 215-by-275-millimeter paper
wxPAPER_10X14,	// 10-by-14-inch sheet
wxPAPER_11X17,	// 11-by-17-inch sheet
wxPAPER_NOTE,	// Note, 8 1/2 by 11 inches
wxPAPER_ENV_9,	// #9 Envelope, 3 7/8 by 8 7/8 inches
wxPAPER_ENV_10,	// #10 Envelope, 4 1/8 by 9 1/2 inches
wxPAPER_ENV_11,	// #11 Envelope, 4 1/2 by 10 3/8 inches

---

```

wxPAPER_ENV_12,           // #12 Envelope, 4 3/4 by 11 inches
wxPAPER_ENV_14,           // #14 Envelope, 5 by 11 1/2 inches
wxPAPER_ENV_DL,           // DL Envelope, 110 by 220 millimeters
wxPAPER_ENV_C5,           // C5 Envelope, 162 by 229 millimeters
wxPAPER_ENV_C3,           // C3 Envelope, 324 by 458 millimeters
wxPAPER_ENV_C4,           // C4 Envelope, 229 by 324 millimeters
wxPAPER_ENV_C6,           // C6 Envelope, 114 by 162 millimeters
wxPAPER_ENV_C65,          // C65 Envelope, 114 by 229 millimeters
wxPAPER_ENV_B4,           // B4 Envelope, 250 by 353 millimeters
wxPAPER_ENV_B5,           // B5 Envelope, 176 by 250 millimeters
wxPAPER_ENV_B6,           // B6 Envelope, 176 by 125 millimeters
wxPAPER_ENV_ITALY,        // Italy Envelope, 110 by 230 millimeters
wxPAPER_ENV_MONARCH,      // Monarch Envelope, 3 7/8 by 7 1/2
inches
wxPAPER_ENV_PERSONAL,     // 6 3/4 Envelope, 3 5/8 by 6 1/2 inches
wxPAPER_FANFOLD_US,       // US Std Fanfold, 14 7/8 by 11 inches
wxPAPER_FANFOLD_STD_GERMAN, // German Std Fanfold, 8 1/2 by 12
inches
wxPAPER_FANFOLD_LGL_GERMAN, // German Legal Fanfold, 8 1/2 by 13
inches

Windows 95 only:
wxPAPER_ISO_B4,           // B4 (ISO) 250 x 353 mm
wxPAPER_JAPANESE_POSTCARD, // Japanese Postcard 100 x 148 mm
wxPAPER_9X11,             // 9 x 11 in
wxPAPER_10X11,            // 10 x 11 in
wxPAPER_15X11,            // 15 x 11 in
wxPAPER_ENV_INVITE,        // Envelope Invite 220 x 220 mm
wxPAPER_LETTER_EXTRA,      // Letter Extra 9 \275 x 12 in
wxPAPER_LEGAL_EXTRA,       // Legal Extra 9 \275 x 15 in
wxPAPER_TABLOID_EXTRA,     // Tabloid Extra 11.69 x 18 in
wxPAPER_A4_EXTRA,          // A4 Extra 9.27 x 12.69 in
wxPAPER_LETTER_TRANSVERSE, // Letter Transverse 8 \275 x 11 in
wxPAPER_A4_TRANSVERSE,     // A4 Transverse 210 x 297 mm
wxPAPER_LETTER_EXTRA_TRANSVERSE, // Letter Extra Transverse
9\275 x 12 in
wxPAPER_A_PLUS,           // SuperA/SuperA/A4 227 x 356 mm
wxPAPER_B_PLUS,           // SuperB/SuperB/A3 305 x 487 mm
wxPAPER_LETTER_PLUS,       // Letter Plus 8.5 x 12.69 in
wxPAPER_A4_PLUS,           // A4 Plus 210 x 330 mm
wxPAPER_A5_TRANSVERSE,     // A5 Transverse 148 x 210 mm
wxPAPER_B5_TRANSVERSE,     // B5 (JIS) Transverse 182 x 257 mm
wxPAPER_A3_EXTRA,          // A3 Extra 322 x 445 mm
wxPAPER_A5_EXTRA,          // A5 Extra 174 x 235 mm
wxPAPER_B5_EXTRA,          // B5 (ISO) Extra 201 x 276 mm
wxPAPER_A2,                // A2 420 x 594 mm
wxPAPER_A3_TRANSVERSE,     // A3 Transverse 297 x 420 mm
wxPAPER_A3_EXTRA_TRANSVERSE // A3 Extra Transverse 322 x 445 mm

```

### **wxPrintData::SetPrinterName**

**void SetPrinterName(const wxString& printerName)**

Sets the printer name. This can be the empty string to indicate that the default printer

should be used.

### **wxPrintData::SetQuality**

**void SetQuality(wxPrintQuality *quality*)**

Sets the desired print quality. This can be a positive integer, denoting the number of dots per inch, or one of the following identifiers:

```
wxPRINT_QUALITY_HIGH  
wxPRINT_QUALITY_MEDIUM  
wxPRINT_QUALITY_LOW  
wxPRINT_QUALITY_DRAFT
```

On input you should pass one of these identifiers, but on return you may get back a positive integer indicating the current resolution setting.

### **wxPrintData::operator =**

**void operator =(const wxPrintData& *data*)**

Assigns print data to this object.

**void operator =(const wxPrintSetupData& *data*)**

Assigns print setup data to this object. wxPrintSetupData is deprecated, but retained for backward compatibility.

## **wxPrintDialog**

This class represents the print and print setup common dialogs. You may obtain a *wxPrinterDC* (p. 1213) device context from a successfully dismissed print dialog.

### **Derived from**

*wxDialog* (p. 496)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### **Include files**

<wx/printdlg.h>

### **See also**

*Printing framework overview* (p. 2145), *wxPrintDialog Overview* (p. 2130)

### **wxPrintDialog::wxPrintDialog**

**wxPrintDialog(wxWindow\* parent, wxPrintDialogData\* data = NULL)**

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

**See also**

*wxPrintDialogData* (p. 1207)

**wxPrintDialog::~~wxPrintDialog**

**~wxPrintDialog()**

Destructor. If `wxPrintDialog::GetPrintDC` has *not* been called, the device context obtained by the dialog (if any) will be deleted.

**wxPrintDialog::GetPrintDialogData**

**wxPrintDialogData& GetPrintDialogData()**

Returns the *print dialog data* (p. 1207) associated with the print dialog.

**wxPrintDialog::GetPrintDC**

**wxDC\* GetPrintDC()**

Returns the device context created by the print dialog, if any. When this function has been called, the ownership of the device context is transferred to the application, so it must then be deleted explicitly.

**wxPrintDialog::ShowModal**

**int ShowModal()**

Shows the dialog, returning `wxID_OK` if the user pressed OK, and `wxID_CANCEL` otherwise. After this function is called, a device context may be retrievable using *wxPrintDialog::GetPrintDC* (p. 1207).

**wxPrintDialogData**

This class holds information related to the visual characteristics of `wxPrintDialog`. It contains a `wxPrintData` object with underlying printing settings.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/cmndata.h>

**See also**

*Printing framework overview* (p. 2145), *wxPrintDialog* (p. 1206), *wxPrintDialog Overview* (p. 2130)

**wxPrintDialogData::wxPrintDialogData****wxPrintDialogData()**

Default constructor.

**wxPrintDialogData(wxPrintDialogData& *dialogData*)**

Copy constructor.

**wxPrintDialogData(wxPrintData& *printData*)**

Construct an object from a print dialog data object.

**wxPrintDialogData::~~wxPrintDialogData****~wxPrintDialogData()**

Destructor.

**wxPrintDialogData::EnableHelp****void EnableHelp(bool *flag*)**

Enables or disables the 'Help' button.

**wxPrintDialogData::EnablePageNumbers****void EnablePageNumbers(bool *flag*)**

Enables or disables the 'Page numbers' controls.

**wxPrintDialogData::EnablePrintToFile****void EnablePrintToFile(bool *flag*)**

Enables or disables the 'Print to file' checkbox.

**wxPrintDialogData::EnableSelection****void EnableSelection(bool *flag*)**

Enables or disables the 'Selection' radio button.



**wxPrintDialogData::GetAllPages****bool GetAllPages() const**

Returns true if the user requested that all pages be printed.

**wxPrintDialogData::GetCollate****bool GetCollate() const**

Returns true if the user requested that the document(s) be collated.

**wxPrintDialogData::GetFromPage****int GetFromPage() const**

Returns the *from* page number, as entered by the user.

**wxPrintDialogData::GetMaxPage****int GetMaxPage() const**

Returns the *maximum* page number.

**wxPrintDialogData::GetMinPage****int GetMinPage() const**

Returns the *minimum* page number.

**wxPrintDialogData::GetNoCopies****int GetNoCopies() const**

Returns the number of copies requested by the user.

**wxPrintDialogData::GetPrintData****wxPrintData& GetPrintData()**

Returns a reference to the internal wxPrintData object.

**wxPrintDialogData::GetPrintToFile****bool GetPrintToFile() const**

Returns true if the user has selected printing to a file.

**wxPrintDialogData::GetSelection**

**bool GetSelection() const**

Returns true if the user requested that the selection be printed (where 'selection' is a concept specific to the application).

**wxPrintDialogData::GetToPage****int GetToPage() const**

Returns the *to* page number, as entered by the user.

**wxPrintDialogData::IsOk****bool IsOk() const**

Returns true if the print data is valid for using in print dialogs. This can return false on Windows if the current printer is not set, for example. On all other platforms, it returns true.

**wxPrintDialogData::SetCollate****void SetCollate(bool *flag*)**

Sets the 'Collate' checkbox to true or false.

**wxPrintDialogData::SetFromPage****void SetFromPage(int *page*)**

Sets the *from* page number.

**wxPrintDialogData::SetMaxPage****void SetMaxPage(int *page*)**

Sets the *maximum* page number.

**wxPrintDialogData::SetMinPage****void SetMinPage(int *page*)**

Sets the *minimum* page number.

**wxPrintDialogData::SetNoCopies****void SetNoCopies(int *n*)**

Sets the default number of copies the user has requested to be printed out.

**wxPrintDialogData::SetPrintData**

**void SetPrintData(const wxPrintData& *printData*)**

Sets the internal wxPrintData.

**wxPrintDialogData::SetPrintToFile**

**void SetPrintToFile(bool *flag*)**

Sets the 'Print to file' checkbox to true or false.

**wxPrintDialogData::SetSelection**

**void SetSelection(bool *flag*)**

Selects the 'Selection' radio button. The effect of printing the selection depends on how the application implements this command, if at all.

**wxPrintDialogData::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).

This function has been deprecated since version 2.5.4.

**wxPrintDialogData::SetToPage**

**void SetToPage(int *page*)**

Sets the *to* page number.

**wxPrintDialogData::operator =**

**void operator =(const wxPrintData& *data*)**

Assigns print data to this object.

**void operator =(const wxPrintDialogData& *data*)**

Assigns another print dialog data object to this object.

## **wxPrinter**

This class represents the Windows or PostScript printer, and is the vehicle through which printing may be launched by an application. Printing can also be achieved through using of lower functions and classes, but this and associated classes provide a more convenient and general method of printing.

**Derived from**

*wxObject* (p. 1148)

#### **Include files**

<wx/print.h>

#### **See also**

*Printing framework overview* (p. 2145), *wxPrinterDC* (p. 1213), *wxPrintDialog* (p. 1206), *wxPrintout* (p. 1214), *wxPrintPreview* (p. 1221).

### **wxPrinter::wxPrinter**

**wxPrinter(wxPrintDialogData\* data = NULL)**

Constructor. Pass an optional pointer to a block of print dialog data, which will be copied to the printer object's local data.

#### **See also**

*wxPrintDialogData* (p. 1207), *wxPrintData* (p. 1200)

### **wxPrinter::CreateAbortWindow**

**void CreateAbortWindow(wxWindow\* parent, wxPrintout\* printout)**

Creates the default printing abort window, with a cancel button.

### **wxPrinter::GetAbort**

**bool GetAbort()**

Returns true if the user has aborted the print job.

### **wxPrinter::GetLastError**

**static wxPrinterError GetLastError()**

Return last error. Valid after calling *Print* (p. 1213), *PrintDialog* (p. 1213) or *wxPrintPreview::Print* (p. 1223). These functions set last error to **wxPRINTER\_NO\_ERROR** if no error happened.

Returned value is one of the following:

<b>wxPRINTER_NO_ERROR</b>	No error happened.
<b>wxPRINTER_CANCELLED</b>	The user cancelled printing.
<b>wxPRINTER_ERROR</b>	There was an error during printing.

**wxPrinter::GetPrintDialogData****wxPrintDialogData& GetPrintDialogData()**

Returns the *print data* (p. 1200) 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) or the user cancelled printing. Call *wxPrinter::GetLastError* (p. 1212) to get detailed information about the kind of the error.

**wxPrinter::PrintDialog****wxDC\* PrintDialog(wxWindow \*parent)**

Invokes the print dialog. If successful (the user did not press Cancel and no error occurred), a suitable device context will be returned (otherwise NULL is returned -- call *wxPrinter::GetLastError* (p. 1212) to get detailed information about the kind of the error).

The application must delete this device context to avoid a memory leak.

**wxPrinter::ReportError****void ReportError(wxWindow \*parent, wxPrintout \*printout, const wxString& message)**

Default error-reporting function.

**wxPrinter::Setup****bool Setup(wxWindow \*parent)**

Invokes the print setup dialog. Note that the setup dialog is obsolete from Windows 95, though retained for backward compatibility.

**wxPrinterDC**

A printer device context is specific to MSW and Mac, and allows access to any printer with a Windows or Macintosh driver. See *wxDC* (p. 456) for further information on device contexts, and *wxDC::GetSize* (p. 468) for advice on achieving the correct scaling for the page.

**Derived from**

*wxDC* (p. 456)

*wxObject* (p. 456)

**Include files**

<wx/dcprint.h>

**See also**

*Printing framework overview* (p. 2145), *wxDC* (p. 456)

**wxPrinterDC::wxPrinterDC**

**wxPrinterDC(const wxPrintData& *printData*)**

Constructor. Pass a *wxPrintData* (p. 1200) object with information necessary for setting up a suitable printer device context. This is the recommended way to construct a *wxPrinterDC*. Make sure you specify a reference to a *wxPrintData* (p. 1200) object, not a pointer - you may not even get a warning if you pass a pointer instead.

**wxPrinterDC(const wxString& *driver*, const wxString& *device*, const wxString& *output*, const bool *interactive* = true, int *orientation* = wxPORTRAIT)**

Constructor. With empty strings for the first three arguments, the default printer dialog is displayed. *device* indicates the type of printer and *output* is an optional file for printing to. The *driver* parameter is currently unused. Use the *Ok* member to test whether the constructor was successful in creating a usable device context.

This constructor is deprecated and retained only for backward compatibility.

**wxPrinterDC::GetPaperRect**

**wxRect wxPrinterDC::GetPaperRect()**

Return the rectangle in device coordinates that corresponds to the full paper area, including the nonprinting regions of the paper. The point (0,0) in device coordinates is the top left corner of the page rectangle, which is the printable area on MSW and Mac. The coordinates of the top left corner of the paper rectangle will therefore have small negative values, while the bottom right coordinates will be somewhat larger than the values returned by *wxDC::GetSize* (p. 468).

**wxPrintout**

This class encapsulates the functionality of printing out an application document. A new class must be derived and members overridden to respond to calls such as *OnPrintPage* and *HasPage* and to render the print image onto an associated *wxDC* (p. 456). Instances of this class are passed to *wxPrinter::Print* or to a *wxPrintPreview* object to initiate printing

or previewing.

Your derived `wxPrintout` is responsible for drawing both the preview image and the printed page. If your windows' drawing routines accept an arbitrary DC as an argument, you can re-use those routines within your `wxPrintout` subclass to draw the printout image. You may also add additional drawing elements within your `wxPrintout` subclass, like headers, footers, and/or page numbers. However, the image on the printed page will often differ from the image drawn on the screen, as will the print preview image -- not just in the presence of headers and footers, but typically in scale. A high-resolution printer presents a much larger drawing surface (i.e., a higher-resolution DC); a zoomed-out preview image presents a much smaller drawing surface (lower-resolution DC). By using the routines `FitThisSizeToXXX()` and/or `MapScreenSizeToXXX()` within your `wxPrintout` subclass to set the user scale and origin of the associated DC, you can easily use a single drawing routine to draw on your application's windows, to create the print preview image, and to create the printed paper image, and achieve a common appearance to the preview image and the printed page.

#### **Derived from**

`wxObject` (p. 1148)

#### **Include files**

<wx/print.h>

#### **See also**

*Printing framework overview* (p. 2145), *wxPrinterDC* (p. 1213), *wxPrintDialog* (p. 1206), *wxPageSetupDialog* (p. 1158), *wxPrinter* (p. 1211), *wxPrintPreview* (p. 1221)

### **wxPrintout::wxPrintout**

**wxPrintout(const wxString& title = "Printout")**

Constructor. Pass an optional title argument - the current filename would be a good idea. This will appear in the printing list (at least in MSW)

### **wxPrintout::~~wxPrintout**

**~wxPrintout()**

Destructor.

### **wxPrintout::GetDC**

**wxDC \* GetDC()**

Returns the device context associated with the printout (given to the printout at start of printing or previewing). This will be a `wxPrinterDC` if printing under Windows or Mac, 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.

**wxPython note:** When this method is implemented in a derived Python class, it should be designed to take no parameters (other than the self reference) and to return a tuple of four integers.

**wxPerl note:** When this method is overridden in a derived class, it must not take any parameters, and returns a 4-element list.

**wxPrintout::GetPageSizeMM****void GetPageSizeMM(int \*w, int \*h)**

Returns the size of the printer page in millimetres.

**wxPython note:** This method returns the output-only parameters as a tuple.

**wxPerl note:** In wxPerl this method takes no arguments and returns a 2-element list ( *w*, *h* )

**wxPrintout::GetPageSizePixels****void GetPageSizePixels(int \*w, int \*h)**

Returns the size of the printer page in pixels, called the *page rectangle*. The page rectangle has a top left corner at (0,0) and a bottom right corner at (w,h). These values may not be the same as the values returned from `wxDC::GetSize` (p. 468); if the printout is being used for previewing, a memory device context is used, which uses a bitmap size reflecting the current preview zoom. The application must take this discrepancy into account if previewing is to be supported.

**wxPython note:** This method returns the output-only parameters as a tuple.

**wxPerl note:** In wxPerl this method takes no arguments and returns a 2-element list ( *w*, *h* )

**wxPrintout::GetPaperRectPixels****wxRect GetPaperRectPixels()**

Returns the rectangle that corresponds to the entire paper in pixels, called the *paper rectangle*. This distinction between paper rectangle and page rectangle reflects the fact that most printers cannot print all the way to the edge of the paper. The page rectangle is



a rectangle whose top left corner is at (0,0) and whose width and height are given by `wxDC::GetPageSizePixels` (p. 1216). On MSW and Mac, the page rectangle gives the printable area of the paper, while the paper rectangle represents the entire paper, including non-printable borders. Thus, the rectangle returned by `GetPaperRectPixels` will have a top left corner whose coordinates are small negative numbers and the bottom right corner will have values somewhat larger than the width and height given by `wxDC::GetPageSizePixels` (p. 1216). On other platforms and for PostScript printing, the paper is treated as if its entire area were printable, so this function will return the same rectangle as the page rectangle.

### **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. Or you can just use the `FitThisSizeToXXX()` and `MapScreenSizeToXXX` routines below, which do most of the scaling calculations for you.

**wxPython note:** This method returns the output-only parameters as a tuple.

**wxPerl note:** In wxPerl this method takes no arguments and returns a 2-element list ( `w`, `h` )

### **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. If you are doing your own scaling, remember to multiply this by a scaling factor to take the preview DC size into account.

### **wxPrintout::GetTitle**

**wxString GetTitle()**

Returns the title of the printout

**wxPython note:** This method returns the output-only parameters as a tuple.

**wxPerl note:** In wxPerl this method takes no arguments and returns a 2-element list ( `w`, `h` )

### **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()**

Returns true if the printout is currently being used for previewing.

### **wxPrintout::FitThisSizeToPaper**

**void FitThisSizeToPaper(const wxSize& imageSize)**

Set the user scale and device origin of the wxDC associated with this wxPrintout so that the given image size fits entirely within the paper and the origin is at the top left corner of the paper. Note that with most printers, the region around the edges of the paper are not printable so that the edges of the image could be cut off. Use this if you're managing your own page margins.

### **wxPrintout::FitThisSizeToPage**

**void FitThisSizeToPage(const wxSize& imageSize)**

Set the user scale and device origin of the wxDC associated with this wxPrintout so that the given image size fits entirely within the page rectangle and the origin is at the top left corner of the page rectangle. On MSW and Mac, the page rectangle is the printable area of the page. On other platforms and PostScript printing, the page rectangle is the entire paper. Use this if you want your printed image as large as possible, but with the caveat that on some platforms, portions of the image might be cut off at the edges.

### **wxPrintout::FitThisSizeToPageMargins**

**void FitThisSizeToPageMargins(const wxSize& imageSize, const wxPageSetupDialogData& pageSetupData)**

Set the user scale and device origin of the wxDC associated with this wxPrintout so that the given image size fits entirely within the page margins set in the given wxPageSetupDialogData object. This function provides the greatest consistency across all platforms because it does not depend on having access to the printable area of the paper. Note that on Mac, the native wxPageSetupDialog does not let you set the page margins; you'll have to provide your own mechanism, or you can use the Mac-only class wxMacPageMarginsDialog.

### **wxPrintout::MapScreenSizeToPaper**

**void MapScreenSizeToPaper()**

Set the user scale and device origin of the wxDC associated with this wxPrintout so that the printed page matches the screen size as closely as possible and the logical origin is in the top left corner of the paper rectangle. That is, a 100-pixel object on screen should appear at the same size on the printed page. (It will, of course, be larger or smaller in the

preview image, depending on the zoom factor.) Use this if you want WYSIWYG behavior, e.g., in a text editor.

### **wxPrintout::MapScreenSizeToPage**

#### **void MapScreenSizeToPage()**

This sets the user scale of the wxDC associated with this wxPrintout to the same scale as *MapScreenSizeToPaper* (p. 1218) but sets the logical origin to the top left corner of the page rectangle.

### **wxPrintout::MapScreenSizeToPageMargins**

#### **void MapScreenSizeToPageMargins(const wxPageSetupDialogData& pageSetupData)**

This sets the user scale of the wxDC associated with this wxPrintout to the same scale as *MapScreenSizeToPageMargins* (p. 1219) but sets the logical origin to the top left corner of the page margins specified by the given wxPageSetupDialogData object.

### **wxPrintout::MapScreenSizeToDevice**

#### **void MapScreenSizeToDevice()**

Set the user scale and device origin of the wxDC associated with this wxPrintout so that one screen pixel maps to one device pixel on the DC. That is, the user scale is set to (1,1) and the device origin is set to (0,0). Use this if you want to do your own scaling prior to calling wxDC drawing calls, for example, if your underlying model is floating-point and you want to achieve maximum drawing precision on high-resolution printers. (Note that while the underlying drawing model of Mac OS X is floating-point, wxWidgets's drawing model scales from integer coordinates.) You can use the *GetLogicalXXXRect()* routines below to obtain the paper rectangle, page rectangle, or page margins rectangle to perform your own scaling.

### **wxPrintout::GetLogicalPaperRect**

#### **wxRect GetLogicalPaperRect()**

Return the rectangle corresponding to the paper in the associated wxDC's logical coordinates for the current user scale and device origin.

### **wxPrintout::GetLogicalPageRect**

#### **wxRect GetLogicalPageRect()**

Return the rectangle corresponding to the page in the associated wxDC's logical coordinates for the current user scale and device origin. On MSW and Mac, this will be the printable area of the paper. On other platforms and PostScript printing, this will be the full paper rectangle.

**wxPrintout::GetLogicalPageMarginsRect****wxRect GetLogicalPageMarginsRect(const wxPageSetupDialogData& pageSetupData)**

Return the rectangle corresponding to the page margins specified by the given wxPageSetupDialogData object in the associated wxDC's logical coordinates for the current user scale and device origin. The page margins are specified with respect to the edges of the paper on all platforms.

**wxPrintout::SetLogicalOrigin****void SetLogicalOrigin(wxCoord x, wxCoord y)**

Set the device origin of the associated wxDC so that the current logical point becomes the new logical origin.

**wxPrintout::OffsetLogicalOrigin****void OffsetLogicalOrigin(wxCoord xoff, wxCoord yoff)**

Shift the device origin by an amount specified in logical coordinates.

**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.

**wxPython note:** If this method is overridden in a Python class then the base class version can be called by using the method `base_OnBeginDocument(startPage, endPage)`.

**wxPrintout::OnEndDocument****void OnEndDocument()**

Called by the framework at the end of document printing. OnEndDocument is called once for every copy printed.

The base wxPrintout::OnEndDocument *must* be called from within the overridden function, since it calls wxDC::EndDoc.

**wxPrintout::OnBeginPrinting****void OnBeginPrinting()**

Called by the framework at the start of printing. OnBeginPrinting is called once for every

print job (regardless of how many copies are being printed).

### **wxPrintout::OnEndPrinting**

**void OnEndPrinting()**

Called by the framework at the end of printing. OnEndPrinting is called once for every print job (regardless of how many copies are being printed).

### **wxPrintout::OnPreparePrinting**

**void OnPreparePrinting()**

Called once by the framework before any other demands are made of the wxPrintout object. This gives the object an opportunity to calculate the number of pages in the document, for example.

### **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.

## **wxPrintPreview**

Objects of this class manage the print preview process. The object is passed a wxPrintout object, and the wxPrintPreview object itself is passed to a wxPreviewFrame object. Previewing is started by initializing and showing the preview frame. Unlike wxPrinter::Print, flow of control returns to the application immediately after the frame is shown.

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/print.h>

### **See also**

*Printing framework overview* (p. 2145), *wxPrinterDC* (p. 1213), *wxPrintDialog* (p. 1206), *wxPrintout* (p. 1214), *wxPrinter* (p. 1211), *wxPreviewCanvas* (p. 1196), *wxPreviewControlBar* (p. 1197), *wxPreviewFrame* (p. 1199).

### **wxPrintPreview::wxPrintPreview**

**wxPrintPreview(wxPrintout\* printout, wxPrintout\* printoutForPrinting, wxPrintData\* data=NULL)**

Constructor. Pass a printout object, an optional printout object to be used for actual printing, and the address of an optional block of printer data, which will be copied to the print preview object's print data.

If *printoutForPrinting* is non-NULL, a **Print...** button will be placed on the preview frame so that the user can print directly from the preview interface.

Do not explicitly delete the printout objects once this destructor has been called, since they will be deleted in the `wxPrintPreview` constructor. The same does not apply to the *data* argument.

Test the `Ok` member to check whether the `wxPrintPreview` object was created correctly. `Ok` could return false if there was a problem initializing the printer device context (current printer not set, for example).

### **wxPrintPreview::~~wxPrintPreview**

**~wxPrinter()**

Destructor. Deletes both print preview objects, so do not destroy these objects in your application.

### **wxPrintPreview::GetCanvas**

**wxPreviewCanvas\* GetCanvas()**

Gets the preview window used for displaying the print preview image.

### **wxPrintPreview::GetCurrentPage**

**int GetCurrentPage()**

Gets the page currently being previewed.

### **wxPrintPreview::GetFrame**

**wxFrame \* GetFrame()**

Gets the frame used for displaying the print preview canvas and control bar.

### **wxPrintPreview::GetMaxPage**

**int GetMaxPage()**

Returns the maximum page number.

### **wxPrintPreview::GetMinPage**

**int GetMinPage()**

Returns the minimum page number.

**wxPrintPreview::GetPrintout****wxPrintout \* GetPrintout()**

Gets the preview printout object associated with the wxPrintPreview object.

**wxPrintPreview::GetPrintoutForPrinting****wxPrintout \* GetPrintoutForPrinting()**

Gets the printout object to be used for printing from within the preview interface, or NULL if none exists.

**wxPrintPreview::IsOk****bool Ok()**

Returns true if the wxPrintPreview is valid, false otherwise. It could return false if there was a problem initializing the printer device context (current printer not set, for example).

**wxPrintPreview::PaintPage****bool PaintPage(wxPreviewCanvas \*canvas, wxDC&dc)**

This refreshes the preview window with the preview image. It must be called from the preview window's OnPaint member.

The implementation simply blits the preview bitmap onto the canvas, creating a new preview bitmap if none exists.

**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.

Returns false in case of error -- call *wxPrinter::GetLastError* (p. 1212) to get detailed information about the kind of the error.

**wxPrintPreview::RenderPage****bool RenderPage(int pageNum)**

Renders a page into a wxMemoryDC. Used internally by wxPrintPreview.

**wxPrintPreview::SetCanvas****void SetCanvas(wxPreviewCanvas\* window)**

Sets the window 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.

## **wxProcess**

The objects of this class are used in conjunction with the *wxExecute* (p. 1913) function. When a *wxProcess* object is passed to *wxExecute*(), its *OnTerminate()* (p. 1228) virtual method is called when the process terminates. This allows the program to be (asynchronously) notified about the process termination and also retrieve its exit status which is unavailable from *wxExecute()* in the case of asynchronous execution.

Please note that if the process termination notification is processed by the parent, it is responsible for deleting the *wxProcess* object which sent it. However, if it is not processed, the object will delete itself and so the library users should only delete those objects whose notifications have been processed (and call *Detach()* (p. 1226) for others).

*wxProcess* also supports IO redirection of the child process. For this, you have to call its *Redirect* (p. 1229) method before passing it to *wxExecute* (p. 1913). If the child process was launched successfully, *GetInputStream* (p. 1226), *GetOutputStream* (p. 1226) and *GetErrorStream* (p. 1226) can then be used to retrieve the streams corresponding to the child process standard output, input and error output respectively.

**wxPerl note:** In *wxPerl* this class has an additional *Destroy* method, for explicit destruction.

### **Derived from**



*wxEvtHandler* (p. 576)

### Include files

<wx/process.h>

### See also

*wxExecute* (p. 1913)

*exec sample* (p. 2034)

## **wxProcess::wxProcess**

**wxProcess**(*wxEvtHandler* \* *parent* = *NULL*, *int* *id* = -1)

**wxProcess**(*int* *flags*)

Constructs a process object. *id* is only used in the case you want to use *wxWidgets* events. It identifies this object, or another window that will receive the event.

If the *parent* parameter is different from *NULL*, it will receive a *wxEVT\_END\_PROCESS* notification event (you should insert *EVT\_END\_PROCESS* macro in the event table of the parent to handle it) with the given *id*.

The second constructor creates an object without any associated parent (and hence no *id* neither) but allows to specify the *flags* which can have the value of *wxPROCESS\_DEFAULT* or *wxPROCESS\_REDIRECT*. Specifying the former value has no particular effect while using the latter one is equivalent to calling *Redirect* (p. 1229).

### Parameters

*parent*

The event handler parent.

*id*

id of an event.

*flags*

either *wxPROCESS\_DEFAULT* or *wxPROCESS\_REDIRECT*

## **wxProcess::~~wxProcess**

**~wxProcess**()

Destroys the *wxProcess* object.

## **wxProcess::CloseOutput**

**void CloseOutput()**

Closes the output stream (the one connected to the stdin of the child process). This function can be used to indicate to the child process that there is no more data to be read - usually, a filter program will only terminate when the input stream is closed.

**wxProcess::Detach****void Detach()**

Normally, a wxProcess object is deleted by its parent when it receives the notification about the process termination. However, it might happen that the parent object is destroyed before the external process is terminated (e.g. a window from which this external process was launched is closed by the user) and in this case it **should not delete** the wxProcess object, but **should call Detach()** instead. After the wxProcess object is detached from its parent, no notification events will be sent to the parent and the object will delete itself upon reception of the process termination notification.

**wxProcess::GetErrorStream****wxInputStream\* GetErrorStream() const**

Returns an input stream which corresponds to the standard error output (stderr) of the child process.

**wxProcess::GetInputStream****wxInputStream\* GetInputStream() const**

It returns an input stream corresponding to the standard output stream of the subprocess. If it is NULL, you have not turned on the redirection. See *wxProcess::Redirect* (p. 1229).

**wxProcess::GetOutputStream****wxOutputStream\* GetOutputStream() const**

It returns an output stream corresponding to the input stream of the subprocess. If it is NULL, you have not turned on the redirection. See *wxProcess::Redirect* (p. 1229).

**wxProcess::IsErrorAvailable****bool IsErrorAvailable() const**

Returns `true` if there is data to be read on the child process standard error stream.

**See also**

*IsInputAvailable* (p. 1226)

**wxProcess::IsInputAvailable**

**bool IsInputAvailable() const**

Returns `true` if there is data to be read on the child process standard output stream. This allows to write simple (and extremely inefficient) polling-based code waiting for a better mechanism in future `wxWidgets` versions.

See the *exec sample* (p. 2034) for an example of using this function.

**See also**

*IsInputOpened* (p. 1227)

**wxProcess::IsInputOpened****bool IsInputOpened() const**

Returns `true` if the child process standard output stream is opened.

**wxProcess::Kill**

**static wxKillError Kill(int pid, wxSignal signal = wxSIGNONE, int flags = wxKILL\_NOCHILDREN)**

Send the specified signal to the given process. Possible signal values are:

```
enum wxSignal
{
    wxSIGNONE = 0, // verify if the process exists under Unix
    wxSIGHUP,
    wxSIGINT,
    wxSIGQUIT,
    wxSIGILL,
    wxSIGTRAP,
    wxSIGABRT,
    wxSIGEMT,
    wxSIGFPE,
    wxSIGKILL,      // forcefully kill, dangerous!
    wxSIGBUS,
    wxSIGSEGV,
    wxSIGSYS,
    wxSIGPIPE,
    wxSIGALRM,
    wxSIGTERM       // terminate the process gently
};
```

`wxSIGNONE`, `wxSIGKILL` and `wxSIGTERM` have the same meaning under both Unix and Windows but all the other signals are equivalent to `wxSIGTERM` under Windows.

The *flags* parameter can be `wxKILL_NOCHILDREN` (the default), or `wxKILL_CHILDREN`, in which case the child processes of this process will be killed too. Note that under Unix, for `wxKILL_CHILDREN` to work you should have created the process passing `wxEXEC_MAKE_GROUP_LEADER`.

Returns the element of `wxKillError` enum:

```
enum wxKillError
{
    wxKILL_OK,                // no error
    wxKILL_BAD_SIGNAL,        // no such signal
    wxKILL_ACCESS_DENIED,     // permission denied
    wxKILL_NO_PROCESS,        // no such process
    wxKILL_ERROR              // another, unspecified error
};
```

**See also**

*wxProcess::Exists* (p. 1228), *wxKill* (p. 1915), *Exec sample* (p. 2034)

**wxProcess::Exists**

**static bool Exists(int pid)**

Returns `true` if the given process exists in the system.

**See also**

*wxProcess::Kill* (p. 1227), *Exec sample* (p. 2034)

**wxProcess::OnTerminate**

**void OnTerminate(int pid, int status)**

It is called when the process with the pid *pid* finishes. It raises a `wxWidgets` event when it isn't overridden.

*pid*

The pid of the process which has just terminated.

*status*

The exit code of the process.

**wxProcess::Open**

**static wxProcess \* Open(const wxString& cmd, int flags = wxEXEC\_ASYNC)**

This static method replaces the standard `popen()` function: it launches the process specified by the *cmd* parameter and returns the `wxProcess` object which can be used to retrieve the streams connected to the standard input, output and error output of the child process.

If the process couldn't be launched, `NULL` is returned. Note that in any case the returned pointer should **not** be deleted, rather the process object will be destroyed automatically when the child process terminates. This does mean that the child process should be told to quit before the main program exits to avoid memory leaks.

**Parameters**

*cmd*

The command to execute, including optional arguments.

*flags*

The flags to pass to *wxExecute* (p. 1913). NOTE: *wxEXEC\_SYNC* should not be used.

### **Return value**

A pointer to new *wxProcess* object or *NULL* on error.

### **See also**

*wxExecute* (p. 1913)

### **wxProcess::GetPid**

#### **long GetPid() const**

Returns the process ID of the process launched by *Open* (p. 1228).

### **wxProcess::Redirect**

#### **void Redirect()**

Turns on redirection. *wxExecute* will try to open a couple of pipes to catch the subprocess *stdio*. The caught input stream is returned by *GetOutputStream()* as a non-seekable stream. The caught output stream is returned by *GetInputStream()* as a non-seekable stream.

## **wxProcessEvent**

A process event is sent when a process is terminated.

### **Derived from**

*wxEvent* (p. 572)

*wxObject* (p. 1148)

### **Include files**

<wx/process.h>

### **Event table macros**

To process a *wxProcessEvent*, use these event handler macros to direct input to a member function that takes a *wxProcessEvent* argument.

#### **EVT\_END\_PROCESS(id, func)**

Process a *wxEVT\_END\_PROCESS* event. *id* is the identifier of the process object (the *id* passed

to the `wxProcess` constructor) or a window to receive the event.

**See also**

`wxProcess` (p. 1224), *Event handling overview* (p. 2077)

**wxProcessEvent::wxProcessEvent**

**wxProcessEvent**(int *id* = 0, int *pid* = 0, int *exitcode* = 0)

Constructor. Takes a `wxProcessObject` or window id, a process id and an exit status.

**wxProcessEvent::GetPid**

**int GetPid()** const

Returns the process id.

**wxProcessEvent::GetExitCode**

**int GetExitCode()**

Returns the exist status.

## wxProgressDialog

This class represents a dialog that shows a short message and a progress bar. Optionally, it can display ABORT and SKIP buttons, the elapsed, remaining and estimated time for the end of the progress.

**Derived from**

`wxDialog` (p. 496)

`wxWindow` (p. 1795)

`wxEvtHandler` (p. 576)

`wxObject` (p. 1148)

**Include files**

<wx/progdlg.h>

**Window styles**

**wxPD\_APP\_MODAL**

Make the progress dialog modal. If this flag is not given, it is only "locally" modal - that is the input to the parent window is disabled, but not to the other ones.

**wxPD\_AUTO\_HIDE**

Causes the progress dialog to disappear from

	screen as soon as the maximum value of the progress meter has been reached.
<b>wxPD_SMOOTH</b>	Causes smooth progress of the gauge control.
<b>wxPD_CAN_ABORT</b>	This flag tells the dialog that it should have a "Cancel" button which the user may press. If this happens, the next call to <i>Update()</i> (p. 1232) will return false.
<b>wxPD_CAN_SKIP</b>	This flag tells the dialog that it should have a "Skip" button which the user may press. If this happens, the next call to <i>Update()</i> (p. 1232) will return true in its skip parameter.
<b>wxPD_ELAPSED_TIME</b>	This flag tells the dialog that it should show elapsed time (since creating the dialog).
<b>wxPD_ESTIMATED_TIME</b>	This flag tells the dialog that it should show estimated time.
<b>wxPD_REMAINING_TIME</b>	This flag tells the dialog that it should show remaining time.

### **wxProgressDialog::wxProgressDialog**

**wxProgressDialog**(const wxString& *title*, const wxString& *message*, int *maximum* = 100, wxWindow \* *parent* = NULL, int *style* = wxPD\_AUTO\_HIDE | wxPD\_APP\_MODAL)

Constructor. Creates the dialog, displays it and disables user input for other windows, or, if wxPD\_APP\_MODAL flag is not given, for its parent window only.

#### **Parameters**

*title*

Dialog title to show in titlebar.

*message*

Message displayed above the progress bar.

*maximum*

Maximum value for the progress bar.

*parent*

Parent window.

*style*

The dialog style. See *wxProgressDialog* (p. 1230).

## **wxProgressDialog::~wxProgressDialog**

### **~wxProgressDialog()**

Destructor. Deletes the dialog and enables all top level windows.

## **wxProgressDialog::Resume**

### **void Resume()**

Can be used to continue with the dialog, after the user had chosen ABORT.

## **wxProgressDialog::Update**

### **virtual bool Update(int value, const wxString& newmsg = "", bool \*skip = NULL)**

Updates the dialog, setting the progress bar to the new value and, if given changes the message above it. Returns `true` unless the Cancel button has been pressed.

If `false` is returned, the application can either immediately destroy the dialog or ask the user for the confirmation and if the abort is not confirmed the dialog may be resumed with *Resume* (p. 1232) function.

### **Parameters**

#### *value*

The new value of the progress meter. It should be less than or equal to the maximum value given to the constructor and the dialog is closed if it is equal to the maximum.

#### *newmsg*

The new messages for the progress dialog text, if it is empty (which is the default) the message is not changed.

#### *skip*

If "Skip" button was pressed since last *Update* (p. 1232) call, this is set to true.

## **wxProgressDialog::Pulse**

### **virtual bool Pulse(const wxString& newmsg = "", bool \*skip = NULL)**

Just like *Update* (p. 1232) but makes the gauge control run in indeterminate mode (see *wxGauge* (p. 701) documentation), sets the remaining and the estimated time labels (if present) to `Unknown` and moves the progress bar a bit to indicate that some progress was done.

## **wxPropertySheetDialog**

This class represents a property sheet dialog: a tabbed dialog for showing settings. It is



optimized to show flat tabs on PocketPC devices, and can be customized to use different controllers instead of the default notebook style.

To use this class, call *wxPropertySheetDialog::Create* (p. 1234) from your own *Create* function. Then call *CreateButtons* (p. 1234), and create pages, adding them to the book control. Finally call *LayoutDialog* (p. 1234).

For example:

```
bool MyPropertySheetDialog::Create(...)
{
    if (!wxPropertySheetDialog::Create(...))
        return false;

    CreateButtons(wxOK | wxCANCEL | wxHELP);

    // Add page
    wxPanel* panel = new wxPanel(GetBookCtrl(), ...);
    GetBookCtrl()->AddPage(panel, wxT("General"));

    LayoutDialog();
    return true;
}
```

If necessary, override *CreateBookCtrl* and *AddBookCtrl* to create and add a different kind of book control. You would then need to use two-step construction for the dialog. Or, change the style of book control by calling *SetSheetStyle* (p. 1235) before calling *Create*.

The dialogs sample shows this class being used with notebook and toolbook controllers (for Windows-style and Mac-style settings dialogs).

#### Derived from

*wxDialog* (p. 496)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

#### Include files

<wx/propdlg.h> <wx/generic/propdlg.h>

#### wxPropertySheetDialog::wxPropertySheetDialog

**wxPropertySheetDialog**(*wxWindow\** parent, *wxWindowID* id, **const wxString&** title, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = *wxDEFAULT\_DIALOG\_STYLE*, **const wxString&** name = "dialogBox")

Constructor.

#### wxPropertySheetDialog::AddBookCtrl

**virtual void AddBookCtrl(wxSizer\* sizer)**

Override this if you wish to add the book control in a way different from the standard way (for example, using different spacing).

**wxPropertySheetDialog::Create**

**bool Create(wxWindow\* parent, wxWindowID id, const wxString& title, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT\_DIALOG\_STYLE, const wxString& name = "dialogBox")**

Call this from your own Create function, before adding buttons and pages.

**wxPropertySheetDialog::CreateBookCtrl**

**virtual wxBookCtrlBase\* CreateBookCtrl()**

Override this if you wish to create a different kind of book control; by default, the value passed to *SetSheetStyle* (p. 1235) is used to determine the control. The default behaviour is to create a notebook except on Smartphone, where a choicebook is used.

**wxPropertySheetDialog::CreateButtons**

**void CreateButtons(int flags=wxOK|wxCANCEL)**

Call this to create the buttons for the dialog. This calls *wxDialog::CreateButtonSizer* (p. 499), and the flags are the same. On PocketPC, no buttons are created.

**wxPropertySheetDialog::GetBookCtrl**

**wxBookCtrlBase\* GetBookCtrl() const**

Returns the book control that will contain your settings pages.

**wxPropertySheetDialog::GetInnerSizer**

**wxSizer\* GetInnerSizer() const**

Returns the inner sizer that contains the book control and button sizer.

**wxPropertySheetDialog::GetSheetStyle**

**long GetSheetStyle() const**

Returns the sheet style. See *SetSheetStyle* (p. 1235) for permissible values.

**wxPropertySheetDialog::LayoutDialog**

**void LayoutDialog(int centreFlags=wxBOTH)**

Call this to lay out the dialog. On PocketPC, this does nothing, since the dialog will be shown full-screen, and the layout will be done when the dialog receives a size event.

### **wxPropertySheetDialog::SetBookCtrl**

**void SetBookCtrl(wxBookCtrlBase\* bookCtrl)**

Sets the book control used for the dialog. You will normally not need to use this.

### **wxPropertySheetDialog::SetInnerSizer**

**void SetInnerSizer(wxSizer\* sizer)**

Sets the inner sizer that contains the book control and button sizer. You will normally not need to use this.

### **wxPropertySheetDialog::SetSheetStyle**

**void SetSheetStyle(long style)**

You can customize the look and feel of the dialog by setting the sheet style. It is a bit list of the following values:

**wxPROPSHEET\_DEFAULT** Uses the default look and feel for the controller window, normally a notebook except on Smartphone where a choice control is used.

**wxPROPSHEET\_NOTEBOOK** Uses a notebook for the controller window.

**wxPROPSHEET\_TOOLBOOK** Uses a toolbook for the controller window.

**wxPROPSHEET\_CHOICEBOOK** Uses a choicebook for the controller window.

**wxPROPSHEET\_LISTBOOK** Uses a listbook for the controller window.

**wxPROPSHEET\_TREEBOOK** Uses a treebook for the controller window.

**wxPROPSHEET\_SHRINKTOFIT** Shrinks the dialog window to fit the currently selected page (common behaviour for property sheets on Mac OS X).

## **wxProtocol**

### **Derived from**

*wxSocketClient* (p. 1488) *wxSocketBase* (p. 1472) *wxObject* (p. 1148)

### **Include files**

<wx/protocol/protocol.h>

### **See also**

*wxSocketBase* (p. 1472), *wxURL* (p. 1763)

### **wxProtocol::Reconnect**

#### **bool Reconnect()**

Tries to reestablish a previous opened connection (close and renegotiate connection).

#### **Return value**

true, if the connection is established, else false.

### **wxProtocol::GetInputStream**

#### **wxInputStream \* GetInputStream(const wxString& path)**

Creates a new input stream on the specified path. You can use all but seek functionality of *wxStream*. Seek isn't available on all streams. For example, HTTP or FTP streams don't deal with it. Other functions like *StreamSize* and *Tell* aren't available for the moment for this sort of stream. You will be notified when the EOF is reached by an error.

#### **Return value**

Returns the initialized stream. You will have to delete it yourself once you don't use it anymore. The destructor closes the network connection.

#### **See also**

*wxInputStream* (p. 941)

### **wxProtocol::Abort**

#### **bool Abort()**

Abort the current stream.

#### **Warning**

It is advised to destroy the input stream instead of aborting the stream this way.

#### **Return value**

Returns true, if successful, else false.

### **wxProtocol::GetError**

#### **wxProtocolError GetError()**

Returns the last occurred error.

#### **wxPROTO\_NOERR**

No error.

<b>wxPROTO_NETERR</b>	A generic network error occurred.
<b>wxPROTO_PROTERR</b>	An error occurred during negotiation.
<b>wxPROTO_CONNERR</b>	The client failed to connect the server.
<b>wxPROTO_INVVAL</b>	Invalid value.
<b>wxPROTO_NOHNDLR</b>	.
<b>wxPROTO_NOFILE</b>	The remote file doesn't exist.
<b>wxPROTO_ABRT</b>	Last action aborted.
<b>wxPROTO_RCNCT</b>	An error occurred during reconnection.
<b>wxPROTO_STREAM</b>	Someone tried to send a command during a transfer.

### **wxProtocol::GetContentType**

**wxString GetContentType()**

Returns the type of the content of the last opened stream. It is a mime-type.

### **wxProtocol::SetUser**

**void SetUser(const wxString& user)**

Sets the authentication user. It is mainly useful when FTP is used.

### **wxProtocol::SetPassword**

**void SetPassword(const wxString& user)**

Sets the authentication password. It is mainly useful when FTP is used.

## **wxQuantize**

Performs quantization, or colour reduction, on a wxImage.

Functions in this class are static and so a wxQuantize object need not be created.

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/quantize.h>

## **wxQuantize::wxQuantize**

### **wxQuantize()**

Constructor. You do not need to construct a wxQuantize object since its functions are static.

## **wxQuantize::Quantize**

**bool Quantize(const wxImage& src, wxImage& dest, wxPalette\*\* pPalette, int desiredNoColours = 236, unsigned char\*\* eightBitData = 0, int flags = wxQUANTIZE\_INCLUDE\_WINDOWS\_COLOURS|wxQUANTIZE\_FILL\_DESTINATION\_IMAGE|wxQUANTIZE\_RETURN\_8BIT\_DATA)**

Reduce the colours in the source image and put the result into the destination image. Both images may be the same, to overwrite the source image. Specify an optional palette pointer to receive the resulting palette. This palette may be passed to ConvertImageToBitmap, for example.

If you pass a palette pointer, you must free the palette yourself.

**bool Quantize(const wxImage& src, wxImage& dest, int desiredNoColours = 236, unsigned char\*\* eightBitData = 0, int flags = wxQUANTIZE\_INCLUDE\_WINDOWS\_COLOURS|wxQUANTIZE\_FILL\_DESTINATION\_IMAGE|wxQUANTIZE\_RETURN\_8BIT\_DATA)**

This version sets a palette in the destination image so you don't have to manage it yourself.

## **wxQuantize::DoQuantize**

**void DoQuantize(unsigned w, unsigned h, unsigned char\*\* in\_rows, unsigned char\*\* out\_rows, unsigned char\* palette, int desiredNoColours)**

Converts input bitmap(s) into 8bit representation with custom palette.

in\_rows and out\_rows are arrays [0..h-1] of pointer to rows (in\_rows contains w \* 3 bytes per row, out\_rows w bytes per row).

Fills out\_rows with indexes into palette (which is also stored into palette variable).

## **wxQueryLayoutInfoEvent**

This event is sent when *wxLayoutAlgorithm* (p. 961) wishes to get the size, orientation and alignment of a window. More precisely, the event is sent by the OnCalculateLayout handler which is itself invoked by wxLayoutAlgorithm.

### **Derived from**

*wxEvent* (p. 572)

*wxObject* (p. 1148)

**Include files**

<wx/laywin.h>

**Event table macros**

**EVT\_QUERY\_LAYOUT\_INFO(func)** Process a wxEVT\_QUERY\_LAYOUT\_INFO event, to get size, orientation and alignment from a window.

**Data structures**

```
enum wxLayoutOrientation {  
    wxLAYOUT_HORIZONTAL,  
    wxLAYOUT_VERTICAL  
};  
  
enum wxLayoutAlignment {  
    wxLAYOUT_NONE,  
    wxLAYOUT_TOP,  
    wxLAYOUT_LEFT,  
    wxLAYOUT_RIGHT,  
    wxLAYOUT_BOTTOM,  
};
```

**See also**

*wxCalculateLayoutEvent* (p. 167), *wxSashLayoutWindow* (p. 1394), *wxLayoutAlgorithm* (p. 961).

**wxQueryLayoutInfoEvent::wxQueryLayoutInfoEvent**

**wxQueryLayoutInfoEvent(wxWindowID id = 0)**

Constructor.

**wxQueryLayoutInfoEvent::GetAlignment**

**void GetAlignment() const**

Specifies the alignment of the window (which side of the remaining parent client area the window sticks to). One of wxLAYOUT\_TOP, wxLAYOUT\_LEFT, wxLAYOUT\_RIGHT, wxLAYOUT\_BOTTOM.

**wxQueryLayoutInfoEvent::GetFlags**

**int GetFlags() const**

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

**wxQueryLayoutInfoEvent::GetOrientation****wxLayoutOrientation GetOrientation() const**

Returns the orientation that the event handler specified to the event object. May be one of `wxLAYOUT_HORIZONTAL`, `wxLAYOUT_VERTICAL`.

**wxQueryLayoutInfoEvent::GetRequestedLength****int GetRequestedLength() const**

Returns the requested length of the window in the direction of the window orientation. This information is not yet used.

**wxQueryLayoutInfoEvent::GetSize****wxSize GetSize() const**

Returns the size that the event handler specified to the event object as being the requested size of the window.

**wxQueryLayoutInfoEvent::SetAlignment****void SetAlignment(wxLayoutAlignment *alignment*)**

Call this to specify the alignment of the window (which side of the remaining parent client area the window sticks to). May be one of `wxLAYOUT_TOP`, `wxLAYOUT_LEFT`, `wxLAYOUT_RIGHT`, `wxLAYOUT_BOTTOM`.

**wxQueryLayoutInfoEvent::SetFlags****void SetFlags(int *flags*)**

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

**wxQueryLayoutInfoEvent::SetOrientation****void SetOrientation(wxLayoutOrientation *orientation*)**

Call this to specify the orientation of the window. May be one of `wxLAYOUT_HORIZONTAL`, `wxLAYOUT_VERTICAL`.

**wxQueryLayoutInfoEvent::SetRequestedLength****void SetRequestedLength(int *length*)**

Sets the requested length of the window in the direction of the window orientation. This information is not yet used.

**wxQueryLayoutInfoEvent::SetSize**



**void SetSize(const wxSize& size)**

Call this to let the calling code know what the size of the window is.

## wxRadioBox

A radio box item is used to select one of number of mutually exclusive choices. It is displayed as a vertical column or horizontal row of labelled buttons.

### Derived from

*wxControlWithItems* (p. 286)

*wxControl* (p. 285)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### Include files

<wx/radiobox.h>

### Window styles

**wxRA\_SPECIFY\_ROWS**      The major dimension parameter refers to the maximum number of rows.

**wxRA\_SPECIFY\_COLS**      The major dimension parameter refers to the maximum number of columns.

**wxRA\_USE\_CHECKBOX**      Use of the checkbox controls instead of radio buttons (currently supported only on PalmOS)

See also *window styles overview* (p. 2089).

### Event handling

**EVT\_RADIOBOX(id, func)**      Process a  
wxEVT\_COMMAND\_RADIOBOX\_SELECTED  
event, when a radiobutton is clicked.

### See also

*Event handling overview* (p. 2077), *wxRadioButton* (p. 1249), *wxCheckBox* (p. 180)

## wxRadioBox::wxRadioBox

**wxRadioBox()**

Default constructor.

**wxRadioBox(wxWindow\* parent, wxWindowID id, const wxString& label, const**

```
wxPoint& point = wxDefaultPosition, const wxSize& size = wxDefaultSize, int n = 0,  
const wxString choices[] = NULL, int majorDimension = 0, long style =  
wxRA_SPECIFY_COLS, const wxValidator& validator = wxDefaultValidator, const  
wxString& name = "radioBox")
```

```
wxRadioBox(wxWindow* parent, wxWindowID id, const wxString& label, const  
wxPoint& point, const wxSize& size, const wxArrayString& choices, int  
majorDimension = 0, long style = wxRA_SPECIFY_COLS, const wxValidator& validator  
= wxDefaultValidator, const wxString& name = "radioBox")
```

Constructor, creating and showing a radiobox.

### Parameters

*parent*

Parent window. Must not be NULL.

*id*

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

*label*

Label for the static box surrounding the radio buttons.

*pos*

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

*size*

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

*n*

Number of choices with which to initialize the radiobox.

*choices*

An array of choices with which to initialize the radiobox.

*majorDimension*

Specifies the maximum number of rows (if style contains wxRA\_SPECIFY\_ROWS) or columns (if style contains wxRA\_SPECIFY\_COLS) for a two-dimensional radiobox.

*style*

Window style. See *wxRadioBox* (p. 1241).

*validator*

Window validator.

*name*

Window name.

### See also

`wxRadioBox::Create` (p. 1243), `wxValidator` (p. 1767)

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

**wxPerl note:** In wxPerl there is just an array reference in place of `nand choices`.

### `wxRadioBox::~wxRadioBox`

`~wxRadioBox()`

Destructor, destroying the radiobox item.

### `wxRadioBox::Create`

**bool** `Create`(**wxWindow\*** *parent*, **wxWindowID** *id*, **const wxString&** *label*, **const wxPoint&** *point* = `wxDefaultPosition`, **const wxSize&** *size* = `wxDefaultSize`, **int** *n* = 0, **const wxString** *choices*[] = `NULL`, **int** *majorDimension* = 0, **long** *style* = `wxRA_SPECIFY_COLS`, **const wxValidator&** *validator* = `wxDefaultValidator`, **const wxString&** *name* = `"radioBox"`)

**bool** `Create`(**wxWindow\*** *parent*, **wxWindowID** *id*, **const wxString&** *label*, **const wxPoint&** *point*, **const wxSize&** *size*, **const wxArrayString&** *choices*, **int** *majorDimension* = 0, **long** *style* = `wxRA_SPECIFY_COLS`, **const wxValidator&** *validator* = `wxDefaultValidator`, **const wxString&** *name* = `"radioBox"`)

Creates the radiobox for two-step construction. See `wxRadioBox::wxRadioBox` (p. 1241) for further details.

### `wxRadioBox::Enable`

**virtual bool** `Enable`(**bool** *enable* = `true`)

Enables or disables the entire radiobox.

**virtual bool** `Enable`(**unsigned int** *n*, **bool** *enable* = `true`)

Enables or disables an individual button in the radiobox.

### Parameters

*enable*

true to enable, false to disable.

*n*

The zero-based button to enable or disable.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

<b>Enable(flag)</b>	Enables or disables the entire radiobox.
<b>EnableItem(n, flag)</b>	Enables or disables an individual button in the radiobox.

**See also**

*wxWindow::Enable* (p. 1806)

**wxRadioBox::FindString**

**int FindString(const wxString& *string*) const**

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

**Parameters**

*string*

The string to find.

**wxRadioBox::GetColumnCount**

**unsigned int GetColumnCount() const**

Returns the number of columns in the radiobox.

**wxRadioBox::GetItemHelpText**

**wxString GetItemHelpText(unsigned int *item*) const**

Returns the helptext associated with the specified *item* if any or `wxEmptyString`.

**Parameters**

*item*

The zero-based item index.

**See also**

*SetItemHelpText* (p. 1247)

**wxRadioBox::GetItemToolTip**

**wxToolTip \* GetItemToolTip(unsigned int *item*) const**

Returns the tooltip associated with the specified *item* if any or `NULL`.

**See also**

*SetItemToolTip* (p. 1248),  
*wxWindow::GetToolTip* (p. 1820)

**wxRadioBox::GetItemFromPoint**

**int GetItemFromPoint(const wxPoint& pt) const**

Returns a radio box item under the point, a zero-based item index, or `wxNOT_FOUND` if no item is under the point.

*pt*

Point in client coordinates.

**wxRadioBox::GetLabel**

**wxString GetLabel() const**

Returns the radiobox label.

**Parameters**

*n*

The zero-based button index.

**See also**

*wxRadioBox::SetLabel* (p. 1247)

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

<b>GetLabel()</b>	Returns the radiobox label.
<b>GetItemLabel(n)</b>	Returns the label for the given button.

**wxRadioBox::GetRowCount**

**unsigned int GetRowCount() const**

Returns the number of rows in the radiobox.

**wxRadioBox::GetSelection**

**int GetSelection() const**

Returns the zero-based position of the selected button.

**wxRadioBox::GetStringSelection****wxString GetStringSelection() const**

Returns the selected string.

**wxRadioBox::GetString****wxString GetString(unsigned int *n*) const**

Returns the label for the button at the given position.

**Parameters***n*

The zero-based button position.

**wxRadioBox::IsItemEnabled****bool IsItemEnabled(unsigned int *n*) const**

Returns `true` if the item is enabled or `false` if it was disabled using *Enable(*n*, false)* (p. 1243).

**Platform note:** Currently only implemented in wxMSW, wxGTK and wxUniversal and always returns `true` in the other ports.

**Parameters***n*

The zero-based button position.

**wxRadioBox::IsItemShown****bool IsItemShown(unsigned int *n*) const**

Returns `true` if the item is currently shown or `false` if it was hidden using *Show(*n*, false)* (p. 1248).

Note that this function returns `true` for an item which hadn't been hidden even if the entire radiobox is not currently shown.

**Platform note:** Currently only implemented in wxMSW, wxGTK and wxUniversal and always returns `true` in the other ports.

**Parameters***n*

The zero-based button position.

### **wxRadioBox::SetItemHelpText**

**void SetItemHelpText(unsigned int *item*, const wxString& *helptext*)**

Sets the helptext for an item. Empty string erases any existing helptext.

#### **Parameters**

*item*

The zero-based item index.

*helptext*

The help text to set for the item.

#### **See also**

*GetItemHelpText* (p. 1244)

### **wxRadioBox::SetLabel**

**void SetLabel(const wxString& *label*)**

Sets the radiobox label.

#### **Parameters**

*label*

The label to set.

*n*

The zero-based button index.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

<b>SetLabel(string)</b>	Sets the radiobox label.
<b>SetItemLabel(n, string)</b>	Sets a label for a radio button.

### **wxRadioBox::SetSelection**

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

Sets a button by passing the desired string position. This does not cause a wxEVT\_COMMAND\_RADIOBOX\_SELECTED event to get emitted.

#### **Parameters**

*n*

The zero-based button position.

### **wxRadioBox::SetStringSelection**

**void SetStringSelection(const wxString& *string*)**

Sets the selection to a button by passing the desired string. This does not cause a `wxEVT_COMMAND_RADIOBOX_SELECTED` event to get emitted.

#### **Parameters**

*string*

The label of the button to select.

### **wxRadioBox::SetItemToolTip**

**void SetItemToolTip(unsigned int *item*, const wxString& *text*)**

Sets the tooltip text for the specified item in the radio group.

**Platform note:** Currently only implemented in `wxMSW` and `wxGTK2` and does nothing in the other ports.

#### **Parameters**

*item*

Index of the item the tooltip will be shown for.

*text*

Tooltip text for the item, the tooltip is removed if empty.

#### **See also**

*GetItemToolTip* (p. 1244),  
*wxWindow::SetToolTip* (p. 1847)

### **wxRadioBox::Show**

**virtual bool Show(const bool *show* = *true*)**

Shows or hides the entire radiobox.

**virtual bool Show(unsigned int *item*, const bool *show* = *true*)**

Shows or hides individual buttons.

#### **Parameters**

*show*



true to show, false to hide.

*item*

The zero-based position of the button to show or hide.

### Return value

true if the box or item has been shown or hidden or false if nothing was done because it already was in the requested state.

### See also

*wxWindow::Show* (p. 1850)

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

<b>Show(flag)</b>	Shows or hides the entire radiobox.
<b>ShowItem(n, flag)</b>	Shows or hides individual buttons.

## wxRadioButton

A radio button item is a button which usually denotes one of several mutually exclusive options. It has a text label next to a (usually) round button.

You can create a group of mutually-exclusive radio buttons by specifying `wxRB_GROUP` for the first in the group. The group ends when another radio button group is created, or there are no more radio buttons.

### Derived from

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/radiobut.h>

### Window styles

<b>wxRB_GROUP</b>	Marks the beginning of a new group of radio buttons.
<b>wxRB_SINGLE</b>	In some circumstances, radio buttons that are not consecutive siblings trigger a hang bug in Windows (only). If this happens, add this style to mark the button as not belonging to a group, and implement the mutually-exclusive group behaviour yourself.

**wxB\_USE\_CHECKBOX** Use a checkbox button instead of radio button (currently supported only on PalmOS).

See also *window styles overview* (p. 2089).

### Event handling

**EVT\_RADIOBUTTON(id, func)** Process a `wxEVT_COMMAND_RADIOBUTTON_SELECTED` event, when the radiobutton is clicked.

### See also

*Event handling overview* (p. 2077), *wxRadioButton* (p. 1241), *wxCheckBox* (p. 180)

## **wxRadioButton::wxRadioButton**

### **wxRadioButton()**

Default constructor.

**wxRadioButton(wxWindow\* parent, wxWindowID id, const wxString& label, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0, const wxValidator& validator = wxDefaultValidator, const wxString& name = "radioButton")**

Constructor, creating and showing a radio button.

### Parameters

*parent*

Parent window. Must not be NULL.

*id*

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

*label*

Label for the radio button.

*pos*

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

*size*

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

*style*

Window style. See *wxRadioButton* (p. 1249).

*validator*

Window validator.

*name*

Window name.

### See also

*wxRadioButton::Create* (p. 1251), *wxValidator* (p. 1767)

## **wxRadioButton::~~wxRadioButton**

**void ~wxRadioButton()**

Destructor, destroying the radio button item.

## **wxRadioButton::Create**

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

Creates the choice for two-step construction. See *wxRadioButton::wxRadioButton* (p. 1250) for further details.

## **wxRadioButton::GetValue**

**bool GetValue() const**

Returns true if the radio button is depressed, false otherwise.

## **wxRadioButton::SetValue**

**void SetValue(const bool value)**

Sets the radio button to selected or deselected status. This does not cause a *wxEVT\_COMMAND\_RADIOBUTTON\_SELECTED* event to get emitted.

### Parameters

*value*

true to select, false to deselect.

## **wxRealPoint**

A **wxRealPoint** is a useful data structure for graphics operations. It contains floating point *x* and *y* members. See also *wxPoint* (p. 1193) for an integer version.

**Derived from**

None

**Include files**

<wx/gdicmn.h>

**See also**

*wxPoint* (p. 1193)

**wxRealPoint::wxRealPoint**

**wxRealPoint()**

**wxRealPoint(double x, double y)**

Create a point.

**double x**

**double y**

Members of the **wxRealPoint** object.

**wxRect**

A class for manipulating rectangles.

**Derived from**

None

**Include files**

<wx/gdicmn.h>

**See also**

*wxPoint* (p. 1193), *wxSize* (p. 1440)

**wxRect::wxRect**

**wxRect()**

Default constructor.

**wxRect(int x, int y, int width, int height)**

Creates a `wxRect` object from `x`, `y`, width and height values.

**`wxRect(const wxPoint& topLeft, const wxPoint& bottomRight)`**

Creates a `wxRect` object from top-left and bottom-right points.

**`wxRect(const wxPoint& pos, const wxSize& size)`**

Creates a `wxRect` object from position and size values.

**`wxRect(const wxSize& size)`**

Creates a `wxRect` object from size values at the origin.

**`wxRect::x`**

**`int x`**

x coordinate of the top-level corner of the rectangle.

**`wxRect::y`**

**`int y`**

y coordinate of the top-level corner of the rectangle.

**`wxRect::width`**

**`int width`**

Width member.

**`wxRect::height`**

**`int height`**

Height member.

**`wxRect::CentreIn`**

**`wxRect CentreIn(const wxRect& r, int dir = wxBOTH) const`**

**`wxRect CenterIn(const wxRect& r, int dir = wxBOTH) const`**

Returns the rectangle having the same size as this one but centered relatively to the given rectangle `r`. By default, rectangle is centred in both directions but if `dir` includes only `wxVERTICAL` or only `wxHORIZONTAL` flag, then it is only centered in this direction while the other component of its position remains unchanged.

**`wxRect::Contains`**

**bool Contains(int x, int y) const**

**bool Contains(const wxPoint& pt) const**

Returns `true` if the given point is inside the rectangle (or on its boundary) and `false` otherwise.

**bool Contains(const wxRect& rect) const**

Returns `true` if the given rectangle is completely inside this rectangle (or touches its boundary) and `false` otherwise.

**wxRect::Deflate**

**void Deflate(wxCoord dx, wxCoord dy)**

**void Deflate(const wxSize& diff)**

**void Deflate(wxCoord diff)**

**wxRect Deflate(wxCoord dx, wxCoord dy) const**

Decrease the rectangle size.

This method is the opposite from *Inflate* (p. 1256): *Deflate*(a, b) is equivalent to *Inflate*(-a, -b). Please refer to *Inflate* (p. 1256) for full description.

**See also**

*Inflate* (p. 1256)

**wxRect::GetBottom**

**int GetBottom() const**

Gets the bottom point of the rectangle.

**wxRect::GetHeight**

**int GetHeight() const**

Gets the height member.

**wxRect::GetLeft**

**int GetLeft() const**

Gets the left point of the rectangle (the same as *wxRect::GetX* (p. 1256)).

**wxRect::GetPosition**

**wxPoint GetPosition() const**

Gets the position.

**wxRect::GetTopLeft****wxPoint GetTopLeft() const**

Gets the position of the top left corner of the rectangle, same as *GetPosition* (p. 1254).

**wxRect::GetTopRight****wxPoint GetTopRight() const**

Gets the position of the top right corner.

**wxRect::GetBottomLeft****wxPoint GetBottomLeft() const**

Gets the position of the bottom left corner.

**wxRect::GetBottomRight****wxPoint GetBottomRight() const**

Gets the position of the bottom right corner.

**wxRect::GetRight****int GetRight() const**

Gets the right point of the rectangle.

**wxRect::GetSize****wxSize GetSize() const**

Gets the size.

**See also**

*wxRect::SetSize* (p. 1257)

**wxRect::GetTop****int GetTop() const**

Gets the top point of the rectangle (the same as *wxRect::GetY* (p. 1256)).

**wxRect::GetWidth**

**int GetWidth() const**

Gets the width member.

**wxRect::GetX**

**int GetX() const**

Gets the x member.

**wxRect::GetY**

**int GetY() const**

Gets the y member.

**wxRect::Inflate**

**void Inflate(wxCoord dx, wxCoord dy)**

**void Inflate(const wxSize& diff)**

**void Inflate(wxCoord diff)**

**wxRect Inflate(wxCoord dx, wxCoord dy) const**

Increases the size of the rectangle.

The second form uses the same *diff* for both *dx* and *dy*.

The first two versions modify the rectangle in place, the last one returns a new rectangle leaving this one unchanged.

The left border is moved farther left and the right border is moved farther right by *dx*. The upper border is moved farther up and the bottom border is moved farther down by *dy*. (Note the the width and height of the rectangle thus change by  $2 * dx$  and  $2 * dy$ , respectively.) If one or both of *dx* and *dy* are negative, the opposite happens: the rectangle size decreases in the respective direction.

Inflating and deflating behaves "naturally". Defined more precisely, that means:

1. "Real" inflates (that is, *dx* and/or *dy*  $\geq 0$ ) are not constrained. Thus inflating a rectangle can cause its upper left corner to move into the negative numbers. (the versions prior to 2.5.4 forced the top left coordinate to not fall below (0, 0), which implied a forced move of the rectangle.)
2. Deflates are clamped to not reduce the width or height of the rectangle below zero. In such cases, the top-left corner is nonetheless handled properly. For example, a rectangle at (10, 10) with size (20, 40) that is inflated by (-15, -15) will become located at (20, 25) at size (0, 10). Finally, observe that the width and height are treated independently. In the above example, the width is



reduced by 20, whereas the height is reduced by the full 30 (rather than also stopping at 20, when the width reached zero).

**See also**

*Deflate* (p. 1254)

**wxRect::Intersect**

**wxRect Intersect(const wxRect& rect) const**

**wxRect& Intersect(const wxRect& rect)**

Modifies the rectangle to contain the overlapping box of this rectangle and the one passed in as parameter. The const version returns the new rectangle, the other one modifies this rectangle in place.

**wxRect::Intersects**

**bool Intersects(const wxRect& rect) const**

Returns `true` if this rectangle has a non-empty intersection with the rectangle *rect* and `false` otherwise.

**wxRect::IsEmpty**

**bool IsEmpty() const**

Returns `true` if this rectangle has a width or height less than or equal to 0 and `false` otherwise.

**wxRect::Offset**

**void Offset(wxCoord dx, wxCoord dy)**

**void Offset(const wxPoint& pt)**

Moves the rectangle by the specified offset. If *dx* is positive, the rectangle is moved to the right, if *dy* is positive, it is moved to the bottom, otherwise it is moved to the left or top respectively.

**wxRect::SetHeight**

**void SetHeight(int height)**

Sets the height.

**wxRect::SetSize**

**void SetSize(const wxSize& s)**

Sets the size.

**See also**

*wxRect::GetSize* (p. 1255)

**wxRect::SetWidth**

**void SetWidth(int width)**

Sets the width.

**wxRect::SetX**

**void SetX(int x)**

Sets the x position.

**wxRect::SetY**

**void SetY(int y)**

Sets the y position.

**wxRect::Union**

**wxRect Union(const wxRect& rect) const**

**wxRect& Union(const wxRect& rect)**

Modifies the rectangle to contain the bounding box of this rectangle and the one passed in as parameter. The const version returns the new rectangle, the other one modifies this rectangle in place.

**wxRect::operator =**

**void operator =(const wxRect& rect)**

Assignment operator.

**wxRect::operator ==**

**bool operator ==(const wxRect& rect)**

Equality operator.

**wxRect::operator !=**

**bool operator !=(const wxRect& rect)**

Inequality operator.

## wxRecursionGuard

wxRecursionGuard is a very simple class which can be used to prevent reentrancy problems in a function. It is not thread-safe and so should be used only in single-threaded programs or in combination with some thread synchronization mechanisms.

wxRecursionGuard is always used together with the *wxRecursionGuardFlag* (p. 1260) like in this example:

```
void Foo()
{
    static wxRecursionGuardFlag s_flag;
    wxRecursionGuard guard(s_flag);
    if ( guard.IsInside() )
    {
        // don't allow reentrancy
        return;
    }
    ...
}
```

As you can see, wxRecursionGuard simply tests the flag value and sets it to true if it hadn't been already set. *IsInside()* (p. 1260) allows testing the old flag value. The advantage of using this class compared to directly manipulating the flag is that the flag is always reset in the wxRecursionGuard destructor and so you don't risk to forget to do it even if the function returns in an unexpected way (for example because an exception has been thrown).

### Derived from

No base class

### Include files

<wx/recguard.h>

### wxRecursionGuard::wxRecursionGuard

**wxRecursionGuard(wxRecursionGuardFlag& flag)**

A wxRecursionGuard object must always be initialized with a (static) *wxRecursionGuardFlag* (p. 1260). The constructor saves the value of the flag to be able to return the correct value from *IsInside* (p. 1260).

### wxRecursionGuard::~~wxRecursionGuard

**~wxRecursionGuard()**

The destructor resets the flag value so that the function can be entered again the next time.

Note that it is not virtual and so this class is not meant to be derived from (besides, there is

absolutely no reason to do it anyhow).

### **wxRecursionGuard::IsInside**

#### **bool IsInside() const**

Returns `true` if we're already inside the code block "protected" by this `wxRecursionGuard` (i.e. between this line and the end of current scope). Usually the function using `wxRecursionGuard` takes some specific actions in such case (may be simply returning) to prevent reentrant calls to itself.

If this method returns `false`, it is safe to continue.

### **wxRecursionGuardFlag**

This is a completely opaque class which exists only to be used with `wxRecursionGuard` (p. 1259), please see the example in that class documentation.

Please notice that `wxRecursionGuardFlag` object *must* be declared `static` or the recursion would never be detected.

#### **Derived from**

No base class

#### **Include files**

<wx/recguard.h>

### **wxRegEx**

`wxRegEx` represents a regular expression. This class provides support for regular expressions matching and also replacement.

It is built on top of either the system library (if it has support for POSIX regular expressions - which is the case of the most modern Unices) or uses the built in Henry Spencer's library. Henry Spencer would appreciate being given credit in the documentation of software which uses his library, but that is not a requirement.

Regular expressions, as defined by POSIX, come in two flavours: *extended* and *basic*. The builtin library also adds a third flavour of expression *advanced* (p. 2210), which is not available when using the system library.

Unicode is fully supported only when using the builtin library. When using the system library in Unicode mode, the expressions and data are translated to the default 8-bit encoding before being passed to the library.

On platforms where a system library is available, the default is to use the builtin library for Unicode builds, and the system library otherwise. It is possible to use the other if preferred by selecting it when building the `wxWidgets`.

**Derived from**

No base class

**Data structures**

Flags for regex compilation to be used with *Compile()* (p. 1262):

```
enum
{
    // use extended regex syntax
    wxRE_EXTENDED = 0,

    // use advanced RE syntax (built-in regex only)
#ifdef wxHAS_REGEX_ADVANCED
    wxRE_ADVANCED = 1,
#endif

    // use basic RE syntax
    wxRE_BASIC = 2,

    // ignore case in match
    wxRE_ICASE = 4,

    // only check match, don't set back references
    wxRE_NOSUB = 8,

    // if not set, treat '\n' as an ordinary character, otherwise it
    // is
    // special: it is not matched by '.' and '^' and '$' always match
    // after/before it regardless of the setting of wxRE_NOT[BE]OL
    wxRE_NEWLINE = 16,

    // default flags
    wxRE_DEFAULT = wxRE_EXTENDED
}
```

Flags for regex matching to be used with *Matches()* (p. 1263).

These flags are mainly useful when doing several matches in a long string to prevent erroneous matches for ' ' and '\$':

```
enum
{
    // '^' doesn't match at the start of line
    wxRE_NOTBOL = 32,

    // '$' doesn't match at the end of line
    wxRE_NOTEOL = 64
}
```

**Examples**

A bad example of processing some text containing email addresses (the example is bad because the real email addresses can have more complicated form than `user@host.net`):

```
wxString text;
...
wxRegex reEmail = wxT("([^\@]+)@([[:alnum:]\._-]+)([[:alnum:]]+);");
if ( reEmail.Matches(text) )
{
    wxString text = reEmail.GetMatch(email);
    wxString username = reEmail.GetMatch(email, 1);
    if ( reEmail.GetMatch(email, 3) == wxT("com") ) // .com TLD?
    {
        ...
    }
}

// or we could do this to hide the email address
size_t count = reEmail.ReplaceAll(text, wxT("HIDDEN@\\2\\3"));
printf("text now contains %u hidden addresses", count);
```

### Include files

<wx/regex.h>

### wxRegex::wxRegex

**wxRegex()**

Default ctor: use *Compile()* (p. 1262) later.

**wxRegex(const wxString& expr, int flags = wxRE\_DEFAULT)**

Create and compile the regular expression, use *IsValid* (p. 1262) to test for compilation errors.

### wxRegex::~~wxRegex

**~wxRegex()**

dtor not virtual, don't derive from this class

### wxRegex::Compile

**bool Compile(const wxString& pattern, int flags = wxRE\_DEFAULT)**

Compile the string into regular expression, return `true` if ok or `false` if string has a syntax error.

### wxRegex::IsValid

**bool IsValid() const**

Return `true` if this is a valid compiled regular expression, `false` otherwise.

**wxRegex::GetMatch****bool GetMatch(size\_t\* start, size\_t\* len, size\_t index = 0) const**

Get the start index and the length of the match of the expression (if *index* is 0) or a bracketed subexpression (*index* different from 0).

May only be called after successful call to *Matches()* (p. 1263) and only if `wxRE_NOSUB` was **not** used in *Compile()* (p. 1262).

Returns `false` if no match or if an error occurred.

**wxString GetMatch(const wxString& text, size\_t index = 0) const**

Returns the part of string corresponding to the match where *index* is interpreted as above. Empty string is returned if match failed

May only be called after successful call to *Matches()* (p. 1263) and only if `wxRE_NOSUB` was **not** used in *Compile()* (p. 1262).

**wxRegex::GetMatchCount****size\_t GetMatchCount() const**

Returns the size of the array of matches, i.e. the number of bracketed subexpressions plus one for the expression itself, or 0 on error.

May only be called after successful call to *Compile()* (p. 1262). and only if `wxRE_NOSUB` was **not** used.

**wxRegex::Matches****bool Matches(const wxChar\* text, int flags = 0) const****bool Matches(const wxChar\* text, int flags, size\_t len) const****bool Matches(const wxString& text, int flags = 0) const**

Matches the precompiled regular expression against the string *text*, returns `true` if matches and `false` otherwise.

*Flags* may be combination of `wxRE_NOTBOL` and `wxRE_NOTEOL`.

Some regex libraries assume that the text given is null terminated, while others require the length be given as a separate parameter. Therefore for maximum portability assume that *text* cannot contain embedded nulls.

When the *Matches(const wxChar \*text, int flags = 0)* form is used, a *wxStrlen()* will be done internally if the regex library requires the length. When using *Matches()* in a loop the *Matches(text, flags, len)* form can be used instead, making it possible to avoid a *wxStrlen()* inside the loop.

May only be called after successful call to *Compile()* (p. 1262).

### **wxRegEx::Replace**

**int Replace(wxString\* text, const wxString& replacement, size\_t maxMatches = 0) const**

Replaces the current regular expression in the string pointed to by *text*, with the text in *replacement* and return number of matches replaced (maybe 0 if none found) or -1 on error.

The replacement text may contain back references `\number` which will be replaced with the value of the corresponding subexpression in the pattern match. `\0` corresponds to the entire match and `&` is a synonym for it. Backslash may be used to quote itself or `&` character.

*maxMatches* may be used to limit the number of replacements made, setting it to 1, for example, will only replace first occurrence (if any) of the pattern in the text while default value of 0 means replace all.

### **wxRegEx::ReplaceAll**

**int ReplaceAll(wxString\* text, const wxString& replacement) const**

Replace all occurrences: this is actually a synonym for *Replace()* (p. 1264).

#### **See also**

*ReplaceFirst* (p. 1264)

### **wxRegEx::ReplaceFirst**

**int ReplaceFirst(wxString\* text, const wxString& replacement) const**

Replace the first occurrence.

#### **See also**

*Replace* (p. 1264)

## **wxRegion**

A *wxRegion* represents a simple or complex region on a device context or window.

This class uses *reference counting and copy-on-write* (p. 2046) internally so that assignments between two instances of this class are very cheap. You can therefore use actual objects instead of pointers without efficiency problems. If an instance of this class is changed it will create its own data internally so that other instances, which previously shared the data using the reference counting, are not affected.

#### **Derived from**

*wxGDIObject* (p. 709)



*wxObject* (p. 1148)

#### **Include files**

<wx/region.h>

#### **See also**

*wxRegionIterator* (p. 1269)

### **wxRegion::wxRegion**

#### **wxRegion()**

Default constructor.

#### **wxRegion(wxCoord x, wxCoord y, wxCoord width, wxCoord height)**

Constructs a rectangular region with the given position and size.

#### **wxRegion(const wxPoint& topLeft, const wxPoint& bottomRight)**

Constructs a rectangular region from the top left point and the bottom right point.

#### **wxRegion(const wxRect& rect)**

Constructs a rectangular region a wxRect object.

#### **wxRegion(const wxRegion& region)**

Copy constructor, uses *reference counting* (p. 2046).

#### **wxRegion(size\_t n, const wxPoint \*points, int fillStyle = wxWINDING\_RULE)**

Constructs a region corresponding to the polygon made of *n* points in the provided array. *fillStyle* parameter may have values `wxWINDING_RULE` or `wxODDEVEN_RULE`.

#### **wxRegion(const wxBitmap& bmp)**

**wxRegion(const wxBitmap& bmp, const wxColour& transColour, int tolerance = 0)**

Constructs a region using the non-transparent pixels of a bitmap. See *Union* (p. 1268) for more details.

### **wxRegion::~~wxRegion**

#### **~wxRegion()**

Destructor. See *reference-counted object destruction* (p. 2046) for more info.

### **wxRegion::Clear**

**void Clear()**

Clears the current region.

**wxRegion::Contains****wxRegionContain Contains(long& x, long& y) const**

Returns a value indicating whether the given point is contained within the region.

**wxRegionContain Contains(const wxPoint& pt) const**

Returns a value indicating whether the given point is contained within the region.

**wxRegionContain Contains(long& x, long& y, long& width, long& height) const**

Returns a value indicating whether the given rectangle is contained within the region.

**wxRegionContain Contains(const wxRect& rect) const**

Returns a value indicating whether the given rectangle is contained within the region.

**Return value**

The return value is one of wxOutRegion, wxPartRegion and wxInRegion.

On Windows, only wxOutRegion and wxInRegion are returned; a value wxInRegion then indicates that all or some part of the region is contained in this region.

**wxRegion::ConvertToBitmap****wxBitmap ConvertToBitmap() const**

Convert the region to a black and white bitmap with the white pixels being inside the region.

**wxRegion::GetBox****void GetBox(wxCoord& x, wxCoord& y, wxCoord& width, wxCoord& height) const**

Returns the outer bounds of the region.

**wxRect GetBox() const**

Returns the outer bounds of the region.

**wxRegion::Intersect****bool Intersect(wxCoord x, wxCoord y, wxCoord width, wxCoord height)**

Finds the intersection of this region and another, rectangular region, specified using position and size.

**bool Intersect(const wxRect& *rect*)**

Finds the intersection of this region and another, rectangular region.

**bool Intersect(const wxRegion& *region*)**

Finds the intersection of this region and another region.

**Return value**

`true` if successful, `false` otherwise.

**Remarks**

Creates the intersection of the two regions, that is, the parts which are in both regions. The result is stored in this region.

**wxRegion::IsEmpty****bool IsEmpty() const**

Returns `true` if the region is empty, `false` otherwise.

**wxRegion::IsEqual****bool IsEqual(const wxRegion& *region*) const**

Returns `true` if the region is equal to, i.e. covers the same area as, another one. Note that if both this region and *region* are invalid, they are considered to be equal.

**wxRegion::Subtract****bool Subtract(const wxRect& *rect*)**

Subtracts a rectangular region from this region.

**bool Subtract(const wxRegion& *region*)**

Subtracts a region from this region.

**Return value**

`true` if successful, `false` otherwise.

**Remarks**

This operation combines the parts of 'this' region that are not part of the second region. The result is stored in this region.

**wxRegion::Offset****bool Offset(wxCoord *x*, wxCoord *y*)**

**bool Offset(const wxPoint& pt)**

Moves the region by the specified offsets in horizontal and vertical directions.

**Return value**

`true` if successful, `false` otherwise (the region is unchanged then).

**wxRegion::Union**

**bool Union(wxCoord x, wxCoord y, wxCoord width, wxCoord height)**

Finds the union of this region and another, rectangular region, specified using position and size.

**bool Union(const wxRect& rect)**

Finds the union of this region and another, rectangular region.

**bool Union(const wxRegion& region)**

Finds the union of this region and another region.

**bool Union(const wxBitmap& bmp)**

Finds the union of this region and the non-transparent pixels of a bitmap. Bitmap's mask is used to determine transparency. If the bitmap doesn't have a mask, solid rectangle of bitmap's dimensions is used.

**bool Union(const wxBitmap& bmp, const wxColour& transColour, int tolerance = 0)**

Finds the union of this region and the non-transparent pixels of a bitmap. Colour to be treated as transparent is specified in the `transColour` argument, along with an optional colour tolerance value.

**Return value**

`true` if successful, `false` otherwise.

**Remarks**

This operation creates a region that combines all of this region and the second region. The result is stored in this region.

**wxRegion::Xor**

**bool Xor(wxCoord x, wxCoord y, wxCoord width, wxCoord height)**

Finds the Xor of this region and another, rectangular region, specified using position and size.

**bool Xor(const wxRect& rect)**

Finds the Xor of this region and another, rectangular region.

**bool Xor(const wxRegion& region)**

Finds the Xor of this region and another region.

**Return value**

true if successful, false otherwise.

**Remarks**

This operation creates a region that combines all of this region and the second region, except for any overlapping areas. The result is stored in this region.

**wxRegion::operator =**

**void operator =(const wxRegion& region)**

Assignment operator, using *reference counting* (p. 2046).

## wxRegionIterator

This class is used to iterate through the rectangles in a region, typically when examining the damaged regions of a window within an OnPaint call.

To use it, construct an iterator object on the stack and loop through the regions, testing the object and incrementing the iterator at the end of the loop.

See *wxPaintEvent* (p. 1164) for an example of use.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/region.h>

**See also**

*wxPaintEvent* (p. 1164)

**wxRegionIterator::wxRegionIterator**

**wxRegionIterator()**

Default constructor.

**wxRegionIterator(const wxRegion& region)**

Creates an iterator object given a region.

**wxRegionIterator::GetX****wxCoord GetX() const**

Returns the x value for the current region.

**wxRegionIterator::GetY****wxCoord GetY() const**

Returns the y value for the current region.

**wxRegionIterator::GetW****wxCoord GetW() const**

An alias for GetWidth.

**wxRegionIterator::GetHeight****wxCoord GetHeight() const**

Returns the height value for the current region.

**wxRegionIterator::GetH****wxCoord GetH() const**

An alias for GetHeight.

**wxRegionIterator::GetRect****wxRect GetRect() const**

Returns the current rectangle.

**wxRegionIterator::GetWidth****wxCoord GetWidth() const**

Returns the width value for the current region.

**wxRegionIterator::HaveRects****bool HaveRects() const**

Returns `true` if there are still some rectangles; otherwise returns `false`.

**wxRegionIterator::Reset****void Reset()**

Resets the iterator to the beginning of the rectangles.

**void Reset(const wxRegion& region)**

Resets the iterator to the given region.

**wxRegionIterator::operator ++****void operator ++()**

Increment operator. Increments the iterator to the next region.

**wxPython note:** A wxPython alias for this operator is called `Next`.

**wxRegionIterator::operator bool****operator bool() const**

Returns `true` if there are still some rectangles; otherwise returns `false`.

You can use this to test the iterator object as if it were of type `bool`.

**wxRegKey**

`wxRegKey` is a class representing the Windows registry (it is only available under Windows). One can create, query and delete registry keys using this class.

The Windows registry is easy to understand. There are five registry keys, namely:

1. `HKEY_CLASSES_ROOT` (HKCR)
2. `HKEY_CURRENT_USER` (HKCU)
3. `HKEY_LOCAL_MACHINE` (HKLM)
4. `HKEY_CURRENT_CONFIG` (HKCC)
5. `HKEY_USERS` (HKU)

After creating a key, it can hold a value. The values can be:

1. String Value
2. Binary Value
3. DWORD Value
4. Multi String Value

## 5. Expandable String Value

### Derived from

None

### Include files

<wx/msw/registry.h>

### Example

```
wxRegKey *pRegKey = new
wxRegKey( "HKEY_LOCAL_MACHINE\\Software\\MyKey" );

//will create the Key if it does not exist
if( !pRegKey->Exists() )
    pRegKey->Create();

//will create a new value MYVALUE and set it to 12
pRegKey->SetValue( "MYVALUE",12);

//Query for the Value and Retrieve it
long lMyVal;
wxString strTemp;
pRegKey->QueryValue( "MYVALUE",&lMyVal);
strTemp.Printf( "%d",lMyVal);
wxMessageBox( strTemp, "Registry Value",0,this);

//Retrive the number of SubKeys and enumerate them
size_t nSubKeys;
pRegKey->GetKeyInfo( &nSubKeys, NULL, NULL, NULL );

pRegKey->GetFirstKey( strTemp,1);
for(int i=0;i<nSubKeys;i++)
{
    wxMessageBox( strTemp, "SubKey Name",0,this);
    pRegKey->GetNextKey( strTemp,1);
}
```

### **wxRegKey::wxRegKey**

#### **wxRegKey()**

The Constructor to set to HKCR

#### **wxRegKey(const wxString& strKey)**

The constructor to set the full name of the key.

#### **wxRegKey(const wxRegKey& keyParent, const wxString& strKey)**

The constructor to set the full name of the key under a previously created parent.



**wxRegKey::Close****void Close()**

Closes the key.

**wxRegKey::Create****bool Create(bool bOkIfExists = true)**

Creates the key. Will fail if the key already exists and *bOkIfExists* is false.

**wxRegKey::DeleteSelf****void DeleteSelf()**

Deletes this key and all of its subkeys and values recursively.

**wxRegKey::DeleteKey****void DeleteKey(const wxChar \*szKey)**

Deletes the subkey with all of its subkeys/values recursively.

**wxRegKey::DeleteValue****void DeleteValue(const wxChar \*szKey)**

Deletes the named value.

**wxRegKey::Exists****static bool Exists() const**

Returns true if the key exists.

**wxRegKey::GetName****wxString GetName(bool bShortPrefix = true) const**

Gets the name of the registry key.

**wxRegKey::GetFirstKey****bool GetFirstKey(wxString& strKeyName, long& lIndex)**

Gets the first key.

**wxRegKey::GetFirstValue**

**bool GetFirstValue(wxString& strValueName, long& lIndex)**

Gets the first value of this key.

**wxRegKey::GetKeyInfo**

**bool Exists(size\_t \*pnSubKeys, size\_t \*pnValues, size\_t \*pnMaxValueLen) const**

Gets information about the key.

**Parameters**

*pnSubKeys*

The number of subkeys.

*pnMaxKeyLen*

The maximum length of the subkey name.

*pnValues*

The number of values.

**wxRegKey::GetNextKey**

**bool GetNextKey(wxString& strKeyName, long& lIndex) const**

Gets the next key.

**wxRegKey::GetNextValue**

**bool GetNextValue(wxString& strValueName, long& lIndex) const**

Gets the next key value for this key.

**wxRegKey::HasValue**

**bool HasValue(const wxChar \*szValue) const**

Returns true if the value exists.

**wxRegKey::HasValues**

**bool HasValues() const**

Returns true if any values exist.

**wxRegKey::HasSubKey**

**bool HasSubKey(const wxChar \*szKey) const**

Returns true if given subkey exists.

### **wxRegKey::HasSubKeys**

**bool HasSubKeys() const**

Returns true if any subkeys exist.

### **wxRegKey::IsEmpty**

**bool IsEmpty() const**

Returns true if this key is empty, nothing under this key.

### **wxRegKey::IsOpened**

**bool IsOpened() const**

Returns true if the key is opened.

### **wxRegKey::Open**

**bool Open(AccessMode mode = Write)**

Explicitly opens the key. This method also allows the key to be opened in read-only mode by passing `wxRegKey::Read` instead of default `wxRegKey::Write` parameter.

### **wxRegKey::QueryValue**

**bool QueryValue(const wxChar \*szValue, wxString& strValue) const**

Retrieves the string value.

**bool QueryValue(const wxChar \*szValue, long \*pIValue) const**

Retrieves the numeric value.

### **wxRegKey::Rename**

**bool Rename(const wxChar \*szNewName)**

Renames the key.

### **wxRegKey::RenameValue**

**bool RenameValue(const wxChar \*szValueOld, const wxChar \*szValueNew)**

Renames a value.

### **wxRegKey::SetValue**

**bool SetValue(const wxChar \*szValue, long lValue)**

**bool SetValue(const wxChar \*szValue, const wxString& strValue)**

**bool SetValue(const wxChar \*szValue, const wxMemoryBuffer& buf)**

Sets the given *szValue* which must be numeric, string or binary depending on the overload used. If the value doesn't exist, it is created.

## wxRendererNative

First, a brief introduction to wxRenderer and why it is needed.

Usually wxWidgets uses the underlying low level GUI system to draw all the controls - this is what we mean when we say that it is a "native" framework. However not all controls exist under all (or even any) platforms and in this case wxWidgets provides a default, generic, implementation of them written in wxWidgets itself.

These controls don't have the native appearance if only the standard line drawing and other graphics primitives are used, because the native appearance is different under different platforms while the lines are always drawn in the same way.

This is why we have renderers: wxRenderer is a class which virtualizes the drawing, i.e. it abstracts the drawing operations and allows you to draw say, a button, without caring about exactly how this is done. Of course, as we can draw the button differently in different renderers, this also allows us to emulate the native look and feel.

So the renderers work by exposing a large set of high-level drawing functions which are used by the generic controls. There is always a default global renderer but it may be changed or extended by the user, see *Render sample* (p. 2037).

All drawing functions take some standard parameters:

- *win* is the window being drawn. It is normally not used and when it is it should only be used as a generic *wxWindow* (p. 1795) (in order to get its low level handle, for example), but you should*not* assume that it is of some given type as the same renderer function may be reused for drawing different kinds of control.
- *dc* is the *wxDC* (p. 456) to draw on. Only this device context should be used for drawing. It is not necessary to restore pens and brushes for it on function exit but, on the other hand, you shouldn't assume that it is in any specific state on function entry: the rendering functions should always prepare it.
- *rect* the bounding rectangle for the element to be drawn.
- *flags* the optional flags (none by default) which can be a combination of the `wxCONTROL_XXX` constants below.

Note that each drawing function restores the *wxDC* (p. 456) attributes if it changes them, so it is safe to assume that the same pen, brush and colours that were active before the call to this function are still in effect after it.

## Constants

The following rendering flags are defined:

```
enum
{
    wxCONTROL_DISABLED    = 0x00000001, // control is disabled
    wxCONTROL_FOCUSED     = 0x00000002, // currently has keyboard focus
    wxCONTROL_PRESSED      = 0x00000004, // (button) is pressed
    wxCONTROL_ISDEFAULT    = 0x00000008, // only applies to the buttons
    wxCONTROL_ISSUBMENU    = wxCONTROL_ISDEFAULT, // only for menu items
    wxCONTROL_EXPANDED     = wxCONTROL_ISDEFAULT, // only for the tree
    items
    wxCONTROL_CURRENT      = 0x00000010, // mouse is currently over the
    control
    wxCONTROL_SELECTED     = 0x00000020, // selected item in e.g.
    listbox
    wxCONTROL_CHECKED      = 0x00000040, // (check/radio button) is
    checked
    wxCONTROL_CHECKABLE    = 0x00000080, // (menu) item can be checked
    wxCONTROL_UNDETERMINED = wxCONTROL_CHECKABLE // (check)
    undetermined state
};
```

## Derived from

No base class

## Include files

<wx/renderer.h>

## wxRendererNative::~~wxRendererNative

**~wxRendererNative()**

Virtual destructor as for any base class.

## wxRendererNative::DrawCheckBox

**void DrawCheckBox(wxWindow \*win, wxDC& dc, const wxRect& rect, int flags)**

Draw a check box (used by wxDataViewCtrl).

*flags* may have the `wxCONTROL_CHECKED`, `wxCONTROL_CURRENT` or `wxCONTROL_UNDETERMINED` bit set.

## wxRendererNative::DrawComboBoxDropButton

**void DrawComboBoxDropButton(wxWindow \*win, wxDC& dc, const wxRect& rect, int flags)**

Draw a button like the one used by *wxComboBox* (p. 225) to show a drop down window. The usual appearance is a downwards pointing arrow.

*flags* may have the `wxCONTROL_PRESSED` or `wxCONTROL_CURRENT` bit set.

### **wxRendererNative::DrawDropArrow**

**void DrawDropArrow(wxWindow\* win, wxDC& dc, const wxRect& rect, int flags)**

Draw a drop down arrow that is suitable for use outside a combo box. Arrow will have transparent background.

*rect* is not entirely filled by the arrow. Instead, you should use bounding rectangle of a drop down button which arrow matches the size you need. *flags* may have the `wxCONTROL_PRESSED` or `wxCONTROL_CURRENT` bit set.

### **wxRendererNative::DrawHeaderButton**

**int DrawHeaderButton(wxWindow\* win, wxDC& dc, const wxRect& rect, int flags = 0, wxHeaderSortIconType sortArrow = wxHDR\_SORT\_ICON\_NONE, wxHeaderButtonParams\* params = NULL)**

Draw the header control button (used, for example, by *wxListCtrl* (p. 980)). Depending on platforms the *flags* parameter may support the `wxCONTROL_SELECTED`, `wxCONTROL_DISABLED` and `wxCONTROL_CURRENT` bits. The *sortArrow* parameter can be one of `wxHDR_SORT_ICON_NONE`, `wxHDR_SORT_ICON_UP`, or `wxHDR_SORT_ICON_DOWN`. Additional values controlling the drawing of a text or bitmap label can be passed in *params*. The value returned is the optimal width to contain the the unabridged label text or bitmap, the sort arrow if present, and internal margins.

### **wxRendererNative::DrawItemSelectionRect**

**void DrawItemSelectionRect(wxWindow\* win, wxDC& dc, const wxRect& rect, int flags = 0)**

Draw a selection rectangle underneath the text as used e.g. in a *wxListCtrl* (p. 980). The supported *flags* are `wxCONTROL_SELECTED` for items which are selected (e.g. often a blue rectangle) and `wxCONTROL_CURRENT` for the item that has the focus (often a dotted line around the item's text). `wxCONTROL_FOCUSED` may be used to indicate if the control has the focus (otherwise the the selection rectangle is e.g. often grey and not blue). This may be ignored by the renderer or deduced by the code directly from the *win*.

### **wxRendererNative::DrawPushButton**

**void DrawPushButton(wxWindow\* win, wxDC& dc, const wxRect& rect, int flags)**

Draw a blank push button that looks very similar to *wxButton* (p. 164).

*flags* may have the `wxCONTROL_PRESSED`, `wxCONTROL_CURRENT` or `wxCONTROL_ISDEFAULT` bit set.

**wxRendererNative::DrawSplitterBorder**

```
void DrawSplitterBorder(wxWindow* win, wxDC& dc, const wxRect& rect, int flags = 0)
```

Draw the border for sash window: this border must be such that the sash drawn by *DrawSash* (p. 1279) blends into it well.

**wxRendererNative::DrawSplitterSash**

```
void DrawSplitterSash(wxWindow* win, wxDC& dc, const wxSize& size, wxCoord position, wxOrientation orient, int flags = 0)
```

Draw a sash. The *orient* parameter defines whether the sash should be vertical or horizontal and how the *position* should be interpreted.

**wxRendererNative::DrawTreeItemButton**

```
void DrawTreeItemButton(wxWindow* win, wxDC& dc, const wxRect& rect, int flags = 0)
```

Draw the expanded/collapsed icon for a tree control item. To draw an expanded button the *flags* parameter must contain `wxCONTROL_EXPANDED` bit.

**wxRendererNative::Get**

```
wxRendererNative& Get()
```

Return the currently used renderer.

**wxRendererNative::GetDefault**

```
wxRendererNative& GetDefault()
```

Return the default (native) implementation for this platform -- this is also the one used by default but this may be changed by calling *Set* (p. 1280) in which case the return value of this method may be different from the return value of *Get* (p. 1279).

**wxRendererNative::GetGeneric**

```
wxRendererNative& GetGeneric()
```

Return the generic implementation of the renderer. Under some platforms, this is the default renderer implementation, others have platform-specific default renderer which can be retrieved by calling *GetDefault* (p. 1279).

**wxRendererNative::GetHeaderButtonHeight**

```
int GetHeaderButtonHeight(const wxWindow* win)
```

Returns the height of a header button, either a fixed platform height if available, or a generic height based on the window's font.

### **wxRendererNative::GetSplitterParams**

**wxSplitterRenderParams GetSplitterParams(const wxWindow\* win)**

Get the splitter parameters, see *wxSplitterRenderParams* (p. 1518).

### **wxRendererNative::GetVersion**

**wxRendererVersion GetVersion() const**

This function is used for version checking: *Load* (p. 1280) refuses to load any shared libraries implementing an older or incompatible version.

The implementation of this method is always the same in all renderers (simply construct *wxRendererVersion* (p. 1280) using the `wxRendererVersion::Current_XXX` values), but it has to be in the derived, not base, class, to detect mismatches between the renderers versions and so you have to implement it anew in all renderers.

### **wxRendererNative::Load**

**wxRendererNative\* Load(const wxString& name)**

Load the renderer from the specified DLL, the returned pointer must be deleted by caller if not `NULL` when it is not used any more.

The *name* should be just the base name of the renderer and not the full name of the DLL file which is constructed differently (using *wxDynamicLibrary::CanonicalizePluginName* (p. 565)) on different systems.

### **wxRendererNative::Set**

**wxRendererNative\* Set(wxRendererNative\* renderer)**

Set the renderer to use, passing `NULL` reverts to using the default renderer (the global renderer must always exist).

Return the previous renderer used with *Set()* or `NULL` if none.

## **wxRendererVersion**

This simple struct represents the *wxRendererNative* (p. 1276) interface version and is only used as the return value of *wxRendererNative::GetVersion* (p. 1280).

The version has two components: the version itself and the age. If the main program and the renderer have different versions they are never compatible with each other because the version is only changed when an existing virtual function is modified or removed. The age, on the other hand, is incremented each time a new virtual method is added and so, at



least for the compilers using a common C++ object model, the calling program is compatible with any renderer which has the age greater or equal to its age. This verification is done by *IsCompatible* (p. 1281) method.

**Derived from**

No base class

**Include files**

<wx/renderer.h>

**wxRendererVersion::IsCompatible**

**static bool IsCompatible(const wxRendererVersion& ver)**

Checks if the main program is compatible with the renderer having the version *ver*, returns `true` if it is and `false` otherwise.

This method is used by *wxRendererNative::Load* (p. 1280) to determine whether a renderer can be used.

**wxRendererVersion::version**

**const int version**

The version component.

**wxRendererVersion::age**

**const int age**

The age component.

**wxRichTextAttr**

*wxRichTextAttr* represents the character and paragraph attributes, or style, for a range of text in a *wxRichTextCtrl* (p. 1316). This class is specific to *wxRichTextCtrl*, although you can also use the standard *wxTextAttr* (p. 1618) class with *wxRichTextCtrl*.

When setting up a *wxRichTextAttr* object, pass a bitlist mask to *SetFlags* (p. 1294) to indicate which style elements should be changed. As a convenience, when you call a setter such as *SetFont*, the relevant bit will be set.

*wxRichTextAttr* stores attributes without a *wxFont* object, so is a more efficient way to query styles than using a *wxTextAttr* (p. 1618) or *wxTextAttrEx* (p. 1623) object.

**Derived from**

No base class

### Include files

<wx/richtext/richtextbuffer.h>

### Constants

The following values can be passed to `wxRichTextAttr::SetAlignment` to determine paragraph alignment.

```
enum wxTextAttrAlignment
{
    wxTEXT_ALIGNMENT_DEFAULT,
    wxTEXT_ALIGNMENT_LEFT,
    wxTEXT_ALIGNMENT_CENTRE,
    wxTEXT_ALIGNMENT_CENTER = wxTEXT_ALIGNMENT_CENTRE,
    wxTEXT_ALIGNMENT_RIGHT,
    wxTEXT_ALIGNMENT_JUSTIFIED
};
```

Of these, `wxTEXT_ALIGNMENT_JUSTIFIED` is unimplemented. In future justification may be supported when printing or previewing, only.

The following values are passed in a bitlist to `wxRichTextAttr::SetFlags` to determine what attributes will be considered when setting the attributes for a text control.

```
// Standard wxTextAttr constants

#define wxTEXT_ATTR_TEXT_COLOUR          0x00000001
#define wxTEXT_ATTR_BACKGROUND_COLOUR    0x00000002
#define wxTEXT_ATTR_FONT_FACE             0x00000004
#define wxTEXT_ATTR_FONT_SIZE            0x00000008
#define wxTEXT_ATTR_FONT_WEIGHT          0x00000010
#define wxTEXT_ATTR_FONT_ITALIC          0x00000020
#define wxTEXT_ATTR_FONT_UNDERLINE       0x00000040
#define wxTEXT_ATTR_FONT \
    wxTEXT_ATTR_FONT_FACE | wxTEXT_ATTR_FONT_SIZE | \
wxTEXT_ATTR_FONT_WEIGHT \
| wxTEXT_ATTR_FONT_ITALIC | wxTEXT_ATTR_FONT_UNDERLINE
#define wxTEXT_ATTR_ALIGNMENT            0x00000080
#define wxTEXT_ATTR_LEFT_INDENT          0x00000100
#define wxTEXT_ATTR_RIGHT_INDENT         0x00000200
#define wxTEXT_ATTR_TABS                  0x00000400

// Extra formatting flags not in wxTextAttr

#define wxTEXT_ATTR_PARA_SPACING_AFTER    0x00000800
#define wxTEXT_ATTR_PARA_SPACING_BEFORE  0x00001000
#define wxTEXT_ATTR_LINE_SPACING         0x00002000
#define wxTEXT_ATTR_CHARACTER_STYLE_NAME  0x00004000
#define wxTEXT_ATTR_PARAGRAPH_STYLE_NAME  0x00008000
#define wxTEXT_ATTR_LIST_STYLE_NAME       0x00010000
#define wxTEXT_ATTR_BULLET_STYLE         0x00020000
```

```
#define wxTEXT_ATTR_BULLET_NUMBER      0x00040000
#define wxTEXT_ATTR_BULLET_TEXT        0x00080000
#define wxTEXT_ATTR_BULLET_NAME        0x00100000
#define wxTEXT_ATTR_URL                  0x00200000
#define wxTEXT_ATTR_PAGE_BREAK           0x00400000
#define wxTEXT_ATTR_EFFECTS              0x00800000
#define wxTEXT_ATTR_OUTLINE_LEVEL        0x01000000
```

The following styles can be passed to `wxRichTextAttr::SetBulletStyle`:

```
#define wxTEXT_ATTR_BULLET_STYLE_NONE      0x00000000
#define wxTEXT_ATTR_BULLET_STYLE_ARABIC    0x00000001
#define wxTEXT_ATTR_BULLET_STYLE_LETTERS_UPPER 0x00000002
#define wxTEXT_ATTR_BULLET_STYLE_LETTERS_LOWER 0x00000004
#define wxTEXT_ATTR_BULLET_STYLE_ROMAN_UPPER 0x00000008
#define wxTEXT_ATTR_BULLET_STYLE_ROMAN_LOWER 0x00000010
#define wxTEXT_ATTR_BULLET_STYLE_SYMBOL     0x00000020
#define wxTEXT_ATTR_BULLET_STYLE_BITMAP     0x00000040
#define wxTEXT_ATTR_BULLET_STYLE_PARENTHESIS 0x00000080
#define wxTEXT_ATTR_BULLET_STYLE_PERIOD     0x00000100
#define wxTEXT_ATTR_BULLET_STYLE_STANDARD   0x00000200
#define wxTEXT_ATTR_BULLET_STYLE_RIGHT_PARENTHESIS 0x00000400
#define wxTEXT_ATTR_BULLET_STYLE_OUTLINE    0x00000800
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_LEFT 0x00000000
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_RIGHT 0x00001000
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_CENTRE 0x00002000
```

Of these, `wxTEXT_ATTR_BULLET_STYLE_BITMAP` is unimplemented.

The following constants can be passed to `wxRichTextAttr::SetLineSpacing`:

```
#define wxTEXT_ATTR_LINE_SPACING_NORMAL      10
#define wxTEXT_ATTR_LINE_SPACING_HALF        15
#define wxTEXT_ATTR_LINE_SPACING_TWICE       20
```

The following styles can be passed to `wxTextAttrEx::SetTextEffects`:

```
#define wxTEXT_ATTR_EFFECT_NONE              0x00000000
#define wxTEXT_ATTR_EFFECT_CAPITALS          0x00000001
#define wxTEXT_ATTR_EFFECT_SMALL_CAPITALS    0x00000002
#define wxTEXT_ATTR_EFFECT_STRIKETHROUGH     0x00000004
#define wxTEXT_ATTR_EFFECT_DOUBLE_STRIKETHROUGH 0x00000008
#define wxTEXT_ATTR_EFFECT_SHADOW             0x00000010
#define wxTEXT_ATTR_EFFECT_EMBOSS             0x00000020
#define wxTEXT_ATTR_EFFECT_OUTLINE            0x00000040
#define wxTEXT_ATTR_EFFECT_ENGRAVE            0x00000080
#define wxTEXT_ATTR_EFFECT_SUPERSCRIPT        0x00000100
#define wxTEXT_ATTR_EFFECT_SUBSCRIPT          0x00000200
```

Of these, only `wxTEXT_ATTR_EFFECT_CAPITALS` and `wxTEXT_ATTR_EFFECT_STRIKETHROUGH` are implemented.

#### See also

`wxTextAttr` (p. 1618), `wxTextAttrEx` (p. 1623), `wxRichTextCtrl` (p. 1316)

### **wxRichTextAttr::wxRichTextAttr**

**wxRichTextAttr()**

**wxRichTextAttr(const wxColour& colText, const wxColour& colBack = wxNullColour, wxTextAttrAlignment alignment = wxTEXT\_ALIGNMENT\_DEFAULT)**

**wxRichTextAttr(const wxTextAttrEx& attr)**

Constructors.

### **wxRichTextAttr::Apply**

**bool Combine(const wxRichTextAttrEx& style, const wxRichTextAttrEx\* compareWith = NULL)**

Applies the attributes in *style* to the original object, but not those attributes from *style* that are the same as those in *compareWith* (if passed).

See also `wxRichTextAttr::Combine` (p. 1284) for a function that does almost the same but returns a new object instead of modifying the original object.

### **wxRichTextAttr::Combine**

**wxRichTextAttr Combine(const wxRichTextAttrEx& style, const wxRichTextAttrEx\* compareWith = NULL) const**

Combines 'this' with *style*, but not applying attributes from *style* that are the same as those in *compareWith* (if passed). A `wxRichTextAttr` object is returned and the original object is not changed.

See also `wxRichTextAttr::Apply` (p. 1284) for a function that does almost the same but modifies the original object instead of returning a new one.

### **wxRichTextAttr::CreateFont**

**wxFont CreateFont() const**

Creates a font from the font attributes.

### **wxRichTextAttr::GetAlignment**

**wxTextAttrAlignment GetAlignment() const**

Returns the alignment flags. See *wxRichTextAttr::SetAlignment* (p. 1292) for a list of available styles.

### **wxRichTextAttr::GetBackgroundColour**

**const wxColour& GetBackgroundColour() const**

Returns the background colour.

### **wxRichTextAttr::GetBulletFont**

**const wxString& GetBulletFont() const**

Returns a string containing the name of the font associated with the bullet symbol. Only valid for attributes with `wxTEXT_ATTR_BULLET_SYMBOL`.

### **wxRichTextAttr::GetBulletName**

**const wxString& GetBulletName() const**

Returns the standard bullet name, applicable if the bullet style is `wxTEXT_ATTR_BULLET_STYLE_STANDARD`. Currently the following standard bullet names are supported:

- `standard/circle`
- `standard/square`
- `standard/diamond`
- `standard/triangle`

If you wish your application to support further bullet graphics, you can derive a class from `wxRichTextRenderer` or `wxRichTextStdRenderer`, override `DrawStandardBullet` and `EnumerateStandardBulletNames`, and set an instance of the class using *wxRichTextBuffer::SetRenderer* (p. 1314).

### **wxRichTextAttr::GetBulletNumber**

**int GetBulletNumber() const**

Returns the bullet number.

### **wxRichTextAttr::GetBulletStyle**

**int GetBulletStyle() const**

Returns the bullet style. See *wxRichTextAttr::SetBulletStyle* (p. 1293) for a list of available styles.

**wxRichTextAttr::GetBulletText****const wxString& GetBulletText() const**

Returns the bullet text, which could be a symbol, or (for example) cached outline text.

**wxRichTextAttr::GetCharacterStyleName****const wxString& GetCharacterStyleName() const**

Returns the name of the character style.

**wxRichTextAttr::GetFlags****long GetFlags() const**

Returns flags indicating which attributes are applicable. See *wxRichTextAttr::SetFlags* (p. 1294) for a list of available flags.

**wxRichTextAttr::GetFontAttributes****bool GetFontAttributes(const wxFont& font)**

Sets the font attributes from the given font.

**wxRichTextAttr::GetFontFaceName****const wxString& GetFontFaceName() const**

Returns the font face name.

**wxRichTextAttr::GetFontSize****int GetFontSize() const**

Returns the font size in points.

**wxRichTextAttr::GetFontStyle****int GetFontStyle() const**

Returns the font style.

**wxRichTextAttr::GetFontUnderlined****bool GetFontUnderlined() const**

Returns `true` if the font is underlined.

**wxRichTextAttr::GetFontWeight**

**int GetFontWeight() const**

Returns the font weight.

**wxRichTextAttr::GetLeftIndent**

**long GetLeftIndent() const**

Returns the left indent in tenths of a millimetre.

**wxRichTextAttr::GetLeftSubIndent**

**long GetLeftSubIndent() const**

Returns the left sub-indent in tenths of a millimetre.

**wxRichTextAttr::GetLineSpacing**

**int GetLineSpacing() const**

Returns the line spacing value, one of wxTEXT\_ATTR\_LINE\_SPACING\_NORMAL, wxTEXT\_ATTR\_LINE\_SPACING\_HALF, and wxTEXT\_ATTR\_LINE\_SPACING\_TWICE.

**wxRichTextAttr::GetListStyleName**

**const wxString& GetListStyleName() const**

Returns the name of the list style.

**wxRichTextAttr::GetOutlineLevel**

**bool GetOutlineLevel() const**

Returns the outline level.

**wxRichTextAttr::GetParagraphSpacingAfter**

**int GetParagraphSpacingAfter() const**

Returns the space in tenths of a millimeter after the paragraph.

**wxRichTextAttr::GetParagraphSpacingBefore**

**int GetParagraphSpacingBefore() const**

Returns the space in tenths of a millimeter before the paragraph.

**wxRichTextAttr::GetParagraphStyleName**

**const wxString& GetParagraphStyleName() const**

Returns the name of the paragraph style.

### **wxRichTextAttr::GetRightIndent**

**long GetRightIndent() const**

Returns the right indent in tenths of a millimeter.

### **wxRichTextAttr::GetTabs**

**const wxArrayInt& GetTabs() const**

Returns an array of tab stops, each expressed in tenths of a millimeter. Each stop is measured from the left margin and therefore each value must be larger than the last.

### **wxRichTextAttr::GetTextColour**

**const wxColour& GetTextColour() const**

Returns the text foreground colour.

### **wxRichTextAttr::GetTextEffectFlags**

**int GetTextEffectFlags() const**

Returns the text effect bits of interest. See *wxRichTextAttr::SetFlags* (p. 1294) for further information.

### **wxRichTextAttr::GetTextEffects**

**int GetTextEffects() const**

Returns the text effects, a bit list of styles. See *wxRichTextAttr::SetTextEffects* (p. 1297) for details.

### **wxRichTextAttr::GetURL**

**const wxString& GetURL() const**

Returns the URL for the content. Content with `wxTEXT_ATTR_URL` style causes `wxRichTextCtrl` to show a hand cursor over it, and `wxRichTextCtrl` generates a `wxTextUrlEvent` when the content is clicked.

### **wxRichTextAttr::HasAlignment**

**bool HasAlignment() const**

Returns `true` if the attribute object specifies alignment.

### **wxRichTextAttr::HasBackgroundColour**



**bool HasBackgroundColour() const**

Returns `true` if the attribute object specifies a background colour.

**wxRichTextAttr::HasBulletName**

**bool HasBulletName() const**

Returns `true` if the attribute object specifies a standard bullet name.

**wxRichTextAttr::HasBulletNumber**

**bool HasBulletNumber() const**

Returns `true` if the attribute object specifies a bullet number.

**wxRichTextAttr::HasBulletStyle**

**bool HasBulletStyle() const**

Returns `true` if the attribute object specifies a bullet style.

**wxRichTextAttr::HasBulletText**

**bool HasBulletText() const**

Returns `true` if the attribute object specifies bullet text (usually specifying a symbol).

**wxRichTextAttr::HasCharacterStyleName**

**bool HasCharacterStyleName() const**

Returns `true` if the attribute object specifies a character style name.

**wxRichTextAttr::HasFontFaceName**

**bool HasFontFaceName() const**

Returns `true` if the attribute object specifies a font face name.

**wxRichTextAttr::HasFlag**

**bool HasFlag(long *flag*) const**

Returns `true` if the *flag* is present in the attribute object's flag bitlist.

**wxRichTextAttr::HasFont**

**bool HasFont() const**

Returns `true` if the attribute object specifies any font attributes.

**wxRichTextAttr::HasFontItalic****bool HasFontItalic() const**

Returns `true` if the attribute object specifies italic style.

**wxRichTextAttr::HasLeftIndent****bool HasLeftIndent() const**

Returns `true` if the attribute object specifies a left indent.

**wxRichTextAttr::HasLineSpacing****bool HasLineSpacing() const**

Returns `true` if the attribute object specifies line spacing.

**wxRichTextAttr::HasListStyleName****bool HasListStyleName() const**

Returns `true` if the attribute object specifies a list style name.

**wxRichTextAttr::HasOutlineLevel****bool HasOutlineLevel() const**

Returns `true` if the attribute object specifies an outline level.

**wxRichTextAttr::HasPageBreak****bool HasPageBreak() const**

Returns `true` if the attribute object specifies a page break before this paragraph.

**wxRichTextAttr::HasParagraphSpacingAfter****bool HasParagraphSpacingAfter() const**

Returns `true` if the attribute object specifies spacing after a paragraph.

**wxRichTextAttr::HasParagraphSpacingBefore****bool HasParagraphSpacingBefore() const**

Returns `true` if the attribute object specifies spacing before a paragraph.

**wxRichTextAttr::HasParagraphStyleName****bool HasParagraphStyleName() const**

Returns `true` if the attribute object specifies a paragraph style name.

**wxRichTextAttr::HasRightIndent****bool HasRightIndent() const**

Returns `true` if the attribute object specifies a right indent.

**wxRichTextAttr::HasFontSize****bool HasFontSize() const**

Returns `true` if the attribute object specifies a font point size.

**wxRichTextAttr::HasTabs****bool HasTabs() const**

Returns `true` if the attribute object specifies tab stops.

**wxRichTextAttr::HasTextColour****bool HasTextColour() const**

Returns `true` if the attribute object specifies a text foreground colour.

**wxRichTextAttr::HasTextEffects****bool HasTextEffects() const**

Returns `true` if the attribute object specifies text effects.

**wxRichTextAttr::HasFontUnderlined****bool HasFontUnderlined() const**

Returns `true` if the attribute object specifies either underlining or no underlining.

**wxRichTextAttr::HasURL****bool HasURL() const**

Returns `true` if the attribute object specifies a URL.

**wxRichTextAttr::HasFontWeight**

**bool HasFontWeight() const**

Returns `true` if the attribute object specifies font weight (bold, light or normal).

**wxRichTextAttr::IsCharacterStyle****bool IsCharacterStyle() const**

Returns `true` if the object represents a character style, that is, the flags specify a font or a text background or foreground colour.

**wxRichTextAttr::IsDefault****bool IsDefault() const**

Returns `false` if we have any attributes set, `true` otherwise.

**wxRichTextAttr::IsParagraphStyle****bool IsParagraphStyle() const**

Returns `true` if the object represents a paragraph style, that is, the flags specify alignment, indentation, tabs, paragraph spacing, or bullet style.

**wxRichTextAttr::SetAlignment****void SetAlignment(wxTextAttrAlignment alignment)**

Sets the paragraph alignment. These are the possible values for *alignment*:

```
enum wxTextAttrAlignment
{
    wxTEXT_ALIGNMENT_DEFAULT,
    wxTEXT_ALIGNMENT_LEFT,
    wxTEXT_ALIGNMENT_CENTRE,
    wxTEXT_ALIGNMENT_CENTER = wxTEXT_ALIGNMENT_CENTRE,
    wxTEXT_ALIGNMENT_RIGHT,
    wxTEXT_ALIGNMENT_JUSTIFIED
};
```

Of these, `wxTEXT_ALIGNMENT_JUSTIFIED` is unimplemented. In future justification may be supported when printing or previewing, only.

**wxRichTextAttr::SetBackgroundColour****void SetBackgroundColour(const wxColour& colBack)**

Sets the background colour.

**wxRichTextAttr::SetBulletFont****void SetBulletFont(const wxString& font)**

Sets the name of the font associated with the bullet symbol. Only valid for attributes with wxTEXT\_ATTR\_BULLET\_SYMBOL.

**wxRichTextAttr::SetBulletName****void SetBulletName(const wxString& name)**

Sets the standard bullet name, applicable if the bullet style is wxTEXT\_ATTR\_BULLET\_STYLE\_STANDARD. See *wxRichTextAttr::GetBulletName* (p. 1285) for a list of supported names, and how to expand the range of supported types.

**wxRichTextAttr::SetBulletNumber****void SetBulletNumber(int n)**

Sets the bullet number.

**wxRichTextAttr::SetBulletStyle****void SetBulletStyle(int style)**

Sets the bullet style. The following styles can be passed:

#define wxTEXT_ATTR_BULLET_STYLE_NONE	0x00000000
#define wxTEXT_ATTR_BULLET_STYLE_ARABIC	0x00000001
#define wxTEXT_ATTR_BULLET_STYLE_LETTERS_UPPER	0x00000002
#define wxTEXT_ATTR_BULLET_STYLE_LETTERS_LOWER	0x00000004
#define wxTEXT_ATTR_BULLET_STYLE_ROMAN_UPPER	0x00000008
#define wxTEXT_ATTR_BULLET_STYLE_ROMAN_LOWER	0x00000010
#define wxTEXT_ATTR_BULLET_STYLE_SYMBOL	0x00000020
#define wxTEXT_ATTR_BULLET_STYLE_BITMAP	0x00000040
#define wxTEXT_ATTR_BULLET_STYLE_PARENTHESSES	0x00000080
#define wxTEXT_ATTR_BULLET_STYLE_PERIOD	0x00000100
#define wxTEXT_ATTR_BULLET_STYLE_STANDARD	0x00000200
#define wxTEXT_ATTR_BULLET_STYLE_RIGHT_PARENTHESIS	0x00000400
#define wxTEXT_ATTR_BULLET_STYLE_OUTLINE	0x00000800
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_LEFT	0x00000000
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_RIGHT	0x00001000
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_CENTRE	0x00002000

Currently wxTEXT\_ATTR\_BULLET\_STYLE\_BITMAP is not supported.

**wxRichTextAttr::SetBulletText****void SetBulletText(const wxString& text)**

Sets the bullet text, which could be a symbol, or (for example) cached outline text.

**wxRichTextAttr::SetCharacterStyleName****void SetCharacterStyleName(const wxString& name)**

Sets the character style name.

**wxRichTextAttr::SetFlags****void SetFlags(long flags)**

Sets the flags determining which styles are being specified. The following flags can be passed in a bitlist:

```
// Standard wxTextAttr constants

#define wxTEXT_ATTR_TEXT_COLOUR          0x00000001
#define wxTEXT_ATTR_BACKGROUND_COLOUR    0x00000002
#define wxTEXT_ATTR_FONT_FACE            0x00000004
#define wxTEXT_ATTR_FONT_SIZE            0x00000008
#define wxTEXT_ATTR_FONT_WEIGHT          0x00000010
#define wxTEXT_ATTR_FONT_ITALIC          0x00000020
#define wxTEXT_ATTR_FONT_UNDERLINE       0x00000040
#define wxTEXT_ATTR_FONT \
    wxTEXT_ATTR_FONT_FACE | wxTEXT_ATTR_FONT_SIZE | \
wxTEXT_ATTR_FONT_WEIGHT | \
| wxTEXT_ATTR_FONT_ITALIC | wxTEXT_ATTR_FONT_UNDERLINE
#define wxTEXT_ATTR_ALIGNMENT            0x00000080
#define wxTEXT_ATTR_LEFT_INDENT          0x00000100
#define wxTEXT_ATTR_RIGHT_INDENT         0x00000200
#define wxTEXT_ATTR_TABS                  0x00000400

// Extra formatting flags not in wxTextAttr

#define wxTEXT_ATTR_PARA_SPACING_AFTER    0x00000800
#define wxTEXT_ATTR_PARA_SPACING_BEFORE  0x00001000
#define wxTEXT_ATTR_LINE_SPACING         0x00002000
#define wxTEXT_ATTR_CHARACTER_STYLE_NAME  0x00004000
#define wxTEXT_ATTR_PARAGRAPH_STYLE_NAME  0x00008000
#define wxTEXT_ATTR_LIST_STYLE_NAME       0x00010000
#define wxTEXT_ATTR_BULLET_STYLE         0x00020000
#define wxTEXT_ATTR_BULLET_NUMBER        0x00040000
#define wxTEXT_ATTR_BULLET_TEXT          0x00080000
#define wxTEXT_ATTR_BULLET_NAME          0x00100000
#define wxTEXT_ATTR_URL                   0x00200000
#define wxTEXT_ATTR_PAGE_BREAK            0x00400000
#define wxTEXT_ATTR_EFFECTS               0x00800000
#define wxTEXT_ATTR_OUTLINE_LEVEL         0x01000000
```

**wxRichTextAttr::SetFontFaceName****void SetFontFaceName(const wxString& faceName)**

Sets the paragraph alignment.

**wxRichTextAttr::SetFontSize****void SetFontSize(int *pointSize*)**

Sets the font size in points.

**wxRichTextAttr::SetFontStyle****void SetFontStyle(int *fontStyle*)**

Sets the font style (normal, italic or slanted).

**wxRichTextAttr::SetFontUnderlined****void SetFontUnderlined(bool *underlined*)**

Sets the font underlining.

**wxRichTextAttr::SetFontWeight****void SetFontWeight(int *fontWeight*)**

Sets the font weight.

**wxRichTextAttr::SetLeftIndent****void SetLeftIndent(int *indent*, int *subIndent* = 0)**

Sets the left indent and left subindent in tenths of a millimetre.

The sub-indent is an offset from the left of the paragraph, and is used for all but the first line in a paragraph. A positive value will cause the first line to appear to the left of the subsequent lines, and a negative value will cause the first line to be indented relative to the subsequent lines.

wxRichTextBuffer uses indentation to render a bulleted item. The left indent is the distance between the margin and the bullet. The content of the paragraph, including the first line, starts at leftMargin + leftSubIndent. So the distance between the left edge of the bullet and the left of the actual paragraph is leftSubIndent.

**wxRichTextAttr::SetLineSpacing****void SetLineSpacing(int *spacing*)**

Sets the line spacing. *spacing* is a multiple, where 10 means single-spacing, 15 means 1.5 spacing, and 20 means double spacing. The following constants are defined for convenience:

```
#define wxTEXT_ATTR_LINE_SPACING_NORMAL 10
```

#define wxTEXT_ATTR_LINE_SPACING_HALF	15
#define wxTEXT_ATTR_LINE_SPACING_TWICE	20

**wxRichTextAttr::SetListStyleName****void SetListStyleName(const wxString& name)**

Sets the list style name.

**wxRichTextAttr::SetOutlineLevel****void SetOutlineLevel(int level)**

Specifies the outline level. Zero represents normal text. At present, the outline level is not used, but may be used in future for determining list levels and for applications that need to store document structure information.

**wxRichTextAttr::SetPageBreak****void SetPageBreak(bool pageBreak = true)**

Specifies a page break before this paragraph.

**wxRichTextAttr::SetParagraphSpacingAfter****void SetParagraphSpacingAfter(int spacing)**

Sets the spacing after a paragraph, in tenths of a millimetre.

**wxRichTextAttr::SetParagraphSpacingBefore****void SetParagraphSpacingBefore(int spacing)**

Sets the spacing before a paragraph, in tenths of a millimetre.

**wxRichTextAttr::SetParagraphStyleName****void SetParagraphStyleName(const wxString& name)**

Sets the name of the paragraph style.

**wxRichTextAttr::SetRightIndent****void SetRightIndent(int indent)**

Sets the right indent in tenths of a millimetre.

**wxRichTextAttr::SetTabs**



**void SetTabs(const wxArrayInt& tabs)**

Sets the tab stops, expressed in tenths of a millimetre. Each stop is measured from the left margin and therefore each value must be larger than the last.

**wxRichTextAttr::SetTextColour****void SetTextColour(const wxColour& col/Text)**

Sets the text foreground colour.

**wxRichTextAttr::SetTextEffectFlags****void SetTextEffectFlags(int flags)**

Sets the text effect bits of interest. You should also pass `wxTEXT_ATTR_EFFECTS` to `wxRichTextAttr::SetFlags` (p. 1294). See `wxRichTextAttr::SetFlags` (p. 1294) for further information.

**wxRichTextAttr::SetTextEffects****void SetTextEffects(int effects)**

Sets the text effects, a bit list of styles.

The following styles can be passed:

<code>#define wxTEXT_ATTR_EFFECT_NONE</code>	<code>0x00000000</code>
<code>#define wxTEXT_ATTR_EFFECT_CAPITALS</code>	<code>0x00000001</code>
<code>#define wxTEXT_ATTR_EFFECT_SMALL_CAPITALS</code>	<code>0x00000002</code>
<code>#define wxTEXT_ATTR_EFFECT_STRIKETHROUGH</code>	<code>0x00000004</code>
<code>#define wxTEXT_ATTR_EFFECT_DOUBLE_STRIKETHROUGH</code>	<code>0x00000008</code>
<code>#define wxTEXT_ATTR_EFFECT_SHADOW</code>	<code>0x00000010</code>
<code>#define wxTEXT_ATTR_EFFECT_EMBOSS</code>	<code>0x00000020</code>
<code>#define wxTEXT_ATTR_EFFECT_OUTLINE</code>	<code>0x00000040</code>
<code>#define wxTEXT_ATTR_EFFECT_ENGRAVE</code>	<code>0x00000080</code>
<code>#define wxTEXT_ATTR_EFFECT_SUPERSCRIPT</code>	<code>0x00000100</code>
<code>#define wxTEXT_ATTR_EFFECT_SUBSCRIPT</code>	<code>0x00000200</code>

Of these, only `wxTEXT_ATTR_EFFECT_CAPITALS` and `wxTEXT_ATTR_EFFECT_STRIKETHROUGH` are implemented. `wxTEXT_ATTR_EFFECT_CAPITALS` capitalises text when displayed (leaving the case of the actual buffer text unchanged), and `wxTEXT_ATTR_EFFECT_STRIKETHROUGH` draws a line through text.

To set effects, you should also pass `wxTEXT_ATTR_EFFECTS` to `wxRichTextAttr::SetFlags` (p. 1294), and call `wxRichTextAttr::SetTextEffectFlags` (p. 1297) with the styles (taken from the above set) that you are interested in setting.

**wxRichTextAttr::SetURL**

**void SetURL(const wxString& url)**

Sets the URL for the content. Sets the wxTEXT\_ATTR\_URL style; content with this style causes wxRichTextCtrl to show a hand cursor over it, and wxRichTextCtrl generates a wxTextUrlEvent when the content is clicked.

**wxRichTextAttr::operator=**

**void operator operator=(const wxTextAttrEx& attr)**

Assignment from a *wxTextAttrEx* (p. 1618) object.

**void operator operator=(const wxRichTextAttr& attr)**

Assignment from a *wxRichTextAttr* (p. 1281) object.

**wxRichTextAttr::wxTextAttrEx**

**operator wxTextAttrEx() const**

Makes a *wxTextAttrEx* (p. 1623) object from this object.

## wxRichTextBuffer

This class represents the whole buffer associated with a *wxRichTextCtrl* (p. 1316).

### Derived from

wxRichTextParagraphLayoutBox

### Include files

<wx/richtext/richtextbuffer.h>

### Data structures

### See also

*wxTextAttr* (p. 1618), *wxTextAttrEx* (p. 1623), *wxRichTextAttr* (p. 1281), *wxRichTextCtrl* (p. 1316)

**wxRichTextBuffer::wxRichTextBuffer**

**wxRichTextBuffer(const wxRichTextBuffer& obj)**

Copy constructor.

**wxRichTextBuffer()**

Default constructors.

**wxRichTextBuffer::~~wxRichTextBuffer****~wxRichTextBuffer()**

Destructor.

**wxRichTextBuffer::AddEventHandler****bool AddEventHandler(wxEvtHandler\* handler)**

Adds an event handler to the buffer's list of handlers. A buffer associated with a control has the control as the only event handler, but the application is free to add more if further notification is required. All handlers are notified of an event originating from the buffer, such as the replacement of a style sheet during loading. The buffer never deletes any of the event handlers, unless *wxRichTextBuffer::RemoveEventHandler* (p. 1312) is called with `true` as the second argument.

**wxRichTextBuffer::AddHandler****void AddHandler(wxRichTextFileHandler\* handler)**

Adds a file handler.

**wxRichTextBuffer::AddParagraph****wxRichTextRange AddParagraph(const wxString& text)**

Adds a paragraph of text.

**wxRichTextBuffer::BatchingUndo****bool BatchingUndo() const**

Returns `true` if the buffer is currently collapsing commands into a single notional command.

**wxRichTextBuffer::BeginAlignment****bool BeginAlignment(wxTextAttrAlignment alignment)**

Begins using alignment.

**wxRichTextBuffer::BeginBatchUndo****bool BeginBatchUndo(const wxString& cmdName)**

Begins collapsing undo/redo commands. Note that this may not work properly if combining commands that delete or insert content, changing ranges for subsequent actions.

*cmdName* should be the name of the combined command that will appear next to Undo

and Redo in the edit menu.

### **wxRichTextBuffer::BeginBold**

**bool BeginBold()**

Begin applying bold.

### **wxRichTextBuffer::BeginCharacterStyle**

**bool BeginCharacterStyle(const wxString& *characterStyle*)**

Begins applying the named character style.

### **wxRichTextBuffer::BeginFont**

**bool BeginFont(const wxFont& *font*)**

Begins using this font.

### **wxRichTextBuffer::BeginFontSize**

**bool BeginFontSize(int *pointSize*)**

Begins using the given point size.

### **wxRichTextBuffer::BeginItalic**

**bool BeginItalic()**

Begins using italic.

### **wxRichTextBuffer::BeginLeftIndent**

**bool BeginLeftIndent(int *leftIndent*, int *leftSubIndent* = 0)**

Begin using *leftIndent* for the left indent, and optionally *leftSubIndent* for the sub-indent. Both are expressed in tenths of a millimetre.

The sub-indent is an offset from the left of the paragraph, and is used for all but the first line in a paragraph. A positive value will cause the first line to appear to the left of the subsequent lines, and a negative value will cause the first line to be indented relative to the subsequent lines.

### **wxRichTextBuffer::BeginLineSpacing**

**bool BeginLineSpacing(int *lineSpacing*)**

Begins line spacing using the specified value. *spacing* is a multiple, where 10 means single-spacing, 15 means 1.5 spacing, and 20 means double spacing. The following

constants are defined for convenience:

```
#define wxTEXT_ATTR_LINE_SPACING_NORMAL    10
#define wxTEXT_ATTR_LINE_SPACING_HALF     15
#define wxTEXT_ATTR_LINE_SPACING_TWICE     20
```

### **wxRichTextBuffer::BeginListStyle**

**bool BeginListStyle(const wxString& listStyle, int level=1, int number=1)**

Begins using a specified list style. Optionally, you can also pass a level and a number.

### **wxRichTextBuffer::BeginNumberedBullet**

**bool BeginNumberedBullet(int bulletNumber, int leftIndent, int leftSubIndent, int bulletStyle = wxTEXT\_ATTR\_BULLET\_STYLE\_ARABIC|wxTEXT\_ATTR\_BULLET\_STYLE\_PERIOD)**

Begins a numbered bullet. This call will be needed for each item in the list, and the application should take care of incrementing the numbering.

*bulletNumber* is a number, usually starting with 1.

*leftIndent* and *leftSubIndent* are values in tenths of a millimetre.

*bulletStyle* is a bitlist of the following values:

```
#define wxTEXT_ATTR_BULLET_STYLE_NONE      0x00000000
#define wxTEXT_ATTR_BULLET_STYLE_ARABIC   0x00000001
#define wxTEXT_ATTR_BULLET_STYLE_LETTERS_UPPER 0x00000002
#define wxTEXT_ATTR_BULLET_STYLE_LETTERS_LOWER 0x00000004
#define wxTEXT_ATTR_BULLET_STYLE_ROMAN_UPPER 0x00000008
#define wxTEXT_ATTR_BULLET_STYLE_ROMAN_LOWER 0x00000010
#define wxTEXT_ATTR_BULLET_STYLE_SYMBOL    0x00000020
#define wxTEXT_ATTR_BULLET_STYLE_BITMAP    0x00000040
#define wxTEXT_ATTR_BULLET_STYLE_PARENTHESSES 0x00000080
#define wxTEXT_ATTR_BULLET_STYLE_PERIOD    0x00000100
#define wxTEXT_ATTR_BULLET_STYLE_STANDARD 0x00000200
#define wxTEXT_ATTR_BULLET_STYLE_RIGHT_PARENTHESIS 0x00000400
#define wxTEXT_ATTR_BULLET_STYLE_OUTLINE   0x00000800
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_LEFT 0x00000000
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_RIGHT 0x00001000
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_CENTRE 0x00002000
```

wxRichTextBuffer uses indentation to render a bulleted item. The left indent is the distance between the margin and the bullet. The content of the paragraph, including the first line, starts at leftMargin + leftSubIndent. So the distance between the left edge of the bullet and the left of the actual paragraph is leftSubIndent.

**wxRichTextBuffer::BeginParagraphSpacing****bool BeginParagraphSpacing**(int *before*, int *after*)

Begins paragraph spacing; pass the before-paragraph and after-paragraph spacing in tenths of a millimetre.

**wxRichTextBuffer::BeginParagraphStyle****bool BeginParagraphStyle**(const wxString& *paragraphStyle*)

Begins applying the named paragraph style.

**wxRichTextBuffer::BeginRightIndent****bool BeginRightIndent**(int *rightIndent*)

Begins a right indent, specified in tenths of a millimetre.

**wxRichTextBuffer::BeginStyle****bool BeginStyle**(const wxTextAttrEx& *style*)

Begins using a specified style.

**wxRichTextBuffer::BeginSuppressUndo****bool BeginSuppressUndo**()

Begins suppressing undo/redo commands. The way undo is suppressed may be implemented differently by each command. If not dealt with by a command implementation, then it will be implemented automatically by not storing the command in the undo history when the action is submitted to the command processor.

**wxRichTextBuffer::BeginStandardBullet****bool BeginStandardBullet**(const wxString& *bulletName*, int *leftIndent*, int *leftSubIndent*, int *bulletStyle* = wxTEXT\_ATTR\_BULLET\_STYLE\_STANDARD)

Begins applying a standard bullet, using one of the standard bullet names (currently *standard/circle* or *standard/square*. See *BeginNumberedBullet* (p. 1301) for an explanation of how indentation is used to render the bulleted paragraph.

**wxRichTextBuffer::BeginSymbolBullet****bool BeginSymbolBullet**(wxChar *symbol*, int *leftIndent*, int *leftSubIndent*, int *bulletStyle* = wxTEXT\_ATTR\_BULLET\_STYLE\_SYMBOL)

Begins applying a symbol bullet, using a character from the current font. See *BeginNumberedBullet* (p. 1301) for an explanation of how indentation is used to render

the bulleted paragraph.

### **wxRichTextBuffer::BeginTextColour**

**bool BeginTextColour(const wxColour& colour)**

Begins using the specified text foreground colour.

### **wxRichTextBuffer::BeginUnderline**

**bool BeginUnderline()**

Begins using underline.

### **wxRichTextBuffer::BeginURL**

**bool BeginURL(const wxString& url, const wxString& characterStyle = wxEmptyString)**

Begins applying wxTEXT\_ATTR\_URL to the content. Pass a URL and optionally, a character style to apply, since it is common to mark a URL with a familiar style such as blue text with underlining.

### **wxRichTextBuffer::CanPasteFromClipboard**

**bool CanPasteFromClipboard() const**

Returns `true` if content can be pasted from the clipboard.

### **wxRichTextBuffer::CleanUpHandlers**

**void CleanUpHandlers()**

Cleans up the file handlers.

### **wxRichTextBuffer::Clear**

**void Clear()**

Clears the buffer.

### **wxRichTextBuffer::ClearListStyle**

**bool ClearListStyle(const wxRichTextRange& range, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO)**

**bool ClearListStyle(const wxRichTextRange& range, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO)**

Clears the list style from the given range, clearing list-related attributes and applying any

named paragraph style associated with each paragraph.

*flags* is a bit list of the following:

- `wxRICHTEXT_SETSTYLE_WITH_UNDO`: specifies that this command will be undoable.

See also `wxRichTextBuffer::SetListStyle` (p. 1313), `wxRichTextBuffer::PromoteList` (p. 1312), `wxRichTextBuffer::NumberList` (p. 1311).

### **`wxRichTextBuffer::ClearStyleStack`**

**`void ClearStyleStack()`**

Clears the style stack.

### **`wxRichTextBuffer::Clone`**

**`wxRichTextObject* Clone() const`**

Clones the object.

### **`wxRichTextBuffer::Copy`**

**`void Copy(const wxRichTextBuffer& obj)`**

Copies the given buffer.

### **`wxRichTextBuffer::CopyToClipboard`**

**`bool CopyToClipboard(const wxRichTextRange& range)`**

Copy the given range to the clipboard.

### **`wxRichTextBuffer::DeleteRangeWithUndo`**

**`bool DeleteRangeWithUndo(const wxRichTextRange& range, wxRichTextCtrl* ctrl)`**

Submits a command to delete the given range.

### **`wxRichTextBuffer::Dump`**

**`void Dump()`**

**`void Dump(wxTextOutputStream& stream)`**

Dumps the contents of the buffer for debugging purposes.

### **`wxRichTextBuffer::EndAlignment`**

**`bool EndAlignment()`**



Ends alignment.

**wxRichTextBuffer::EndAllStyles****bool EndAllStyles()**

Ends all styles that have been started with a Begin... command.

**wxRichTextBuffer::EndBatchUndo****bool EndBatchUndo()**

Ends collapsing undo/redo commands, and submits the combined command.

**wxRichTextBuffer::EndBold****bool EndBold()**

Ends using bold.

**wxRichTextBuffer::EndCharacterStyle****bool EndCharacterStyle()**

Ends using the named character style.

**wxRichTextBuffer::EndFont****bool EndFont()**

Ends using a font.

**wxRichTextBuffer::EndFontSize****bool EndFontSize()**

Ends using a point size.

**wxRichTextBuffer::EndItalic****bool EndItalic()**

Ends using italic.

**wxRichTextBuffer::EndLeftIndent****bool EndLeftIndent()**

Ends using a left indent.

**wxRichTextBuffer::EndLineSpacing****bool EndLineSpacing()**

Ends using a line spacing.

**wxRichTextBuffer::EndListStyle****bool EndListStyle()**

Ends using a specified list style.

**wxRichTextBuffer::EndNumberedBullet****bool EndNumberedBullet()**

Ends a numbered bullet.

**wxRichTextBuffer::EndParagraphSpacing****bool EndParagraphSpacing()**

Ends paragraph spacing.

**wxRichTextBuffer::EndParagraphStyle****bool EndParagraphStyle()**

Ends applying a named character style.

**wxRichTextBuffer::EndRightIndent****bool EndRightIndent()**

Ends using a right indent.

**wxRichTextBuffer::EndStyle****bool EndStyle()**

Ends the current style.

**wxRichTextBuffer::EndSuppressUndo****bool EndSuppressUndo()**

Ends suppressing undo/redo commands.

**wxRichTextBuffer::EndSymbolBullet**

**bool EndSymbolBullet()**

Ends using a symbol bullet.

**wxRichTextBuffer::EndStandardBullet**

**bool EndStandardBullet()**

Ends using a standard bullet.

**wxRichTextBuffer::EndTextColour**

**bool EndTextColour()**

Ends using a text foreground colour.

**wxRichTextBuffer::EndUnderline**

**bool EndUnderline()**

Ends using underline.

**wxRichTextBuffer::EndURL**

**bool EndURL()**

Ends applying a URL.

**wxRichTextBuffer::FindHandler**

**wxRichTextFileHandler\* FindHandler(int *imageType*)**

Finds a handler by type.

**wxRichTextFileHandler\* FindHandler(const wxString& *extension*, int *imageType*)**

Finds a handler by extension and type.

**wxRichTextFileHandler\* FindHandler(const wxString& *name*)**

Finds a handler by name.

**wxRichTextBuffer::FindHandlerFilenameOrType**

**wxRichTextFileHandler\* FindHandlerFilenameOrType(const wxString& *filename*, int *imageType*)**

Finds a handler by filename or, if supplied, type.

**wxRichTextBuffer::GetBasicStyle**

**const wxTextAttrEx& GetBasicStyle() const**

Gets the basic (overall) style. This is the style of the whole buffer before further styles are applied, unlike the default style, which only affects the style currently being applied (for example, setting the default style to bold will cause subsequently inserted text to be bold).

**wxRichTextBuffer::GetBatchedCommand****wxRichTextCommand\* GetBatchedCommand() const**

Gets the collapsed command.

**wxRichTextBuffer::GetCommandProcessor****wxCommandProcessor\* GetCommandProcessor() const**

Gets the command processor. A text buffer always creates its own command processor when it is initialized.

**wxRichTextBuffer::GetDefaultStyle****const wxTextAttrEx& GetDefaultStyle() const**

Returns the current default style, affecting the style currently being applied (for example, setting the default style to bold will cause subsequently inserted text to be bold).

**wxRichTextBuffer::GetExtWildcard****wxString GetExtWildcard(bool combine = false, bool save = false, wxArrayInt\* types = NULL)**

Gets a wildcard incorporating all visible handlers. If *types* is present, it will be filled with the file type corresponding to each filter. This can be used to determine the type to pass to *LoadFile* (p. 1308) given a selected filter.

**wxRichTextBuffer::GetHandlers****wxList& GetHandlers()**

Returns the list of file handlers.

**wxRichTextBuffer::GetRenderer****static wxRichTextRenderer\* GetRenderer()**

Returns the object to be used to render certain aspects of the content, such as bullets.

**wxRichTextBuffer::GetStyle****bool GetStyle(long position, wxRichTextAttr& style)**

**bool GetStyle(long position, wxTextAttrEx& style)**

Gets the attributes at the given position.

This function gets the combined style - that is, the style you see on the screen as a result of combining base style, paragraph style and character style attributes. To get the character or paragraph style alone, use *GetUncombinedStyle* (p. 1309).

**wxRichTextBuffer::GetStyleForRange**

**bool GetStyleForRange(const wxRichTextRange& range, wxTextAttrEx& style)**

This function gets a style representing the common, combined attributes in the given range. Attributes which have different values within the specified range will not be included the style flags.

The function is used to get the attributes to display in the formatting dialog: the user can edit the attributes common to the selection, and optionally specify the values of further attributes to be applied uniformly.

To apply the edited attributes, you can use *SetStyle* (p. 1314) specifying the `wxRICHTEXT_SETSTYLE_OPTIMIZE` flag, which will only apply attributes that are different from the *combined* attributes within the range. So, the user edits the effective, displayed attributes for the range, but his choice won't be applied unnecessarily to content. As an example, say the style for a paragraph specifies bold, but the paragraph text doesn't specify a weight. The combined style is bold, and this is what the user will see on-screen and in the formatting dialog. The user now specifies red text, in addition to bold. When applying with *SetStyle*, the content font weight attributes won't be changed to bold because this is already specified by the paragraph. However the text colour attributes *will* be changed to show red.

**wxRichTextBuffer::GetStyleSheet**

**wxRichTextStyleSheet\* GetStyleSheet() const**

Returns the current style sheet associated with the buffer, if any.

**wxRichTextBuffer::GetStyleStackSize**

**size\_t GetStyleStackSize() const**

Get the size of the style stack, for example to check correct nesting.

**wxRichTextBuffer::GetUncombinedStyle**

**bool GetUncombinedStyle(long position, wxRichTextAttr& style)**

**bool GetUncombinedStyle(long position, wxTextAttrEx& style)**

Gets the attributes at the given position.

This function gets the *uncombined style* - that is, the attributes associated with the

paragraph or character content, and not necessarily the combined attributes you see on the screen. To get the combined attributes, use *GetStyle* (p. 1308).

If you specify (any) paragraph attribute in *style*'s flags, this function will fetch the paragraph attributes. Otherwise, it will return the character attributes.

### **wxRichTextBuffer::HitTest**

**int HitTest(wxDC& dc, const wxPoint& pt, long& textPosition)**

Finds the text position for the given position, putting the position in *textPosition* if one is found. *pt* is in logical units (a zero y position is at the beginning of the buffer).

The function returns one of the following values:

```
// The point was not on this object
#define wxRICHTEXT_HITTEST_NONE      0x01
// The point was before the position returned from HitTest
#define wxRICHTEXT_HITTEST_BEFORE    0x02
// The point was after the position returned from HitTest
#define wxRICHTEXT_HITTEST_AFTER     0x04
// The point was on the position returned from HitTest
#define wxRICHTEXT_HITTEST_ON        0x08
// The point was on space outside content
#define wxRICHTEXT_HITTEST_OUTSIDE   0x10
```

### **wxRichTextBuffer::Init**

**void Init()**

Initialisation.

### **wxRichTextBuffer::InitStandardHandlers**

**void InitStandardHandlers()**

Initialises the standard handlers. Currently, only the plain text loading/saving handler is initialised by default.

### **wxRichTextBuffer::InsertHandler**

**void InsertHandler(wxRichTextFileHandler\* handler)**

Inserts a handler at the front of the list.

### **wxRichTextBuffer::InsertImageWithUndo**

**bool InsertImageWithUndo(long pos, const wxRichTextImageBlock& imageBlock, wxRichTextCtrl\* ctrl)**

Submits a command to insert the given image.

#### **wxRichTextBuffer::InsertNewlineWithUndo**

**bool InsertNewlineWithUndo(long pos, wxRichTextCtrl\* ctrl)**

Submits a command to insert a newline.

#### **wxRichTextBuffer::InsertTextWithUndo**

**bool InsertTextWithUndo(long pos, const wxString& text, wxRichTextCtrl\* ctrl)**

Submits a command to insert the given text.

#### **wxRichTextBuffer::IsModified**

**bool IsModified() const**

Returns `true` if the buffer has been modified.

#### **wxRichTextBuffer::LoadFile**

**bool LoadFile(wxInputStream& stream, int type = wxRICHTEXT\_TYPE\_ANY)**

Loads content from a stream.

**bool LoadFile(const wxString& filename, int type = wxRICHTEXT\_TYPE\_ANY)**

Loads content from a file.

#### **wxRichTextBuffer::Modify**

**void Modify(bool modify = true)**

Marks the buffer as modified or unmodified.

#### **wxRichTextBuffer::NumberList**

**bool NumberList(const wxRichTextRange& range, const wxRichTextListStyleDefinition\* style, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO, int startFrom = -1, int listLevel = -1)**

**bool Number(const wxRichTextRange& range, const wxString& styleName, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO, int startFrom = -1, int listLevel = -1)**

Numbers the paragraphs in the given range. Pass flags to determine how the attributes are set. Either the style definition or the name of the style definition (in the current sheet) can be passed.

*flags* is a bit list of the following:

- `wxRICHTEXT_SETSTYLE_WITH_UNDO`: specifies that this command will be undoable.
- `wxRICHTEXT_SETSTYLE_RENUMBER`: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- `wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL`: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also `wxRichTextBuffer::SetListStyle` (p. 1313), `wxRichTextBuffer::PromoteList` (p. 1312), `wxRichTextBuffer::ClearListStyle` (p. 1303).

### **wxRichTextBuffer::PasteFromClipboard**

**bool PasteFromClipboard(long position)**

Pastes the clipboard content to the buffer at the given position.

### **wxRichTextBuffer::PromoteList**

**bool PromoteList(int promoteBy, const wxRichTextRange& range, const wxRichTextListStyleDefinition\* style, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO, int listLevel = -1)**

**bool PromoteList(int promoteBy, const wxRichTextRange& range, const wxString& styleName, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO, int listLevel = -1)**

Promotes or demotes the paragraphs in the given range. A positive *promoteBy* produces a smaller indent, and a negative number produces a larger indent. Pass flags to determine how the attributes are set. Either the style definition or the name of the style definition (in the current sheet) can be passed.

*flags* is a bit list of the following:

- `wxRICHTEXT_SETSTYLE_WITH_UNDO`: specifies that this command will be undoable.
- `wxRICHTEXT_SETSTYLE_RENUMBER`: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- `wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL`: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also `wxRichTextBuffer::SetListStyle` (p. 1313), See also `wxRichTextBuffer::SetListStyle` (p. 1311), `wxRichTextBuffer::ClearListStyle` (p. 1303).

### **wxRichTextBuffer::RemoveEventHandler**

**bool RemoveEventHandler(wxEvtHandler\* handler, bool deleteHandler = false)**



Removes an event handler from the buffer's list of handlers, deleting the object if *deleteHandler* is `true`.

### **wxRichTextBuffer::RemoveHandler**

**bool RemoveHandler(const wxString& name)**

Removes a handler.

### **wxRichTextBuffer::ResetAndClearCommands**

**void ResetAndClearCommands()**

Clears the buffer, adds a new blank paragraph, and clears the command history.

### **wxRichTextBuffer::SaveFile**

**bool SaveFile(wxOutputStream& stream, int type = wxRICHTEXT\_TYPE\_ANY)**

Saves content to a stream.

**bool SaveFile(const wxString& filename, int type = wxRICHTEXT\_TYPE\_ANY)**

Saves content to a file.

### **wxRichTextBuffer::SetBasicStyle**

**void SetBasicStyle(const wxRichTextAttr& style)**

**void SetBasicStyle(const wxTextAttrEx& style)**

Sets the basic (overall) style. This is the style of the whole buffer before further styles are applied, unlike the default style, which only affects the style currently being applied (for example, setting the default style to bold will cause subsequently inserted text to be bold).

### **wxRichTextBuffer::SetDefaultStyle**

**void SetDefaultStyle(const wxTextAttrEx& style)**

Sets the default style, affecting the style currently being applied (for example, setting the default style to bold will cause subsequently inserted text to be bold).

This is not cumulative - setting the default style will replace the previous default style.

### **wxRichTextBuffer::SetListStyle**

**bool SetListStyle(const wxRichTextRange& range, const wxRichTextListStyleDefinition\* style, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO, int startFrom = -1, int listLevel = -1)**

**bool SetListStyle(const wxRichTextRange& range, const wxString& styleName, int**

*flags* = `wxRICHTEXT_SETSTYLE_WITH_UNDO`, `int startFrom = -1`, `int listLevel = -1`)

Sets the list attributes for the given range, passing flags to determine how the attributes are set. Either the style definition or the name of the style definition (in the current sheet) can be passed.

*flags* is a bit list of the following:

- `wxRICHTEXT_SETSTYLE_WITH_UNDO`: specifies that this command will be undoable.
- `wxRICHTEXT_SETSTYLE_RENUMBER`: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- `wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL`: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also `wxRichTextBuffer::NumberList` (p. 1311), `wxRichTextBuffer::PromoteList` (p. 1312), `wxRichTextBuffer::ClearListStyle` (p. 1303).

### **wxRichTextBuffer::SetRenderer**

**static void SetRenderer(wxRichTextRenderer\* renderer)**

Sets *renderer* as the object to be used to render certain aspects of the content, such as bullets. You can override default rendering by deriving a new class from `wxRichTextRenderer` or `wxRichTextStdRenderer`, overriding one or more virtual functions, and setting an instance of the class using this function.

### **wxRichTextBuffer::SetStyle**

**bool SetStyle(const wxRichTextRange& range, const wxRichTextAttr& style, int flags = `wxRICHTEXT_SETSTYLE_WITH_UNDO`)**

**bool SetStyle(const wxRichTextRange& range, const wxTextAttrEx& style, int flags = `wxRICHTEXT_SETSTYLE_WITH_UNDO`)**

Sets the attributes for the given range. Pass flags to determine how the attributes are set.

The end point of range is specified as the last character position of the span of text. So, for example, to set the style for a character at position 5, use the range (5,5). This differs from the `wxRichTextCtrl` API, where you would specify (5,6).

*flags* may contain a bit list of the following values:

- `wxRICHTEXT_SETSTYLE_NONE`: no style flag.
- `wxRICHTEXT_SETSTYLE_WITH_UNDO`: specifies that this operation should be undoable.
- `wxRICHTEXT_SETSTYLE_OPTIMIZE`: specifies that the style should not be applied if the combined style at this point is already the style in question.

- `wxRICHTEXT_SETSTYLE_PARAGRAPHS_ONLY`: specifies that the style should only be applied to paragraphs, and not the content. This allows content styling to be preserved independently from that of e.g. a named paragraph style.
- `wxRICHTEXT_SETSTYLE_CHARACTERS_ONLY`: specifies that the style should only be applied to characters, and not the paragraph. This allows content styling to be preserved independently from that of e.g. a named paragraph style.
- `wxRICHTEXT_SETSTYLE_RESET`: resets (clears) the existing style before applying the new style.
- `wxRICHTEXT_SETSTYLE_REMOVE`: removes the specified style. Only the style flags are used in this operation.

### **`wxRichTextBuffer::SetStyleSheet`**

**`void SetStyleSheet(wxRichTextStyleSheet* styleSheet)`**

Sets the current style sheet, if any. This will allow the application to use named character and paragraph styles found in the style sheet.

### **`wxRichTextBuffer::SubmitAction`**

**`bool SubmitAction(wxRichTextAction* action)`**

Submit an action immediately, or delay it according to whether collapsing is on.

### **`wxRichTextBuffer::SuppressingUndo`**

**`bool SuppressingUndo() const`**

Returns `true` if undo suppression is currently on.

## **`wxRichTextCharacterStyleDefinition`**

This class represents a character style definition, usually added to a *wxRichTextStyleSheet* (p. 1387).

### **Derived from**

*wxRichTextStyleDefinition* (p. 1376)

### **Include files**

<wx/richtext/richtextstyles.h>

### **Data structures**

**`wxRichTextCharacterStyleDefinition::wxRichTextCharacterStyleDefinition`**

**wxRichTextCharacterStyleDefinition(const wxString& name = wxEmptyString)**

Constructor.

**wxRichTextCharacterStyleDefinition::~~wxRichTextCharacterStyleDefinition**

**~wxRichTextCharacterStyleDefinition()**

Destructor.

## wxRichTextCtrl

wxRichTextCtrl provides a generic, ground-up implementation of a text control capable of showing multiple styles and images.

wxRichTextCtrl sends notification events: see *wxRichTextEvent* (p. 1346). It also sends the standard wxTextCtrl events wxEVT\_COMMAND\_TEXT\_ENTER and wxEVT\_COMMAND\_TEXT\_UPDATED, and wxTextUrlEvent when URL content is clicked.

For more information, see the *wxRichTextCtrl overview* (p. 2188).

### Derived from

wxTextCtrlBase

### Include files

<wx/richtext/richtextctrl.h>

### Data structures

## wxRichTextCtrl::wxRichTextCtrl

**wxRichTextCtrl()**

**wxRichTextCtrl(wxWindow\* parent, wxWindowID id = wxID\_ANY, const wxString& value = wxEmptyString, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxRE\_MULTILINE, const wxValidator& validator = wxDefaultValidator, const wxString& name = wxTextCtrlNameStr)**

Constructors.

## wxRichTextCtrl::~~wxRichTextCtrl

**~wxRichTextCtrl()**

Destructor.

**wxRichTextCtrl::AddImage****wxRichTextRange AddImage(const wxImage& *image*)**

Adds an image to the control's buffer.

**wxRichTextCtrl::AddParagraph****wxRichTextRange AddParagraph(const wxString& *text*)**

Adds a new paragraph of text to the end of the buffer.

**wxRichTextCtrl::AppendText****void AppendText(const wxString& *text*)**

Sets the insertion point to the end of the buffer and writes the text.

**wxRichTextCtrl::ApplyAlignmentToSelection****bool ApplyAlignmentToSelection(wxTextAttrAlignment *alignment*)**

Applies the given alignment to the selection (undoable).

For alignment values, see *wxTextAttrEx* (p. 1623).

**wxRichTextCtrl::ApplyBoldToSelection****bool ApplyBoldToSelection()**

Applies bold to the selection (undoable).

**wxRichTextCtrl::ApplyItalicToSelection****bool ApplyItalicToSelection()**

Applies italic to the selection (undoable).

**wxRichTextCtrl::ApplyStyle****bool ApplyStyle(wxRichTextStyleDefinition\* *def*)**

Applies the given style to the selection.

**wxRichTextCtrl::ApplyStyleSheet****bool ApplyStyleSheet(wxRichTextStyleSheet\* *sheet* = NULL)**

Applies the style sheet to the buffer, matching paragraph styles in the sheet against named styles in the buffer. This might be useful if the styles have changed. If *sheet* is

NULL, the sheet set with `SetStyleSheet` is used.

Currently this applies paragraph styles only.

### **wxRichTextCtrl::ApplyUnderlineToSelection**

**bool ApplyUnderlineToSelection()**

Applies underline to the selection (undoable).

### **wxRichTextCtrl::BatchingUndo**

**bool BatchingUndo() const**

Returns `true` if undo commands are being batched.

### **wxRichTextCtrl::BeginAlignment**

**bool BeginAlignment(wxTextAttrAlignment alignment)**

Begins using alignment

For alignment values, see *wxTextAttrEx* (p. 1623).

### **wxRichTextCtrl::BeginBatchUndo**

**bool BeginBatchUndo(const wxString& cmdName)**

Starts batching undo history for commands.

### **wxRichTextCtrl::BeginBold**

**bool BeginBold()**

Begins using bold.

### **wxRichTextCtrl::BeginCharacterStyle**

**bool BeginCharacterStyle(const wxString& characterStyle)**

Begins using the named character style.

### **wxRichTextCtrl::BeginFont**

**bool BeginFont(const wxFont& font)**

Begins using this font.

### **wxRichTextCtrl::BeginFontSize**

**bool BeginFontSize(int pointSize)**

Begins using the given point size.

**wxRichTextCtrl::BeginItalic**

**bool BeginItalic()**

Begins using italic.

**wxRichTextCtrl::BeginLeftIndent**

**bool BeginLeftIndent(int leftIndent, int leftSubIndent = 0)**

Begins applying a left indent and subindent in tenths of a millimetre.

The sub-indent is an offset from the left of the paragraph, and is used for all but the first line in a paragraph. A positive value will cause the first line to appear to the left of the subsequent lines, and a negative value will cause the first line to be indented relative to the subsequent lines.

wxRichTextBuffer uses indentation to render a bulleted item. The left indent is the distance between the margin and the bullet. The content of the paragraph, including the first line, starts at leftMargin + leftSubIndent. So the distance between the left edge of the bullet and the left of the actual paragraph is leftSubIndent.

**wxRichTextCtrl::BeginLineSpacing**

**bool BeginLineSpacing(int lineSpacing)**

Begins applying line spacing. *spacing* is a multiple, where 10 means single-spacing, 15 means 1.5 spacing, and 20 means double spacing. The following constants are defined for convenience:

```
#define wxTEXT_ATTR_LINE_SPACING_NORMAL      10
#define wxTEXT_ATTR_LINE_SPACING_HALF       15
#define wxTEXT_ATTR_LINE_SPACING_TWICE      20
```

**wxRichTextCtrl::BeginListStyle**

**bool BeginListStyle(const wxString& listStyle, int level=1, int number=1)**

Begins using a specified list style. Optionally, you can also pass a level and a number.

**wxRichTextCtrl::BeginNumberedBullet**

**bool BeginNumberedBullet(int bulletNumber, int leftIndent, int leftSubIndent, int bulletStyle = wxTEXT\_ATTR\_BULLET\_STYLE\_ARABIC|wxTEXT\_ATTR\_BULLET\_STYLE\_PERIOD)**

Begins a numbered bullet. This call will be needed for each item in the list, and the application should take care of incrementing the numbering.

*bulletNumber* is a number, usually starting with 1.

*leftIndent* and *leftSubIndent* are values in tenths of a millimetre.

*bulletStyle* is a bitlist of the following values:

#define wxTEXT_ATTR_BULLET_STYLE_NONE	0x00000000
#define wxTEXT_ATTR_BULLET_STYLE_ARABIC	0x00000001
#define wxTEXT_ATTR_BULLET_STYLE_LETTERS_UPPER	0x00000002
#define wxTEXT_ATTR_BULLET_STYLE_LETTERS_LOWER	0x00000004
#define wxTEXT_ATTR_BULLET_STYLE_ROMAN_UPPER	0x00000008
#define wxTEXT_ATTR_BULLET_STYLE_ROMAN_LOWER	0x00000010
#define wxTEXT_ATTR_BULLET_STYLE_SYMBOL	0x00000020
#define wxTEXT_ATTR_BULLET_STYLE_BITMAP	0x00000040
#define wxTEXT_ATTR_BULLET_STYLE_PARENTHESSES	0x00000080
#define wxTEXT_ATTR_BULLET_STYLE_PERIOD	0x00000100
#define wxTEXT_ATTR_BULLET_STYLE_STANDARD	0x00000200
#define wxTEXT_ATTR_BULLET_STYLE_RIGHT_PARENTHESIS	0x00000400
#define wxTEXT_ATTR_BULLET_STYLE_OUTLINE	0x00000800
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_LEFT	0x00000000
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_RIGHT	0x00001000
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_CENTRE	0x00002000

`wxRichTextBuffer` uses indentation to render a bulleted item. The left indent is the distance between the margin and the bullet. The content of the paragraph, including the first line, starts at `leftMargin + leftSubIndent`. So the distance between the left edge of the bullet and the left of the actual paragraph is `leftSubIndent`.

### **wxRichTextCtrl::BeginParagraphSpacing**

**bool** `BeginParagraphSpacing`(int *before*, int *after*)

Begins paragraph spacing; pass the before-paragraph and after-paragraph spacing in tenths of a millimetre.

### **wxRichTextCtrl::BeginParagraphStyle**

**bool** `BeginParagraphStyle`(const `wxString&` *paragraphStyle*)

Begins applying the named paragraph style.

### **wxRichTextCtrl::BeginRightIndent**

**bool** `BeginRightIndent`(int *rightIndent*)

Begins a right indent, specified in tenths of a millimetre.

### **wxRichTextCtrl::BeginStyle**



**bool BeginStyle(const wxTextAttrEx& style)**

Begins applying a style.

**wxRichTextCtrl::BeginSuppressUndo**

**bool BeginSuppressUndo()**

Starts suppressing undo history for commands.

**wxRichTextCtrl::BeginSymbolBullet**

**bool BeginSymbolBullet(wxChar symbol, int leftIndent, int leftSubIndent, int bulletStyle = wxTEXT\_ATTR\_BULLET\_STYLE\_SYMBOL)**

Begins applying a symbol bullet, using a character from the current font. See *BeginNumberedBullet* (p. 1319) for an explanation of how indentation is used to render the bulleted paragraph.

**wxRichTextCtrl::BeginTextColour**

**bool BeginTextColour(const wxColour& colour)**

Begins using this colour.

**wxRichTextCtrl::BeginUnderline**

**bool BeginUnderline()**

Begins using underlining.

**wxRichTextCtrl::BeginURL**

**bool BeginURL(const wxString& url, const wxString& characterStyle = wxEmptyString)**

Begins applying wxTEXT\_ATTR\_URL to the content. Pass a URL and optionally, a character style to apply, since it is common to mark a URL with a familiar style such as blue text with underlining.

**wxRichTextCtrl::CanCopy**

**bool CanCopy() const**

Returns `true` if selected content can be copied to the clipboard.

**wxRichTextCtrl::CanCut**

**bool CanCut() const**

Returns `true` if selected content can be copied to the clipboard and deleted.

### **wxRichTextCtrl::CanDeleteSelection**

**bool CanDeleteSelection() const**

Returns `true` if selected content can be deleted.

### **wxRichTextCtrl::CanPaste**

**bool CanPaste() const**

Returns `true` if the clipboard content can be pasted to the buffer.

### **wxRichTextCtrl::CanRedo**

**bool CanRedo() const**

Returns `true` if there is a command in the command history that can be redone.

### **wxRichTextCtrl::CanUndo**

**bool CanUndo() const**

Returns `true` if there is a command in the command history that can be undone.

### **wxRichTextCtrl::Clear**

**void Clear()**

Clears the buffer content, leaving a single empty paragraph. Cannot be undone.

### **wxRichTextCtrl::ClearListStyle**

**bool ClearListStyle(const wxRichTextRange& range, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO)**

**bool ClearListStyle(const wxRichTextRange& range, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO)**

Clears the list style from the given range, clearing list-related attributes and applying any named paragraph style associated with each paragraph.

*flags* is a bit list of the following:

- `wxRICHTEXT_SETSTYLE_WITH_UNDO`: specifies that this command will be undoable.

See also *wxRichTextCtrl::SetListStyle* (p. 1342), *wxRichTextCtrl::PromoteList* (p. 1339), *wxRichTextCtrl::NumberList* (p. 1336).

**wxRichTextCtrl::Command****void Command(wxCommandEvent& event)**

Sends the event to the control.

**wxRichTextCtrl::Copy****void Copy()**

Copies the selected content (if any) to the clipboard.

**wxRichTextCtrl::Create****bool Create(wxWindow\* parent, wxWindowID id = wxID\_ANY, const wxString& value = wxEmptyString, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxRE\_MULTILINE, const wxValidator& validator = wxDefaultValidator, const wxString& name = wxTextCtrlNameStr)**

Creates the underlying window.

**wxRichTextCtrl::Cut****void Cut()**

Copies the selected content (if any) to the clipboard and deletes the selection. This is undoable.

**wxRichTextCtrl::Delete****bool Delete(const wxRichTextRange& range)**

Deletes the content within the given range.

**wxRichTextCtrl::DeleteSelectedContent****bool DeleteSelectedContent(long\* newPos = NULL)**

Deletes content if there is a selection, e.g. when pressing a key. Returns the new caret position in *newPos*, or leaves it if there was no action. This is undoable.

**wxRichTextCtrl::DeleteSelection****void DeleteSelection()**

Deletes the content in the selection, if any. This is undoable.

**wxRichTextCtrl::DiscardEdits****void DiscardEdits()**

Sets the buffer's modified status to `false`, and clears the buffer's command history.

**wxRichTextCtrl::DoGetBestSize****wxSize DoGetBestSize() const**

Currently this simply returns `wxSize(10, 10)`.

**wxRichTextCtrl::EndAlignment****bool EndAlignment()**

Ends alignment.

**wxRichTextCtrl::EndAllStyles****bool EndAllStyles()**

Ends application of all styles in the current style stack.

**wxRichTextCtrl::EndBatchUndo****bool EndBatchUndo()**

Ends batching undo command history.

**wxRichTextCtrl::EndBold****bool EndBold()**

Ends using bold.

**wxRichTextCtrl::EndCharacterStyle****bool EndCharacterStyle()**

Ends application of a named character style.

**wxRichTextCtrl::EndFont****bool EndFont()**

Ends using a font.

**wxRichTextCtrl::EndFontSize****bool EndFontSize()**

Ends using a point size.

**wxRichTextCtrl::EndItalic****bool EndItalic()**

Ends using italic.

**wxRichTextCtrl::EndLeftIndent****bool EndLeftIndent()**

Ends left indent.

**wxRichTextCtrl::EndLineSpacing****bool EndLineSpacing()**

Ends line spacing.

**wxRichTextCtrl::EndListStyle****bool EndListStyle()**

Ends using a specified list style.

**wxRichTextCtrl::EndNumberedBullet****bool EndNumberedBullet()**

Ends application of a numbered bullet.

**wxRichTextCtrl::EndParagraphSpacing****bool EndParagraphSpacing()**

Ends paragraph spacing.

**wxRichTextCtrl::EndParagraphStyle****bool EndParagraphStyle()**

Ends application of a named character style.

**wxRichTextCtrl::EndRightIndent****bool EndRightIndent()**

Ends right indent.

**wxRichTextCtrl::EndStyle**

**bool EndStyle()**

Ends the current style.

**wxRichTextCtrl::EndSuppressUndo****bool EndSuppressUndo()**

Ends suppressing undo command history.

**wxRichTextCtrl::EndSymbolBullet****bool EndSymbolBullet()**

Ends applying a symbol bullet.

**wxRichTextCtrl::EndTextColour****bool EndTextColour()**

Ends applying a text colour.

**wxRichTextCtrl::EndUnderline****bool EndUnderline()**

End applying underlining.

**wxRichTextCtrl::EndURL****bool EndURL()**

Ends applying a URL.

**wxRichTextCtrl::ExtendSelection****bool ExtendSelection(long oldPosition, long newPosition, int flags)**

Helper function for extending the selection, returning `true` if the selection was changed. Selections are in caret positions.

**wxRichTextCtrl::FindNextWordPosition****long FindNextWordPosition(int direction = 1) const**

Helper function for finding the caret position for the next word. Direction is 1 (forward) or -1 (backwards).

**wxRichTextCtrl::Freeze**

**void Freeze()**

Call this function to prevent refresh and allow fast updates, and then *Thaw* (p. 1345) to refresh the control.

**wxRichTextCtrl::GetBasicStyle**

**const wxTextAttrEx& GetBasicStyle() const**

Gets the basic (overall) style. This is the style of the whole buffer before further styles are applied, unlike the default style, which only affects the style currently being applied (for example, setting the default style to bold will cause subsequently inserted text to be bold).

**wxRichTextCtrl::GetBuffer**

**const wxRichTextBuffer& GetBuffer() const**

**wxRichTextBuffer& GetBuffer()**

Returns the buffer associated with the control.

**wxRichTextCtrl::GetCaretPosition**

**long GetCaretPosition() const**

Returns the current caret position.

**wxRichTextCtrl::GetCaretPositionForIndex**

**bool GetCaretPositionForIndex(long position, wxRect& rect)**

Returns the caret height and position for the given character position

**wxRichTextCtrl::GetCommandProcessor**

**wxCommandProcessor\* GetCommandProcessor() const**

Gets the command processor associated with the control's buffer.

**wxRichTextCtrl::GetDefaultStyleEx**

**const wxTextAttrEx& GetDefaultStyleEx() const**

Returns the current default style, which can be used to change how subsequently inserted text is displayed. When wxTextAttrEx is merged with wxTextAttr, this function will become GetDefaultStyle.

**wxRichTextCtrl::GetDelayedLayoutThreshold**

**long GetDelayedLayoutThreshold() const**

Gets the size of the buffer beyond which layout is delayed during resizing. This optimizes sizing for large buffers. The default is 20000.

**wxRichTextCtrl::GetFilename****wxString GetFilename() const**

Gets the current filename associated with the control.

**wxRichTextCtrl::GetFirstVisiblePosition****long GetFirstVisiblePosition() const**

Returns the first visible position in the current view.

**wxRichTextCtrl::GetHandlerFlags****int GetHandlerFlags() const**

Returns flags that change the behaviour of loading or saving. See the documentation for each handler class to see what flags are relevant for each handler.

**wxRichTextCtrl::GetInsertionPoint****long GetInsertionPoint() const**

Returns the current insertion point.

**wxRichTextCtrl::GetLastPosition****wxTextPos GetLastPosition() const**

Returns the last position in the buffer.

**wxRichTextCtrl::GetLineLength****int GetLineLength(long *lineNo*) const**

Returns the length of the specified line in characters.

**wxRichTextCtrl::GetLineText****wxString GetLineText(long *lineNo*) const**

Returns the text for the given line.

**wxRichTextCtrl::GetLogicalPoint****wxPoint GetLogicalPoint(const wxPoint& *ptPhysical*) const**



Transforms physical window position to logical (unscrolled) position.

### **wxRichTextCtrl::GetNumberOfLines**

**int GetNumberOfLines() const**

Returns the number of lines in the buffer.

### **wxRichTextCtrl::GetPhysicalPoint**

**wxPoint GetPhysicalPoint(const wxPoint& ptLogical) const**

Transforms logical (unscrolled) position to physical window position.

### **wxRichTextCtrl::GetRange**

**wxString GetRange(long from, long to) const**

Gets the text for the given range.

The end point of range is specified as the last character position of the span of text, plus one.

### **wxRichTextCtrl::GetSelection**

**void GetSelection(long\* from, long\* to) const**

Returns the range of the current selection.

The end point of range is specified as the last character position of the span of text, plus one.

If the return values *from* and *to* are the same, there is no selection.

### **wxRichTextCtrl::GetSelectionRange**

**const wxRichTextRange& GetSelectionRange() const**

Returns the selection range in character positions. -1, -1 means no selection.

### **wxRichTextCtrl::GetStringSelection**

**wxString GetStringSelection() const**

Returns the text within the current selection range, if any.

### **wxRichTextCtrl::GetStyle**

**bool GetStyle(long position, wxRichTextAttr& style)**

**bool GetStyle(long position, wxTextAttrEx& style)**

**bool GetStyle(long position, wxTextAttr& style)**

Gets the attributes at the given position. The `wxRichTextAttr` version is generally more efficient because it does not use `wxFont` objects.

This function gets the combined style - that is, the style you see on the screen as a result of combining base style, paragraph style and character style attributes. To get the character or paragraph style alone, use *GetUncombinedStyle* (p. 1330).

**wxRichTextCtrl::GetStyleForRange****bool GetStyleForRange(const wxRichTextRange& range, wxRichTextAttr& style)****bool GetStyleForRange(const wxRichTextRange& range, wxTextAttrEx& style)**

Gets the attributes common to the specified range. Attributes that differ in value within the range will not be included in *style's* flags.

**wxRichTextCtrl::GetStyleSheet****wxRichTextStyleSheet\* GetStyleSheet() const**

Returns the style sheet associated with the control, if any. A style sheet allows named character and paragraph styles to be applied.

**wxRichTextCtrl::GetUncombinedStyle****bool GetUncombinedStyle(long position, wxRichTextAttr& style)****bool GetUncombinedStyle(long position, wxTextAttrEx& style)****bool GetUncombinedStyle(long position, wxTextAttr& style)**

Gets the attributes at the given position. The `wxRichTextAttr` version is generally more efficient because it does not use `wxFont` objects.

This function gets the *uncombined style* - that is, the attributes associated with the paragraph or character content, and not necessarily the combined attributes you see on the screen. To get the combined attributes, use *GetStyle* (p. 1329).

If you specify (any) paragraph attribute in *style's* flags, this function will fetch the paragraph attributes. Otherwise, it will return the character attributes.

**wxRichTextCtrl::GetValue****wxString GetValue() const**

Returns the content of the entire control as a string.

**wxRichTextCtrl::GetVisibleLineForCaretPosition****wxRichTextLine\* GetVisibleLineForCaretPosition(long caretPosition) const**

Internal helper function returning the line for the visible caret position. If the caret is shown at the very end of the line, it means the next character is actually on the following line. So this function gets the line we're expecting to find if this is the case.

### **wxRichTextCtrl::HasCharacterAttributes**

**bool HasCharacterAttributes(const wxRichTextRange& range, const wxTextAttrEx& style) const**

**bool HasCharacterAttributes(const wxRichTextRange& range, const wxRichTextAttr& style) const**

Test if this whole range has character attributes of the specified kind. If any of the attributes are different within the range, the test fails. You can use this to implement, for example, bold button updating. *style* must have flags indicating which attributes are of interest.

### **wxRichTextCtrl::HasParagraphAttributes**

**bool HasParagraphAttributes(const wxRichTextRange& range, const wxTextAttrEx& style) const**

**bool HasParagraphAttributes(const wxRichTextRange& range, const wxRichTextAttr& style) const**

Test if this whole range has paragraph attributes of the specified kind. If any of the attributes are different within the range, the test fails. You can use this to implement, for example, centering button updating. *style* must have flags indicating which attributes are of interest.

### **wxRichTextCtrl::HasSelection**

**bool HasSelection() const**

Returns `true` if there is a selection.

### **wxRichTextCtrl::HitTest**

**wxTextCtrlHitTestResult HitTest(const wxPoint& pt, long\* pos) const**

**wxTextCtrlHitTestResult HitTest(const wxPoint& pt, wxTextCoord\* col, wxTextCoord\* row) const**

Finds the character at the given position in pixels.

*pt* is in device coords (not adjusted for the client area origin nor for scrolling).

### **wxRichTextCtrl::Init**

**void Init()**

Initialises the members of the control.

### **wxRichTextCtrl::InitCommandEvent**

**void InitCommandEvent(wxCommandEvent& event) const**

Initialises the command event.

### **wxRichTextCtrl::IsDefaultStyleShowing**

**bool IsDefaultStyleShowing() const**

Returns `true` if the user has recently set the default style without moving the caret, and therefore the UI needs to reflect the default style and not the style at the caret.

Below is an example of code that uses this function to determine whether the UI should show that the current style is bold.

```
/// Is all of the selection bold?
bool wxRichTextCtrl::IsSelectionBold()
{
    if (HasSelection())
    {
        wxRichTextAttr attr;
        wxRichTextRange range = GetInternalSelectionRange();
        attr.SetFlags(wxTEXT_ATTR_FONT_WEIGHT);
        attr.SetFontWeight(wxBOLD);

        return HasCharacterAttributes(range, attr);
    }
    else
    {
        // If no selection, then we need to combine current style with
        default style
        // to see what the effect would be if we started typing.
        wxRichTextAttr attr;
        attr.SetFlags(wxTEXT_ATTR_FONT_WEIGHT);

        long pos = GetAdjustedCaretPosition(GetCaretPosition());
        if (GetStyle(pos, attr))
        {
            if (IsDefaultStyleShowing())
                wxRichTextApplyStyle(attr, GetDefaultStyleEx());
            return attr.GetFontWeight() == wxBOLD;
        }
    }
    return false;
}
```

See also *SetAndShowDefaultStyle* (p. 1341).

### **wxRichTextCtrl::IsEditable**

**bool IsEditable() const**

Returns `true` if the control is editable.

**wxRichTextCtrl::IsFrozen****bool IsFrozen() const**

Returns `true` if Freeze has been called without a Thaw.

**wxRichTextCtrl::IsModified****bool IsModified() const**

Returns `true` if the buffer has been modified.

**wxRichTextCtrl::IsMultiLine****bool IsMultiLine() const**

Returns `true` if the control is multiline.

**wxRichTextCtrl::IsPositionVisible****bool IsPositionVisible(long pos) const**

Returns `true` if the given position is visible on the screen.

**wxRichTextCtrl::IsSelectionAligned****bool IsSelectionAligned(wxTextAttrAlignment alignment) const**

Returns `true` if all of the selection is aligned according to the specified flag.

**wxRichTextCtrl::IsSelectionBold****bool IsSelectionBold() const**

Returns `true` if all of the selection is bold.

**wxRichTextCtrl::IsSelectionItalics****bool IsSelectionItalics() const**

Returns `true` if all of the selection is italic.

**wxRichTextCtrl::IsSelectionUnderlined****bool IsSelectionUnderlined() const**

Returns `true` if all of the selection is underlined.

**wxRichTextCtrl::IsSingleLine****bool IsSingleLine() const**

Returns `true` if the control is single-line. Currently `wxRichTextCtrl` does not support single-line editing.

**wxRichTextCtrl::KeyboardNavigate****bool KeyboardNavigate(int keyCode, int flags)**

Helper function implementing keyboard navigation.

**wxRichTextCtrl::LayoutContent****bool LayoutContent(bool onlyVisibleRect = false)**

Lays out the buffer, which must be done before certain operations, such as setting the caret position. This function should not normally be required by the application.

**wxRichTextCtrl::LineBreak****bool LineBreak()**

Inserts a line break at the current insertion point. A line break forces wrapping within a paragraph, and can be introduced by using this function, by appending the `wxChar` value `wxRichTextLineBreakChar` to text content, or by typing Shift-Return.

**wxRichTextCtrl::LoadFile****bool LoadFile(const wxString& file, int type = wxRICHTEXT\_TYPE\_ANY)**

Loads content into the control's buffer using the given type. If the specified type is `wxRICHTEXT_TYPE_ANY`, the type is deduced from the filename extension.

This function looks for a suitable *wxRichTextFileHandler* (p. 1350) object.

**wxRichTextCtrl::MarkDirty****void MarkDirty()**

Marks the buffer as modified.

**wxRichTextCtrl::MoveCaret****bool MoveCaret(long pos, bool showAtLineStart = false)**

Move the caret to the given character position.

**wxRichTextCtrl::MoveCaretBack**

**void MoveCaretBack(long oldPosition)**

Move the caret one visual step forward: this may mean setting a flag and keeping the same position if we're going from the end of one line to the start of the next, which may be the exact same caret position.

**wxRichTextCtrl::MoveCaretForward**

**void MoveCaretForward(long oldPosition)**

Move the caret one visual step forward: this may mean setting a flag and keeping the same position if we're going from the end of one line to the start of the next, which may be the exact same caret position.

**wxRichTextCtrl::MoveDown**

**bool MoveDown(int noLines = 1, int flags = 0)**

Moves the caret down.

**wxRichTextCtrl::MoveEnd**

**bool MoveEnd(int flags = 0)**

Moves to the end of the buffer.

**wxRichTextCtrl::MoveHome**

**bool MoveHome(int flags = 0)**

Moves to the start of the buffer.

**wxRichTextCtrl::MoveLeft**

**bool MoveLeft(int noPositions = 1, int flags = 0)**

Moves left.

**wxRichTextCtrl::MoveRight**

**bool MoveRight(int noPositions = 1, int flags = 0)**

Moves right.

**wxRichTextCtrl::MoveToLineEnd**

**bool MoveToLineEnd(int flags = 0)**

Moves to the end of the line.

**wxRichTextCtrl::MoveToLineStart****bool MoveToLineStart(int flags = 0)**

Moves to the start of the line.

**wxRichTextCtrl::MoveToParagraphEnd****bool MoveToParagraphEnd(int flags = 0)**

Moves to the end of the paragraph.

**wxRichTextCtrl::MoveToParagraphStart****bool MoveToParagraphStart(int flags = 0)**

Moves to the start of the paragraph.

**wxRichTextCtrl::MoveUp****bool MoveUp(int noLines = 1, int flags = 0)**

Moves up.

**wxRichTextCtrl::Newline****bool Newline()**

Inserts a new paragraph at the current insertion point. See also *wxRichTextCtrl::LineBreak* (p. 1334).

**wxRichTextCtrl::NumberList****bool NumberList(const wxRichTextRange& range, const wxRichTextListStyleDefinition\* style, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO, int startFrom = -1, int listLevel = -1)****bool Number(const wxRichTextRange& range, const wxString& styleName, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO, int startFrom = -1, int listLevel = -1)**

Numbers the paragraphs in the given range. Pass flags to determine how the attributes are set. Either the style definition or the name of the style definition (in the current sheet) can be passed.

*flags* is a bit list of the following:

- `wxRICHTEXT_SETSTYLE_WITH_UNDO`: specifies that this command will be undoable.
- `wxRICHTEXT_SETSTYLE_RENUMBER`: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.



- `wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL`: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also `wxRichTextCtrl::SetListStyle` (p. 1342), `wxRichTextCtrl::PromoteList` (p. 1339), `wxRichTextCtrl::ClearListStyle` (p. 1322).

### **wxRichTextCtrl::OnClear**

**void OnClear(wxCommandEvent& event)**

Standard handler for the `wxID_CLEAR` command.

### **wxRichTextCtrl::OnContextMenu**

**void OnContextMenu(wxContextMenuEvent& event)**

Shows a standard context menu with undo, redo, cut, copy, paste, clear, and select all commands.

### **wxRichTextCtrl::OnCopy**

**void OnCopy(wxCommandEvent& event)**

Standard handler for the `wxID_COPY` command.

### **wxRichTextCtrl::OnCut**

**void OnCut(wxCommandEvent& event)**

Standard handler for the `wxID_CUT` command.

### **wxRichTextCtrl::OnDropFiles**

**void OnDropFiles(wxDropFilesEvent& event)**

Loads the first dropped file.

### **wxRichTextCtrl::OnPaste**

**void OnPaste(wxCommandEvent& event)**

Standard handler for the `wxID_PASTE` command.

### **wxRichTextCtrl::OnRedo**

**void OnRedo(wxCommandEvent& event)**

Standard handler for the `wxID_REDO` command.

**wxRichTextCtrl::OnSelectAll****void OnSelectAll(wxCommandEvent& event)**

Standard handler for the wxID\_SELECTALL command.

**wxRichTextCtrl::OnUndo****void OnUndo(wxCommandEvent& event)**

Standard handler for the wxID\_PASTE command.

**wxRichTextCtrl::OnUpdateClear****void OnUpdateClear(wxUpdateUIEvent& event)**

Standard update handler for the wxID\_CLEAR command.

**wxRichTextCtrl::OnUpdateCopy****void OnUpdateCopy(wxUpdateUIEvent& event)**

Standard update handler for the wxID\_COPY command.

**wxRichTextCtrl::OnUpdateCut****void OnUpdateCut(wxUpdateUIEvent& event)**

Standard update handler for the wxID\_CUT command.

**wxRichTextCtrl::OnUpdatePaste****void OnUpdatePaste(wxUpdateUIEvent& event)**

Standard update handler for the wxID\_PASTE command.

**wxRichTextCtrl::OnUpdateRedo****void OnUpdateRedo(wxUpdateUIEvent& event)**

Standard update handler for the wxID\_REDO command.

**wxRichTextCtrl::OnUpdateSelectAll****void OnUpdateSelectAll(wxUpdateUIEvent& event)**

Standard update handler for the wxID\_SELECTALL command.

**wxRichTextCtrl::OnUpdateUndo**

**void OnUpdateUndo(wxUpdateUIEvent& event)**

Standard update handler for the wxID\_UNDO command.

**wxRichTextCtrl::PageDown**

**bool PageDown(int noPages = 1, int flags = 0)**

Moves one or more pages down.

**wxRichTextCtrl::PageUp**

**bool PageUp(int noPages = 1, int flags = 0)**

Moves one or more pages up.

**wxRichTextCtrl::PaintBackground**

**void PaintBackground(wxDC& dc)**

Paints the background.

**wxRichTextCtrl::Paste**

**void Paste()**

Pastes content from the clipboard to the buffer.

**wxRichTextCtrl::PositionCaret**

**void PositionCaret()**

Internal function to position the visible caret according to the current caret position.

**wxRichTextCtrl::PositionToXY**

**bool PositionToXY(long pos, long\* x, long\* y) const**

Converts a text position to zero-based column and line numbers.

**wxRichTextCtrl::PromoteList**

**bool PromoteList(int promoteBy, const wxRichTextRange& range, const wxRichTextListStyleDefinition\* style, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO, int listLevel = -1)**

**bool PromoteList(int promoteBy, const wxRichTextRange& range, const wxString& styleName, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO, int listLevel = -1)**

Promotes or demotes the paragraphs in the given range. A positive *promoteBy* produces a

smaller indent, and a negative number produces a larger indent. Pass flags to determine how the attributes are set. Either the style definition or the name of the style definition (in the current sheet) can be passed.

*flags* is a bit list of the following:

- `wxRICHTEXT_SETSTYLE_WITH_UNDO`: specifies that this command will be undoable.
- `wxRICHTEXT_SETSTYLE_RENUMBER`: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- `wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL`: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also `wxRichTextCtrl::SetListStyle` (p. 1342), See also `wxRichTextCtrl::SetListStyle` (p. 1336), `wxRichTextCtrl::ClearListStyle` (p. 1322).

### **wxRichTextCtrl::Redo**

**void Redo()**

Redoes the current command.

### **wxRichTextCtrl::Remove**

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

Removes the content in the specified range.

### **wxRichTextCtrl::Replace**

**void Replace(long from, long to, const wxString& value)**

Replaces the content in the specified range with the string specified by *value*.

### **wxRichTextCtrl::SaveFile**

**bool SaveFile(const wxString& file = wxEmptyString, int type = wxRICHTEXT\_TYPE\_ANY)**

Saves the buffer content using the given type. If the specified type is `wxRICHTEXT_TYPE_ANY`, the type is deduced from the filename extension.

This function looks for a suitable `wxRichTextFileHandler` (p. 1350) object.

### **wxRichTextCtrl::ScrollIntoView**

**bool ScrollIntoView(long position, int keyCode)**

Scrolls *position* into view. This function takes a caret position.

### **wxRichTextCtrl::SelectAll**

**void SelectAll()**

Selects all the text in the buffer.

### **wxRichTextCtrl::SelectNone**

**void SelectNone()**

Cancels any selection.

### **wxRichTextCtrl::SetAndShowDefaultStyle**

**void SetAndShowDefaultStyle(const wxRichTextAttr& attr)**

Sets *attr* as the default style and tells the control that the UI should reflect this attribute until the user moves the caret.

See also *IsDefaultStyleShowing* (p. 1332).

### **wxRichTextCtrl::SetBasicStyle**

**void SetBasicStyle(const wxRichTextAttr& style)**

**void SetBasicStyle(const wxTextAttrEx& style)**

Sets the basic (overall) style. This is the style of the whole buffer before further styles are applied, unlike the default style, which only affects the style currently being applied (for example, setting the default style to bold will cause subsequently inserted text to be bold).

### **wxRichTextCtrl::SetCaretPosition**

**void SetCaretPosition(long position, bool showAtLineStart = false)**

The caret position is the character position just before the caret. A value of -1 means the caret is at the start of the buffer.

### **wxRichTextCtrl::SetDefaultStyle**

**bool SetDefaultStyle(const wxTextAttrEx& style)**

Sets the current default style, which can be used to change how subsequently inserted text is displayed.

### **wxRichTextCtrl::SetDefaultStyleToCursorStyle**

**bool SetDefaultStyleToCursorStyle()**

Sets the default style to the style under the cursor.

### **wxRichTextCtrl::SetDelayedLayoutThreshold**

**void SetDelayedLayoutThreshold(long *threshold*)**

Sets the size of the buffer beyond which layout is delayed during resizing. This optimizes sizing for large buffers. The default is 20000.

### **wxRichTextCtrl::SetEditable**

**void SetEditable(bool *editable*)**

Makes the control editable, or not.

### **wxRichTextCtrl::SetFilename**

**void SetFilename(const wxString& *filename*)**

Sets the current filename.

### **wxRichTextCtrl::SetFont**

**bool SetFont(const wxFont& *font*)**

Sets the font, and also the basic and default attributes (see *SetDefaultStyle* (p. 1341)).

### **wxRichTextCtrl::SetHandlerFlags**

**void SetHandlerFlags(int *flags*)**

Sets flags that change the behaviour of loading or saving. See the documentation for each handler class to see what flags are relevant for each handler.

### **wxRichTextCtrl::SetInsertionPoint**

**void SetInsertionPoint(long *pos*)**

Sets the insertion point.

### **wxRichTextCtrl::SetInsertionPointEnd**

**void SetInsertionPointEnd()**

Sets the insertion point to the end of the text control.

### **wxRichTextCtrl::SetListStyle**

**bool SetListStyle(const wxRichTextRange& *range*, const wxRichTextListStyleDefinition\* *style*, int *flags* =**

*wxRICHTEXT\_SETSTYLE\_WITH\_UNDO*, **int** *startFrom* = -1, **int** *listLevel* = -1)

**bool** **SetListStyle**(const **wxRichTextRange**& *range*, const **wxString**& *styleName*, **int** *flags* = *wxRICHTEXT\_SETSTYLE\_WITH\_UNDO*, **int** *startFrom* = -1, **int** *listLevel* = -1)

Sets the list attributes for the given range, passing flags to determine how the attributes are set. Either the style definition or the name of the style definition (in the current sheet) can be passed.

*flags* is a bit list of the following:

- *wxRICHTEXT\_SETSTYLE\_WITH\_UNDO*: specifies that this command will be undoable.
- *wxRICHTEXT\_SETSTYLE\_RENUMBER*: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- *wxRICHTEXT\_SETSTYLE\_SPECIFY\_LEVEL*: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also *wxRichTextCtrl::NumberList* (p. 1336), *wxRichTextCtrl::PromoteList* (p. 1339), *wxRichTextCtrl::ClearListStyle* (p. 1322).

### **wxRichTextCtrl::SetSelection**

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

Sets the selection to the given range.

The end point of range is specified as the last character position of the span of text, plus one. So, for example, to set the selection for a character at position 5, use the range (5,6).

### **wxRichTextCtrl::SetSelectionRange**

**void** **SetSelectionRange**(const **wxRichTextRange**& *range*)

Sets the selection to the given range.

The end point of range is specified as the last character position of the span of text, plus one. So, for example, to set the selection for a character at position 5, use the range (5,6).

### **wxRichTextCtrl::SetStyle**

**bool** **SetStyle**(const **wxRichTextRange**& *range*, const **wxRichTextAttr**& *style*)

**bool** **SetStyle**(**long** *start*, **long** *end*, const **wxTextAttrEx**& *style*)

**bool** **SetStyle**(**long** *start*, **long** *end*, const **wxTextAttr**& *style*)

Sets the attributes for the given range. The *wxRichTextAttr* version is more efficient because it does not use *wxFont* objects.

The end point of range is specified as the last character position of the span of text, plus one. So, for example, to set the style for a character at position 5, use the range (5,6).

### **wxRichTextCtrl::SetStyleEx**

**bool SetStyleEx(const wxRichTextRange& range, const wxRichTextAttr& style, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO)**

**bool SetStyleEx(const wxRichTextRange& range, const wxTextAttrEx& style, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO)**

**bool SetStyleEx(long start, long end, const wxTextAttrEx& style, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO)**

Sets the attributes for the given range, passing flags to determine how the attributes are set. The wxRichTextAttr version is more efficient because it does not use wxFont objects.

The end point of range is specified as the last character position of the span of text, plus one. So, for example, to set the style for a character at position 5, use the range (5,6).

*flags* may contain a bit list of the following values:

- **wxRICHTEXT\_SETSTYLE\_NONE**: no style flag.
- **wxRICHTEXT\_SETSTYLE\_WITH\_UNDO**: specifies that this operation should be undoable.
- **wxRICHTEXT\_SETSTYLE\_OPTIMIZE**: specifies that the style should not be applied if the combined style at this point is already the style in question.
- **wxRICHTEXT\_SETSTYLE\_PARAGRAPHS\_ONLY**: specifies that the style should only be applied to paragraphs, and not the content. This allows content styling to be preserved independently from that of e.g. a named paragraph style.
- **wxRICHTEXT\_SETSTYLE\_CHARACTERS\_ONLY**: specifies that the style should only be applied to characters, and not the paragraph. This allows content styling to be preserved independently from that of e.g. a named paragraph style.
- **wxRICHTEXT\_SETSTYLE\_RESET**: resets (clears) the existing style before applying the new style.
- **wxRICHTEXT\_SETSTYLE\_REMOVE**: removes the specified style. Only the style flags are used in this operation.

### **wxRichTextCtrl::SetStyleSheet**

**void SetStyleSheet(wxRichTextStyleSheet\* styleSheet)**

Sets the style sheet associated with the control. A style sheet allows named character and paragraph styles to be applied.

### **wxRichTextCtrl::SetValue**



**void SetValue(const wxString& value)**

Replaces existing content with the given text.

**wxRichTextCtrl::SetupScrollbars**

**void SetupScrollbars(bool atTop = false)**

A helper function setting up scrollbars, for example after a resize.

**wxRichTextCtrl::ShowPosition**

**void ShowPosition(long pos)**

Scrolls the buffer so that the given position is in view.

**wxRichTextCtrl::SuppressingUndo**

**bool SuppressingUndo() const**

Returns `true` if undo history suppression is on.

**wxRichTextCtrl::Thaw**

**void Thaw()**

Call this function to end a Freeze and refresh the display.

**wxRichTextCtrl::Undo**

**void Undo()**

Undoes the command at the top of the command history, if there is one.

**wxRichTextCtrl::WordLeft**

**bool WordLeft(int noWords = 1, int flags = 0)**

Moves a number of words to the left.

**wxRichTextCtrl::WordRight**

**bool WordRight(int noWords = 1, int flags = 0)**

Move a nuber of words to the right.

**wxRichTextCtrl::WriteImage**

**bool WriteImage(const wxString& filename, int bitmapType)**

Loads an image from a file and writes it at the current insertion point.

**bool WriteImage(const wxRichTextImageBlock& imageBlock)**

Writes an image block at the current insertion point.

**bool WriteImage(const wxBitmap& bitmap, int bitmapType = wxBITMAP\_TYPE\_PNG)**

**bool WriteImage(const wxImage& image, int bitmapType = wxBITMAP\_TYPE\_PNG)**

Write a bitmap or image at the current insertion point. Supply an optional type to use for internal and file storage of the raw data.

### **wxRichTextCtrl::WriteText**

**void WriteText(const wxString& text)**

Writes text at the current position.

### **wxRichTextCtrl::XYToPosition**

**long XYToPosition(long x, long y) const**

Translates from column and line number to position.

## **wxRichTextEvent**

This is the event class for *wxRichTextCtrl* (p. 1316) notifications.

### **Event table macros**

To process a rich text event, use these event handler macros to direct input to a member function that takes a *wxRichTextEvent* argument.

**EVT\_RICHTEXT\_CHARACTER(id, func)**

Process a *wxEVT\_COMMAND\_RICHTEXT\_CHARACTER* event, generated when the user presses a character key. Valid event functions: *GetFlags*, *GetPosition*, *GetCharacter*.

**EVT\_RICHTEXT\_DELETE(id, func)**

Process a *wxEVT\_COMMAND\_RICHTEXT\_DELETE* event, generated when the user presses the backspace or delete key. Valid event functions: *GetFlags*, *GetPosition*.

**EVT\_RICHTEXT\_RETURN(id, func)**

Process a

	<code>wxEVT_COMMAND_RICHTEXT_RETURN</code> event, generated when the user presses the return key. Valid event functions: <code>GetFlags</code> , <code>GetPosition</code> .
<b><code>EVT_RICHTEXT_STYLE_CHANGED(id, func)</code></b>	Process a <code>wxEVT_COMMAND_RICHTEXT_STYLE_CHANGED</code> event, generated when styling has been applied to the control. Valid event functions: <code>GetPosition</code> , <code>GetRange</code> .
<b><code>EVT_RICHTEXT_STYLESHEET_CHANGED(id, func)</code></b>	Process a <code>wxEVT_COMMAND_RICHTEXT_STYLESHEET_CHANGING</code> event, generated when the control's stylesheet has changed, for example the user added, edited or deleted a style. Valid event functions: <code>GetRange</code> , <code>GetPosition</code> .
<b><code>EVT_RICHTEXT_STYLESHEET_REPLACING(id, func)</code></b>	Process a <code>wxEVT_COMMAND_RICHTEXT_STYLESHEET_REPLACING</code> event, generated when the control's stylesheet is about to be replaced, for example when a file is loaded into the control. Valid event functions: <code>Veto</code> , <code>GetOldStyleSheet</code> , <code>GetNewStyleSheet</code> .
<b><code>EVT_RICHTEXT_STYLESHEET_REPLACED(id, func)</code></b>	Process a <code>wxEVT_COMMAND_RICHTEXT_STYLESHEET_REPLACED</code> event, generated when the control's stylesheet has been replaced, for example when a file is loaded into the control. Valid event functions: <code>GetOldStyleSheet</code> , <code>GetNewStyleSheet</code> .
<b><code>EVT_RICHTEXT_CONTENT_INSERTED(id, func)</code></b>	Process a <code>wxEVT_COMMAND_RICHTEXT_CONTENT_INSERTED</code> event, generated when

content has been inserted into the control. Valid event functions: `GetPosition`, `GetRange`.

### **EVT\_RICHTEXT\_CONTENT\_DELETED(id, func)**

Process a `wxEVT_COMMAND_RICHTEXT_CONTENT_DELETED` event, generated when content has been deleted from the control. Valid event functions: `GetPosition`, `GetRange`.

### **EVT\_RICHTEXT\_BUFFER\_RESET(id, func)**

Process a `wxEVT_COMMAND_RICHTEXT_BUFFER_RESET` event, generated when the buffer has been reset by deleting all content. You can use this to set a default style for the first new paragraph.

**Derived from**  
*wxNotifyEvent* (p. 1146)

### **Include files**

`<wx/richtext/richtextctrl.h>`

### **Data structures**

## **wxRichTextEvent::wxRichTextEvent**

**wxRichTextEvent(const wxRichTextEvent& event)**

**wxRichTextEvent(wxEventType commandType = wxEVT\_NULL, int winid = 0)**

Constructors.

## **wxRichTextEvent::Clone**

**wxEvent\* Clone() const**

Clones the event.

## **wxRichTextEvent::GetCharacter**

**wxChar GetCharacter() const**

Returns the character pressed, within a `wxEVT_COMMAND_RICHTEXT_CHARACTER` event.

### **`wxRichTextEvent::GetFlags`**

**`int GetFlags() const`**

Returns flags indicating modifier keys pressed. Possible values are `wxRICHTEXT_CTRL_DOWN`, `wxRICHTEXT_SHIFT_DOWN`, and `wxRICHTEXT_ALT_DOWN`.

### **`wxRichTextEvent::GetNewStyleSheet`**

**`wxRichTextStyleSheet* GetNewStyleSheet() const`**

Returns the new style sheet. Can be used in a `wxEVT_COMMAND_RICHTEXT_STYLESHEET_CHANGING` or `wxEVT_COMMAND_RICHTEXT_STYLESHEET_CHANGED` event handler.

### **`wxRichTextEvent::GetOldStyleSheet`**

**`wxRichTextStyleSheet* GetOldStyleSheet() const`**

Returns the old style sheet. Can be used in a `wxEVT_COMMAND_RICHTEXT_STYLESHEET_CHANGING` or `wxEVT_COMMAND_RICHTEXT_STYLESHEET_CHANGED` event handler.

### **`wxRichTextEvent::GetPosition`**

**`long GetPosition() const`**

Returns the buffer position at which the event occurred.

### **`wxRichTextEvent::GetRange`**

**`wxRichTextRange GetRange() const`**

Gets the range for the current operation.

### **`wxRichTextEvent::SetCharacter`**

**`void SetCharacter(wxChar ch)`**

Sets the character variable.

### **`wxRichTextEvent::SetFlags`**

**`void SetFlags(int flags)`**

Sets flags indicating modifier keys pressed. Possible values are

`wxRICHTEXT_CTRL_DOWN`, `wxRICHTEXT_SHIFT_DOWN`, and `wxRICHTEXT_ALT_DOWN`.

### **`wxRichTextEvent::SetNewStyleSheet`**

**`void SetNewStyleSheet(wxRichTextStyleSheet* sheet)`**

Sets the new style sheet variable.

### **`wxRichTextEvent::SetOldStyleSheet`**

**`void SetOldStyleSheet(wxRichTextStyleSheet* sheet)`**

Sets the old style sheet variable.

### **`wxRichTextEvent::SetPosition`**

**`void SetPosition(long pos)`**

Sets the buffer position variable.

### **`wxRichTextEvent::SetRange`**

**`void SetRange(const wxRichTextRange& range)`**

Sets the range variable.

## **`wxRichTextFileHandler`**

This is the base class for file handlers, for loading and/or saving content associated with a *wxRichTextBuffer* (p. 1298).

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/richtext/richtextbuffer.h>

### **Data structures**

### **`wxRichTextFileHandler::wxRichTextFileHandler`**

**`wxRichTextFileHandler(const wxString& name = wxEmptyString, const wxString& ext = wxEmptyString, int type = 0)`**

Constructor.

**wxRichTextFileHandler::CanHandle****bool CanHandle(const wxString& filename) const**

Override this function and return `true` if this handler can we handle *filename*. By default, this function checks the extension.

**wxRichTextFileHandler::CanLoad****bool CanLoad() const**

Override and return `true` if this handler can load content.

**wxRichTextFileHandler::CanSave****bool CanSave() const**

Override and return `true` if this handler can save content.

**wxRichTextFileHandler::DoLoadFile****bool DoLoadFile(wxRichTextBuffer\* buffer, wxInputStream& stream)**

Override to load content from *stream* into *buffer*.

**wxRichTextFileHandler::DoSaveFile****bool DoSaveFile(wxRichTextBuffer\* buffer, wxOutputStream& stream)**

Override to save content to *stream* from *buffer*.

**wxRichTextFileHandler::GetEncoding****const wxString& GetEncoding() const**

Returns the encoding associated with the handler (if any).

**wxRichTextFileHandler::GetExtension****wxString GetExtension() const**

Returns the extension associated with the handler.

**wxRichTextFileHandler::GetFlags****int GetFlags() const**

Returns flags that change the behaviour of loading or saving. See the documentation for each handler class to see what flags are relevant for each handler.

**wxRichTextFileHandler::GetName****wxString GetName() const**

Returns the name of the handler.

**wxRichTextFileHandler::GetType****int GetType() const**

Returns the type of the handler.

**wxRichTextFileHandler::IsVisible****bool IsVisible() const**

Returns `true` if this handler should be visible to the user.

**wxRichTextFileHandler::LoadFile****bool LoadFile(wxRichTextBuffer\* buffer, wxInputStream& stream)****bool LoadFile(wxRichTextBuffer\* buffer, const wxString& filename)**

Loads content from a stream or file. Not all handlers will implement file loading.

**wxRichTextFileHandler::SaveFile****bool SaveFile(wxRichTextBuffer\* buffer, wxOutputStream& stream)****bool SaveFile(wxRichTextBuffer\* buffer, const wxString& filename)**

Saves content to a stream or file. Not all handlers will implement file saving.

**wxRichTextFileHandler::SetEncoding****void SetEncoding(const wxString& encoding)**

Sets the encoding to use when saving a file. If empty, a suitable encoding is chosen.

**wxRichTextFileHandler::SetExtension****void SetExtension(const wxString& ext)**

Sets the default extension to recognise.

**wxRichTextFileHandler::SetFlags****void SetFlags(int flags)**

Sets flags that change the behaviour of loading or saving. See the documentation for each



handler class to see what flags are relevant for each handler.

You call this function directly if you are using a file handler explicitly (without going through the text control or buffer LoadFile/SaveFile API). Or, you can call the control or buffer's SetHandlerFlags function to set the flags that will be used for subsequent load and save operations.

### **wxRichTextFileHandler::SetName**

**void SetName(const wxString& name)**

Sets the name of the handler.

### **wxRichTextFileHandler::SetType**

**void SetType(int type)**

Sets the handler type.

### **wxRichTextFileHandler::SetVisible**

**void SetVisible(bool visible)**

Sets whether the handler should be visible to the user (via the application's load and save dialogs).

## **wxRichTextFormattingDialog**

This dialog allows the user to edit a character and/or paragraph style.

In the constructor, specify the pages that will be created. Use GetStyle to retrieve the common style for a given range, and then use ApplyStyle to apply the user-selected formatting to a control. For example:

```
wxRichTextRange range;
if (m_richTextCtrl->HasSelection())
    range = m_richTextCtrl->GetSelectionRange();
else
    range = wxRichTextRange(0,
m_richTextCtrl->GetLastPosition()+1);

int pages =
wxRICHTEXT_FORMAT_FONT|wxRICHTEXT_FORMAT_INDENTS_SPACING|wxRICHTEXT
T_FORMAT_TABS|wxRICHTEXT_FORMAT_BULLETS;

wxRichTextFormattingDialog formatDlg(pages, this);
formatDlg.GetStyle(m_richTextCtrl, range);

if (formatDlg.ShowModal() == wxID_OK)
{
    formatDlg.ApplyStyle(m_richTextCtrl, range);
}
```

**Derived from**

*wxPropertySheetDialog* (p. 1232)

**Include files**

<wx/richtext/richtextformatdlg.h>

**Data structures**

The following flags passed to the dialog constructor indicate the pages to be created:

```
#define wxRICHTEXT_FORMAT_STYLE_EDITOR    0x0001
#define wxRICHTEXT_FORMAT_FONT           0x0002
#define wxRICHTEXT_FORMAT_TABS           0x0004
#define wxRICHTEXT_FORMAT_BULLETS        0x0008
#define wxRICHTEXT_FORMAT_INDENTS_SPACING 0x0010
```

**wxRichTextFormattingDialog::wxRichTextFormattingDialog**

**wxRichTextFormattingDialog**(*long flags*, *wxWindow\** *parent*, **const wxString&** *title* = *\_("Formatting")*, **wxWindowID** *id* = *wxID\_ANY*, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *sz* = *wxDefaultSize*, **long** *style* = *wxDEFAULT\_DIALOG\_STYLE*)

**wxRichTextFormattingDialog**()

Constructors.

**Parameters**

*flags*

The pages to show.

*parent*

The dialog's parent.

*id*

The dialog's identifier.

*title*

The dialog's caption.

*pos*

The dialog's position.

*size*

The dialog's size.

*style*

The dialog's window style.

### **wxRichTextFormattingDialog::~wxRichTextFormattingDialog**

**~wxRichTextFormattingDialog()**

Destructor.

### **wxRichTextFormattingDialog::ApplyStyle**

**bool ApplyStyle(wxRichTextCtrl\* ctrl, const wxRichTextRange& range, int flags = wxRICHTEXT\_SETSTYLE\_WITH\_UNDO|wxRICHTEXT\_SETSTYLE\_OPTIMIZE)**

Apply attributes to the given range, only changing attributes that need to be changed.

### **wxRichTextFormattingDialog::Create**

**bool Create(long flags, wxWindow\* parent, const wxString& title, wxWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& sz = wxDefaultSize, long style = wxDEFAULT\_DIALOG\_STYLE)**

Creation: see *the constructor* (p. 1353) for details about the parameters.

### **wxRichTextFormattingDialog::GetAttributes**

**const wxTextAttrEx& GetAttributes() const**

**wxTextAttrEx& GetAttributes()**

Gets the attributes being edited.

### **wxRichTextFormattingDialog::GetDialog**

**wxRichTextFormattingDialog\* GetDialog(wxWindow\* win)**

Helper for pages to get the top-level dialog.

### **wxRichTextFormattingDialog::GetDialogAttributes**

**wxTextAttrEx\* GetDialogAttributes(wxWindow\* win)**

Helper for pages to get the attributes.

### **wxRichTextFormattingDialog::GetDialogStyleDefinition**

**wxRichTextStyleDefinition\* GetDialogStyleDefinition(wxWindow\* win)**

Helper for pages to get the style.

### **wxRichTextFormattingDialog::GetFormattingDialogFactory**

**wxRichTextFormattingDialogFactory\* GetFormattingDialogFactory()**

Returns the object to be used to customize the dialog and provide pages.

### **wxRichTextFormattingDialog::GetImageList**

**wxImageList\* GetImageList() const**

Returns the image list associated with the dialog, used for example if showing the dialog as a toolbook.

### **wxRichTextFormattingDialog::GetStyle**

**bool GetStyle(wxRichTextCtrl\* ctrl, const wxRichTextRange& range)**

Gets common attributes from the given range and calls SetAttributes. Attributes that do not have common values in the given range will be omitted from the style's flags.

### **wxRichTextFormattingDialog::GetStyleDefinition**

**wxRichTextStyleDefinition\* GetStyleDefinition() const**

Gets the associated style definition, if any.

### **wxRichTextFormattingDialog::GetStyleSheet**

**wxRichTextStyleSheet\* GetStyleSheet() const**

Gets the associated style sheet, if any.

### **wxRichTextFormattingDialog::SetAttributes**

**void SetAttributes(const wxTextAttrEx& attr)**

Sets the attributes to be edited.

### **wxRichTextFormattingDialog::SetFormattingDialogFactory**

**void SetFormattingDialogFactory(wxRichTextFormattingDialogFactory\* factory)**

Sets the formatting factory object to be used for customization and page creation. It deletes the existing factory object.

### **wxRichTextFormattingDialog::SetImageList**

**void SetImageList(wxImageList\* imageList)**

Sets the image list associated with the dialog's property sheet.

### **wxRichTextFormattingDialog::SetStyle**

**bool SetStyle(const wxTextAttrEx& style, bool update = true)**

Sets the attributes and optionally updates the display, if *update* is `true`.

### **wxRichTextFormattingDialog::SetStyleDefinition**

**bool SetStyleDefinition(const wxRichTextStyleDefinition& styleDef,  
wxRichTextStyleSheet\* sheet, bool update = true)**

Sets the style definition and optionally update the display, if *update* is `true`.

### **wxRichTextFormattingDialog::UpdateDisplay**

**bool UpdateDisplay()**

Updates the display.

## **wxRichTextFormattingDialogFactory**

This class provides pages for `wxRichTextFormattingDialog`, and allows other customization of the dialog. A default instance of this class is provided automatically. If you wish to change the behaviour of the formatting dialog (for example add or replace a page), you may derive from this class, override one or more functions, and call the static function `wxRichTextFormattingDialog::SetFormattingDialogFactory`. **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/richtext/richtextformatdlg.h>

### **wxRichTextFormattingDialogFactory::wxRichTextFormattingDialogFactory**

**wxRichTextFormattingDialogFactory()**

Constructor.

### **wxRichTextFormattingDialogFactory::~wxRichTextFormattingDialogFactory**

**~wxRichTextFormattingDialogFactory()**

Destructor.

**wxRichTextFormattingDialogFactory::CreateButtons****virtual bool CreateButtons(wxRichTextFormattingDialog\* *dialog*)**

Creates the main dialog buttons.

**wxRichTextFormattingDialogFactory::CreatePage****virtual wxPanel\* CreatePage(int *page*, wxString& *title*,  
wxRichTextFormattingDialog\* *dialog*)**

Creates a page, given a page identifier.

**wxRichTextFormattingDialogFactory::CreatePages****virtual bool CreatePages(long *pages*, wxRichTextFormattingDialog\* *dialog*)**

Creates all pages under the dialog's book control, also calling AddPage.

**wxRichTextFormattingDialogFactory::GetPageld****virtual int GetPageld(int *i*) const**

Enumerate all available page identifiers.

**wxRichTextFormattingDialogFactory::GetPageldCount****virtual int GetPageldCount() const**

Gets the number of available page identifiers.

**wxRichTextFormattingDialogFactory::GetPagelImage****virtual int GetPagelImage(int *id*) const**

Gets the image index for the given page identifier.

**wxRichTextFormattingDialogFactory::SetSheetStyle****virtual bool SetSheetStyle(wxRichTextFormattingDialog\* *dialog*)**

Set the property sheet style, called at the start of wxRichTextFormattingDialog::Create.

**wxRichTextFormattingDialogFactory::ShowHelp****virtual bool ShowHelp(int *page*, wxRichTextFormattingDialog\* *dialog*)**

Invokes help for the dialog.

## wxRichTextHeaderFooterData

This class represents header and footer data to be passed to the *wxRichTextPrinting* (p. 1368) and *wxRichTextPrintout* (p. 1371) classes.

Headers and footers can be specified independently for odd, even or both page sides. Different text can be specified for left, centre and right locations on the page, and the font and text colour can also be specified. You can specify the following keywords in header and footer text, which will be substituted for the actual values during printing and preview.

- @DATE@: the current date.
- @PAGESCNT@: the total number of pages.
- @PAGENUM@: the current page number.
- @TIME@: the current time.
- @TITLE@: the title of the document, as passed to the *wxRichTextPrinting* or *wxRichTextLayout* constructor. **Derived from**

*wxObject* (p. 1148)

### Include files

<wx/richtex/richtextprint.h>

### Data structures

These are the header and footer page identifiers, passed to functions such as *SetFooterText* to specify the odd or even page for the text:

```
enum wxRichTextOddEvenPage {
    wxRICHTEXT_PAGE_ODD,
    wxRICHTEXT_PAGE_EVEN,
    wxRICHTEXT_PAGE_ALL,
}
```

These are the location identifiers for passing to functions such as *SetFooterText*, to specify whether the text is on the left, centre or right of the page:

```
enum wxRichTextPageLocation {
    wxRICHTEXT_PAGE_LEFT,
    wxRICHTEXT_PAGE_CENTRE,
    wxRICHTEXT_PAGE_RIGHT
}
```

## wxRichTextHeaderFooterData::wxRichTextHeaderFooterData

**wxRichTextHeaderFooterData()**

**wxRichTextHeaderFooterData(const wxRichTextHeaderFooterData& data)**

Constructors.

### **wxRichTextHeaderFooterData::Clear**

**void Clear()**

Clears all text.

### **wxRichTextHeaderFooterData::Copy**

**void Copy(const wxRichTextHeaderFooterData& data)**

Copies the data.

### **wxRichTextHeaderFooterData::GetFont**

**const wxFont& GetFont() const**

Returns the font specified for printing the header and footer.

### **wxRichTextHeaderFooterData::GetFooterMargin**

**int GetFooterMargin() const**

Returns the margin between the text and the footer.

### **wxRichTextHeaderFooterData::GetFooterText**

**wxString GetFooterText(wxRichTextOddEvenPage page =  
wxRICHTEXT\_PAGE\_EVEN, wxRichTextPageLocation location =  
wxRICHTEXT\_PAGE\_CENTRE) const**

Returns the footer text on odd or even pages, and at a given position on the page (left, centre or right).

### **wxRichTextHeaderFooterData::GetHeaderMargin**

**int GetHeaderMargin() const**

Returns the margin between the text and the header.

### **wxRichTextHeaderFooterData::GetHeaderText**

**wxString GetHeaderText(wxRichTextOddEvenPage page =  
wxRICHTEXT\_PAGE\_EVEN, wxRichTextPageLocation location =  
wxRICHTEXT\_PAGE\_CENTRE) const**

Returns the header text on odd or even pages, and at a given position on the page (left, centre or right).



**wxRichTextHeaderFooterData::GetShowOnFirstPage****bool GetShowOnFirstPage() const**

Returns `true` if the header and footer will be shown on the first page.

**wxRichTextHeaderFooterData::GetText****wxString GetText(int headerFooter, wxRichTextOddEvenPage page,  
wxRichTextPageLocation location) const**

Helper function for getting the header or footer text, odd or even pages, and at a given position on the page (left, centre or right).

**wxRichTextHeaderFooterData::GetTextColour****const wxColour& GetTextColour() const**

Returns the text colour for drawing the header and footer.

**wxRichTextHeaderFooterData::Init****void Init()**

Initialises the object.

**wxRichTextHeaderFooterData::SetFont****void SetFont(const wxFont& font)**

Sets the font for drawing the header and footer.

**wxRichTextHeaderFooterData::SetFooterText****void SetFooterText(const wxString& text, wxRichTextOddEvenPage page =  
wxRICHTEXT\_PAGE\_ALL, wxRichTextPageLocation location =  
wxRICHTEXT\_PAGE\_CENTRE)**

Sets the footer text on odd or even pages, and at a given position on the page (left, centre or right).

**wxRichTextHeaderFooterData::SetHeaderText****void SetHeaderText(const wxString& text, wxRichTextOddEvenPage page =  
wxRICHTEXT\_PAGE\_ALL, wxRichTextPageLocation location =  
wxRICHTEXT\_PAGE\_CENTRE)**

Sets the header text on odd or even pages, and at a given position on the page (left, centre or right).

**wxRichTextHeaderFooterData::SetMargins****void SetMargins**(int *headerMargin*, int *footerMargin*)

Sets the margins between text and header or footer, in tenths of a millimeter.

**wxRichTextHeaderFooterData::SetShowOnFirstPage****void SetShowOnFirstPage**(bool *showOnFirstPage*)

Pass `true` to show the header or footer on first page (the default).

**wxRichTextHeaderFooterData::SetText****void SetText**(const wxString& *text*, int *headerFooter*, wxRichTextOddEvenPage *page*, wxRichTextPageLocation *location*)

Helper function for setting the header or footer text, odd or even pages, and at a given position on the page (left, centre or right).

**wxRichTextHeaderFooterData::SetTextColour****void SetTextColour**(const wxColour& *col*)

Sets the text colour for drawing the header and footer.

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

Assignment operator.

**wxRichTextHTMLHandler**

Handles HTML output (only) for *wxRichTextCtrl* (p. 1316) content.

The most flexible way to use this class is to create a temporary object and call its functions directly, rather than use *wxRichTextBuffer::SaveFile* (p. 1313) or *wxRichTextCtrl::SaveFile* (p. 1340).

Image handling requires a little extra work from the application, to choose an appropriate image format for the target HTML viewer and to clean up the temporary images later. If you are planning to load the HTML into a standard web browser, you can specify the handler flag `wxRICHTEXT_HANDLER_SAVE_IMAGES_TO_BASE64` (the default) and no extra work is required: the images will be written with the HTML.

However, if you want wxHTML compatibility, you will need to use `wxRICHTEXT_HANDLER_SAVE_IMAGES_TO_MEMORY` or `wxRICHTEXT_HANDLER_SAVE_IMAGES_TO_FILES`. In this case, you must either call *DeleteTemporaryImages* (p. 1363) before the next load operation, or you must store the

image locations and delete them yourself when appropriate. You can call *GetTemporaryImageLocations* (p. 1364) to get the array of temporary image names.

### Handler flags

The following flags can be used with this handler, via the handler's *SetFlags* function or the buffer or control's *SetHandlerFlags* function:

**wxRICHTEXT\_HANDLER\_SAVE\_IMAGES\_TO\_MEMORY** Images are saved to the memory filesystem: suitable for showing wxHTML windows.

**wxRICHTEXT\_HANDLER\_SAVE\_IMAGES\_TO\_FILES** Images are saved to temporary files: suitable for showing in wxHTML windows.

**wxRICHTEXT\_HANDLER\_SAVE\_IMAGES\_TO\_BASE64** Images are written with the HTML files in Base 64 format: suitable for showing in web browsers.

**wxRICHTEXT\_HANDLER\_NO\_HEADER\_FOOTER** Don't include header and footer tags (HTML, HEAD, BODY), so that the HTML can be used as part of a larger document.

### Derived from

*wxRichTextFileHandler* (p. 1350)

### Include files

<wx/richtext/richtexthtml.h>

### Data structures

#### **wxRichTextHTMLHandler::wxRichTextHTMLHandler**

**wxRichTextHTMLHandler(const wxString& name = wxT("HTML"), const wxString& ext = wxT("html"), int type = wxRICHTEXT\_TYPE\_HTML)**

Constructor.

#### **wxRichTextHTMLHandler::ClearTemporaryImageLocations**

**void ClearTemporaryImageLocations()**

Clears the image locations generated by the last operation.

#### **wxRichTextHTMLHandler::DeleteTemporaryImages**

**bool DeleteTemporaryImages()**

Deletes the in-memory or temporary files generated by the last operation.

**bool DeleteTemporaryImages(int flags, const wxArrayString& imageLocations)**

Delete the in-memory or temporary files generated by the last operation. This is a static function that can be used to delete the saved locations from an earlier operation, for example after the user has viewed the HTML file.

**wxRichTextHTMLHandler::DoSaveFile**

**bool DoSaveFile(wxRichTextBuffer\* buffer, wxOutputStream& stream)**

Saves the buffer content to the HTML stream.

**wxRichTextHTMLHandler::GetFontSizeMapping**

**wxArrayInt GetFontSizeMapping()**

Returns the mapping for converting point sizes to HTML font sizes.

**wxRichTextHTMLHandler::GetTempDir**

**const wxString& GetTempDir() const**

Returns the directory used to store temporary image files.

**wxRichTextHTMLHandler::GetTemporaryImageLocations**

**const wxArrayString& GetTemporaryImageLocations() const**

Returns the image locations for the last operation.

**wxRichTextHTMLHandler::SetFileCounter**

**void SetFileCounter(int counter)**

Reset the file counter, in case, for example, the same names are required each time

**wxRichTextHTMLHandler::SetFontSizeMode**

**void SetFontSizeMapping(const wxArrayInt& fontSizeMapping)**

Sets the mapping for converting point sizes to HTML font sizes. There should be 7 elements, one for each HTML font size, each element specifying the maximum point size for that HTML font size.

For example:

```
wxArrayInt fontSizeMapping;  
fontSizeMapping.Add(7);  
fontSizeMapping.Add(9);
```

```
fontSizeMapping.Add(11);
fontSizeMapping.Add(12);
fontSizeMapping.Add(14);
fontSizeMapping.Add(22);
fontSizeMapping.Add(100);

htmlHandler.SetFontSizeMapping(fontSizeMapping);
```

### **wxRichTextHTMLHandler::SetTempDir**

**void SetTempDir(const wxString& tempDir)**

Sets the directory for storing temporary files. If empty, the system temporary directory will be used.

### **wxRichTextHTMLHandler::SetTemporaryImageLocations**

**void SetTemporaryImageLocations(const wxArrayString& locations)**

Sets the list of image locations generated by the last operation.

## **wxRichTextListStyleDefinition**

This class represents a list style definition, usually added to a *wxRichTextStyleSheet* (p. 1387).

The class inherits paragraph attributes from *wxRichTextStyleParagraphDefinition*, and adds 10 further attribute objects, one for each level of a list. When applying a list style to a paragraph, the list style's base and appropriate level attributes are merged with the paragraph's existing attributes.

You can apply a list style to one or more paragraphs using *wxRichTextCtrl::SetListStyle* (p. 1342). You can also use the functions *wxRichTextCtrl::NumberList* (p. 1336), *wxRichTextCtrl::PromoteList* (p. 1339) and *wxRichTextCtrl::ClearListStyle* (p. 1322). As usual, there are *wxRichTextBuffer* versions of these functions so that you can apply them directly to a buffer without requiring a control.

### **Derived from**

*wxRichTextParagraphStyleDefinition* (p. 1367)

### **Include files**

<wx/richtext/richtextstyles.h>

### **Data structures**

### **wxRichTextListStyleDefinition::wxRichTextListStyleDefinition**

**wxRichTextListStyleDefinition(const wxString& name = wxEmptyString)**

Constructor.

**wxRichTextListStyleDefinition::~~wxRichTextListStyleDefinition**

**~wxRichTextListStyleDefinition()**

Destructor.

**wxRichTextListStyleDefinition::CombineWithParagraphStyle**

**wxRichTextAttr CombineWithParagraphStyle(int indent, const wxRichTextAttr& paraStyle, wxRichTextStyleSheet\* styleSheet = NULL)**

This function combines the given paragraph style with the list style's base attributes and level style matching the given indent, returning the combined attributes. If *styleSheet* is specified, the base style for this definition will also be included in the result.

**wxRichTextListStyleDefinition::FindLevelForIndent**

**int FindLevelForIndent(int indent) const**

This function finds the level (from 0 to 9) whose indentation attribute mostly closely matches *indent* (expressed in tenths of a millimetre).

**wxRichTextListStyleDefinition::GetCombinedStyle**

**wxRichTextAttr GetCombinedStyle(int indent, wxRichTextStyleSheet\* styleSheet = NULL) const**

This function combines the list style's base attributes and the level style matching the given indent, returning the combined attributes. If *styleSheet* is specified, the base style for this definition will also be included in the result.

**wxRichTextListStyleDefinition::GetCombinedStyleForLevel**

**wxRichTextAttr GetCombinedStyleLevel(int level, wxRichTextStyleSheet\* styleSheet = NULL) const**

This function combines the list style's base attributes and the style for the specified level, returning the combined attributes. If *styleSheet* is specified, the base style for this definition will also be included in the result.

**wxRichTextListStyleDefinition::GetLevelAttributes**

**const wxRichTextAttr\* GetLevelAttributes(int level) const**

Returns the style for the given level. *level* is a number between 0 and 9.

**wxRichTextListStyleDefinition::GetLevelCount****int GetLevelCount() const**

Returns the number of levels. This is hard-wired to 10.

Returns the style for the given level. *level* is a number between 0 and 9.

**wxRichTextListStyleDefinition::IsNumbered****int IsNumbered(int level) const**

Returns `true` if the given level has numbered list attributes.

**wxRichTextListStyleDefinition::SetLevelAttributes****void SetLevelAttributes(int level, const wxRichTextAttr& attr)****void SetLevelAttributes(int level, int leftIndent, int leftSubIndent, int bulletStyle, const wxString& bulletSymbol = wxEmptyString)**

Sets the style for the given level. *level* is a number between 0 and 9.

The first and most flexible form uses a `wxRichTextAttr` object, while the second form is for convenient setting of the most commonly-used attributes.

**wxRichTextParagraphStyleDefinition**

This class represents a paragraph style definition, usually added to a *wxRichTextStyleSheet* (p. 1387).

**Derived from**

*wxRichTextStyleDefinition* (p. 1376)

**Include files**

<wx/richtext/richtextstyles.h>

**Data structures****wxRichTextParagraphStyleDefinition::wxRichTextParagraphStyleDefinition****wxRichTextParagraphStyleDefinition(const wxString& name = wxEmptyString)**

Constructor.

**wxRichTextParagraphStyleDefinition::~~wxRichTextParagraphStyleDefinition**

**~wxRichTextParagraphStyleDefinition()**

Destructor.

**wxRichTextParagraphStyleDefinition::GetNextStyle**

**const wxString& GetNextStyle() const**

Returns the style that should normally follow this style.

**wxRichTextParagraphStyleDefinition::SetNextStyle**

**void SetNextStyle(const wxString& name)**

Sets the style that should normally follow this style.

## wxRichTextPrinting

This class provides a simple interface for performing *wxRichTextBuffer* (p. 1298) printing and previewing. It uses *wxRichTextPrintout* (p. 1371) for layout and rendering. **Derived from**

*wxObject* (p. 1148)

### Include files

<wx/richtext/richtextprint.h>

### Data structures

**wxRichTextPrinting::wxRichTextPrinting**

**wxRichTextPrinting(const wxString& name = wxT("Printing"), wxWindow\* parentWindow = NULL)**

Constructor. Optionally pass a title to be used in the preview frame and printing wait dialog, and also a parent window for these windows.

**wxRichTextPrinting::GetFooterText**

**wxString GetFooterText(wxRichTextOddEvenPage page = wxRICHTEXT\_PAGE\_EVEN, wxRichTextPageLocation location = wxRICHTEXT\_PAGE\_CENTRE) const**

A convenience function to get the footer text. See *wxRichTextHeaderFooterData* (p. 1359) for details.

**wxRichTextPrinting::GetHeaderFooterData**



**const wxRichTextHeaderFooterData& GetHeaderFooterData() const**

Returns the internal *wxRichTextHeaderFooterData* (p. 1359) object.

**wxRichTextPrinting::GetHeaderText**

**wxString GetHeaderText(wxRichTextOddEvenPage page =  
wxRICHTEXT\_PAGE\_EVEN, wxRichTextPageLocation location =  
wxRICHTEXT\_PAGE\_CENTRE) const**

A convenience function to get the header text. See *wxRichTextHeaderFooterData* (p. 1359) for details.

**wxRichTextPrinting::GetPageSetupData**

**wxPageSetupDialogData\* GetPageSetupData()**

Returns a pointer to the internal page setup data.

**wxRichTextPrinting::GetParentWindow**

**wxWindow\* GetParentWindow() const**

Returns the parent window to be used for the preview window and printing wait dialog.

**wxRichTextPrinting::GetPreviewRect**

**const wxRect& GetPreviewRect() const**

Returns the dimensions to be used for the preview window.

**wxRichTextPrinting::GetPrintData**

**wxPrintData\* GetPrintData()**

Returns a pointer to the internal print data.

**wxRichTextPrinting::GetTitle**

**const wxString& GetTitle() const**

Returns the title of the preview window or printing wait caption.

**wxRichTextPrinting::PageSetup**

**void PageSetup()**

Shows the page setup dialog.

**wxRichTextPrinting::PreviewBuffer**

**bool PreviewBuffer(const wxRichTextBuffer& *buffer*)**

Shows a preview window for the given buffer. The function takes its own copy of *buffer*.

**wxRichTextPrinting::PreviewFile**

**bool PreviewFile(const wxString& *richTextFile*)**

Shows a preview window for the given file. *richTextFile* can be a text file or XML file, or other file depending on the available file handlers.

**wxRichTextPrinting::PrintBuffer**

**bool PrintBuffer(const wxRichTextBuffer& *buffer*)**

Prints the given buffer. The function takes its own copy of *buffer*.

**wxRichTextPrinting::PrintFile**

**bool PrintFile(const wxString& *richTextFile*)**

Prints the given file. *richTextFile* can be a text file or XML file, or other file depending on the available file handlers.

**wxRichTextPrinting::SetFooterText**

**void SetFooterText(const wxString& *text*, wxRichTextOddEvenPage *page* = wxRICHTEXT\_PAGE\_ALL, wxRichTextPageLocation *location* = wxRICHTEXT\_PAGE\_CENTRE)**

A convenience function to set the footer text. See *wxRichTextHeaderFooterData* (p. 1359) for details.

**wxRichTextPrinting::SetHeaderFooterData**

**void SetHeaderFooterData(const wxRichTextHeaderFooterData& *data*)**

Sets the internal *wxRichTextHeaderFooterData* (p. 1359) object.

**wxRichTextPrinting::SetHeaderFooterFont**

**void SetHeaderFooterFont(const wxFont& *font*)**

Sets the *wxRichTextHeaderFooterData* (p. 1359) font.

**wxRichTextPrinting::SetHeaderFooterTextColour**

**void SetHeaderFooterTextColour(const wxColour& *colour*)**

Sets the *wxRichTextHeaderFooterData* (p. 1359) text colour.

**wxRichTextPrinting::SetHeaderText**

```
void SetHeaderText(const wxString& text, wxRichTextOddEvenPage page =  
wxRICHTEXT_PAGE_ALL, wxRichTextPageLocation location =  
wxRICHTEXT_PAGE_CENTRE)
```

A convenience function to set the header text. See *wxRichTextHeaderFooterData* (p. 1359) for details.

**wxRichTextPrinting::SetPageSetupData**

```
void SetPageSetupData(const wxPageSetupData& pageSetupData)
```

Sets the page setup data.

**wxRichTextPrinting::SetParentWindow**

```
void SetParentWindow(wxWindow* parent)
```

Sets the parent window to be used for the preview window and printing wait dialog.

**wxRichTextPrinting::SetPreviewRect**

```
void SetPreviewRect(const wxRect& rect)
```

Sets the dimensions to be used for the preview window.

**wxRichTextPrinting::SetPrintData**

```
void SetPrintData(const wxPrintData& printData)
```

Sets the print data.

**wxRichTextPrinting::SetShowOnFirstPage**

```
void SetShowOnFirstPage(bool show)
```

Pass `true` to show the header and footer on the first page.

**wxRichTextPrinting::SetTitle**

```
void SetTitle(const wxString& title)
```

Pass the title of the preview window or printing wait caption.

**wxRichTextPrintout**

This class implements print layout for *wxRichTextBuffer* (p. 1298). Instead of using it directly, you should normally use the *wxRichTextPrinting* (p. 1368) class.

**Derived from**

*wxPrintout* (p. 1214)

**Include files**

<wx/richtext/richtextprint.h>

**Data structures****wxRichTextPrintout::wxRichTextPrintout**

**wxRichTextPrintout(const wxString& title = wxT("Printout"))**

Constructor.

**wxRichTextPrintout::CalculateScaling**

**void CalculateScaling(wxDC\* dc, wxRect& textRect, wxRect& headerRect, wxRect& footerRect)**

Calculates scaling and text, header and footer rectangles.

**wxRichTextPrintout::GetHeaderFooterData**

**const wxRichTextHeaderFooterData& GetHeaderFooterData() const**

Returns the header and footer data associated with the printout.

**wxRichTextPrintout::GetPageInfo**

**void GetPageInfo(int\* minPage, int\* maxPage, int\* selPageFrom, int\* selPageTo)**

Gets the page information.

**wxRichTextPrintout::GetRichTextBuffer**

**wxRichTextBuffer\* GetRichTextBuffer() const**

Returns a pointer to the buffer being rendered.

**wxRichTextPrintout::HasPage**

**bool HasPage(int page)**

Returns `true` if the given page exists in the printout.

**wxRichTextPrintout::OnPreparePrinting**

**void OnPreparePrinting()**

Prepares for printing, laying out the buffer and calculating pagination.

**wxRichTextPrintout::OnPrintPage****bool OnPrintPage(int *page*)**

Does the actual printing for this page.

**wxRichTextPrintout::SetHeaderFooterData****void SetHeaderFooterData(const wxRichTextHeaderFooterData& *data*)**

Sets the header and footer data associated with the printout.

**wxRichTextPrintout::SetMargins****void SetMargins(int *top* = 252, int *bottom* = 252, int *left* = 252, int *right* = 252)**

Sets margins in 10ths of millimetre. Defaults to 1 inch for margins.

**wxRichTextPrintout::SetRichTextBuffer****void SetRichTextBuffer(wxRichTextBuffer\* *buffer*)**

Sets the buffer to print. wxRichTextPrintout does not manage this pointer; it should be managed by the calling code, such as wxRichTextPrinting.

**wxRichTextRange**

This class stores beginning and end positions for a range of data.

**Derived from**

No base class

**Include files**

<wx/richtext/richtextbuffer.h>

**Data structures****wxRichTextRange::wxRichTextRange****wxRichTextRange(long *start*, long *end*)****wxRichTextRange(const wxRichTextRange& *range*)**

**wxRichTextRange()**

Constructors.

**wxRichTextRange::~~wxRichTextRange****~wxRichTextRange()**

Destructor.

**wxRichTextRange::Contains****bool Contains(long pos) const**

Returns `true` if the given position is within this range. Does not match if the range is empty.

**wxRichTextRange::GetEnd****long GetEnd() const**

Returns the end position.

**wxRichTextRange::GetLength****long GetLength() const**

Returns the length of the range.

**wxRichTextRange::GetStart****long GetStart() const**

Returns the start of the range.

**wxRichTextRange::FromInternal****wxRichTextRange FromInternal() const**

Converts the internal range, which uses the first and last character positions of the range, to the API-standard range, whose end is one past the last character in the range. In other words, one is added to the end position.

**wxRichTextRange::IsOutside****bool IsOutside(const wxRichTextRange& range) const**

Returns `true` if this range is completely outside *range*.

**wxRichTextRange::IsWithin**

**bool IsWithin(const wxRichTextRange& range) const**

Returns `true` if this range is completely within *range*.

**wxRichTextRange::LimitTo**

**bool LimitTo(const wxRichTextRange& range)**

Limits this range to be within *range*.

**wxRichTextRange::SetEnd**

**void SetEnd(long end)**

Sets the end of the range.

**wxRichTextRange::SetRange**

**void SetRange(long start, long end)**

Sets the range.

**wxRichTextRange::SetStart**

**void SetStart(long start)**

Sets the start of the range.

**wxRichTextRange::Swap**

**void Swap()**

Swaps the start and end.

**wxRichTextRange::ToInternal**

**wxRichTextRange ToInternal() const**

Converts the API-standard range, whose end is one past the last character in the range, to the internal form, which uses the first and last character positions of the range. In other words, one is subtracted from the end position.

**wxRichTextRange::operator+**

**wxRichTextRange operator+(const wxRichTextRange& range) const**

Adds *range* to this range.

**wxRichTextRange::operator-**

**wxRichTextRange operator-(const wxRichTextRange& range) const**

Subtracts *range* from this range.

**wxRichTextRange::operator=**

**void operator=(const wxRichTextRange& range)**

Assigns *range* to this range.

**wxRichTextRange::operator==**

**bool operator==(const wxRichTextRange& range) const**

Returns `true` if *range* is the same as this range.

## wxRichTextStyleDefinition

This is a base class for paragraph and character styles.

### Derived from

*wxObject* (p. 1148)

### Include files

<wx/richtext/richtextstyles.h>

### Data structures

**wxRichTextStyleDefinition::wxRichTextStyleDefinition**

**wxRichTextStyleDefinition(const wxString& name = wxEmptyString)**

Constructor.

**wxRichTextStyleDefinition::~~wxRichTextStyleDefinition**

**~wxRichTextStyleDefinition()**

Destructor.

**wxRichTextStyleDefinition::GetBaseStyle**

**const wxString& GetBaseStyle() const**

Returns the style on which this style is based.



**wxRichTextStyleDefinition::GetDescription****const wxString& GetDescription() const**

Returns the style's description.

**wxRichTextStyleDefinition::GetName****const wxString& GetName() const**

Returns the style name.

**wxRichTextStyleDefinition::GetStyle****wxRichTextAttr& GetStyle()****const wxRichTextAttr& GetStyle() const**

Returns the attributes associated with this style.

**wxRichTextStyleDefinition::GetStyleMergedWithBase****wxRichTextAttr GetStyleMergedWithBase(wxRichTextStyleSheet\* *sheet*) const**

Returns the style attributes combined with the attributes of the specified base style, if any. This function works recursively.

**wxRichTextStyleDefinition::SetBaseStyle****void SetBaseStyle(const wxString& *name*)**

Sets the name of the style that this style is based on.

**wxRichTextStyleDefinition::SetDescription****void SetDescription(const wxString& *descr*)**

Sets the style description.

**wxRichTextStyleDefinition::SetName****void SetName(const wxString& *name*)**

Sets the name of the style.

**wxRichTextStyleDefinition::SetStyle****void SetStyle(const wxRichTextAttr& *style*)**

Sets the attributes for this style.

## **wxRichTextStyleComboCtrl**

This is a combo control that can display the styles in a *wxRichTextStyleSheet* (p. 1387), and apply the selection to an associated *wxRichTextCtrl* (p. 1316).

See `samples/richtext` for an example of how to use it.

### **Derived from**

*wxComboCtrl* (p. 232)

### **Include files**

`<wx/richtext/richtextstyles.h>`

### **See also**

*wxRichTextStyleListBox* (p. 1379), *wxRichTextCtrl* overview (p. 2188)

## **wxRichTextStyleComboCtrl::wxRichTextStyleComboCtrl**

**wxRichTextStyleComboCtrl**(wxWindow\* *parent*, wxWindowID *id* = wxID\_ANY, const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size* = wxDefaultSize, long *style* = 0)

Constructor.

## **wxRichTextStyleComboCtrl::~~wxRichTextStyleComboCtrl**

**~wxRichTextStyleComboCtrl**()

Destructor.

## **wxRichTextStyleComboCtrl::GetRichTextCtrl**

**wxRichTextCtrl\*** GetRichTextCtrl() const

Returns the *wxRichTextCtrl* (p. 1316) associated with this control.

## **wxRichTextStyleComboCtrl::GetStyleSheet**

**wxRichTextStyleSheet\*** GetStyleSheet() const

Returns the style sheet associated with this control.

## **wxRichTextStyleComboCtrl::SetRichTextCtrl**

**void** SetRichTextCtrl(wxRichTextCtrl\* *ctrl*)

Associates the control with a *wxRichTextCtrl* (p. 1316).

**wxRichTextStyleComboCtrl::SetStyleSheet****void SetStyleSheet(wxRichTextStyleSheet\* styleSheet)**

Associates the control with a style sheet.

**wxRichTextStyleComboCtrl::UpdateStyles****void UpdateStyles()**

Updates the combo control from the associated style sheet.

**wxRichTextStyleListBox**

This is a listbox that can display the styles in a *wxRichTextStyleSheet* (p. 1387), and apply the selection to an associated *wxRichTextCtrl* (p. 1316).

See `samples/richtext` for an example of how to use it.

**Derived from**

*wxHtmlListBox* (p. 853)

**Include files**

<wx/richtext/richtextstyles.h>

**See also**

*wxRichTextStyleComboCtrl* (p. 1378), *wxRichTextCtrl* overview (p. 2188)

**wxRichTextStyleListBox::wxRichTextStyleListBox****wxRichTextStyleListBox(wxWindow\* parent, wxWindowID id = wxID\_ANY, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0)**

Constructor.

**wxRichTextStyleListBox::~~wxRichTextStyleListBox****~wxRichTextStyleListBox()**

Destructor.

**wxRichTextStyleListBox::ApplyStyle****void ApplyStyle(int i)**

Applies the *i*th style to the associated rich text control.

**wxRichTextStyleListBox::ConvertTenthsMMToPixels****int ConvertTenthsMMToPixels(wxDC& dc, int units) const**

Converts units in tenths of a millimetre to device units.

**wxRichTextStyleListBox::CreateHTML****wxString CreateHTML(wxRichTextStyleDefinition\* def) const**

Creates a suitable HTML fragment for a definition.

**wxRichTextStyleListBox::GetApplyOnSelection****bool GetApplyOnSelection() const**

If the return value is `true`, clicking on a style name in the list will immediately apply the style to the associated rich text control.

**wxRichTextStyleListBox::GetRichTextCtrl****wxRichTextCtrl\* GetRichTextCtrl() const**

Returns the *wxRichTextCtrl* (p. 1316) associated with this listbox.

**wxRichTextStyleListBox::GetStyle****wxRichTextStyleDefinition\* GetStyle(size\_t i) const**

Gets a style for a listbox index.

**wxRichTextStyleListBox::GetStyleSheet****wxRichTextStyleSheet\* GetStyleSheet() const**

Returns the style sheet associated with this listbox.

**wxRichTextStyleListBox::GetStyleType****wxRichTextStyleListBox::wxRichTextStyleType GetStyleType() const**

Returns the type of style to show in the list box.

**wxRichTextStyleListBox::OnGetItem****wxString OnGetItem(size\_t n) const**

Returns the HTML for this item.

**wxRichTextStyleListBox::OnLeftDown**

**void OnLeftDown(wxMouseEvent& event)**

Implements left click behaviour, applying the clicked style to the wxRichTextCtrl.

**wxRichTextStyleListBox::OnSelect**

**void OnSelect(wxCommandEvent& event)**

Reacts to selection.

**wxRichTextStyleListBox::SetApplyOnSelection**

**void SetApplyOnSelection(bool applyOnSelection)**

If *applyOnSelection* is `true`, clicking on a style name in the list will immediately apply the style to the associated rich text control.

**wxRichTextStyleListBox::SetRichTextCtrl**

**void SetRichTextCtrl(wxRichTextCtrl\* ctrl)**

Associates the listbox with a *wxRichTextCtrl* (p. 1316).

**wxRichTextStyleListBox::SetStyleSheet**

**void SetStyleSheet(wxRichTextStyleSheet\* styleSheet)**

Associates the control with a style sheet.

**wxRichTextStyleListBox::SetStyleType**

**void SetStyleType(wxRichTextStyleListBox::wxRichTextStyleType styleType)**

Sets the style type to display. One of  
wxRichTextStyleListBox::wxRICHTEXT\_STYLE\_ALL,  
wxRichTextStyleListBox::wxRICHTEXT\_STYLE\_PARAGRAPH,  
wxRichTextStyleListBox::wxRICHTEXT\_STYLE\_CHARACTER and  
wxRichTextStyleListBox::wxRICHTEXT\_STYLE\_LIST.

**wxRichTextStyleListBox::UpdateStyles**

**void UpdateStyles()**

Updates the list from the associated style sheet.

**wxRichTextStyleListCtrl**

This class incorporates a *wxRichTextStyleListBox* (p. 1379) and a choice control that allows the user to select the category of style to view. It is demonstrated in the

`wxRichTextCtrl` sample in `samples/richtext`.

To use `wxRichTextStyleListCtrl`, add the control to your window hierarchy and call `SetStyleType` (p. 1383) with one of `wxRichTextStyleListBox::wxRICHTEXT_STYLE_ALL`, `wxRichTextStyleListBox::wxRICHTEXT_STYLE_PARAGRAPH`, `wxRichTextStyleListBox::wxRICHTEXT_STYLE_CHARACTER` and `wxRichTextStyleListBox::wxRICHTEXT_STYLE_LIST` to set the current view. Associate the control with a style sheet and rich text control with `SetStyleSheet` and `SetRichTextCtrl`, so that when a style is double-clicked, it is applied to the selection.

### Window styles

**wxRICHTEXTSTYLELIST\_HIDE\_TYPE\_SELECTOR**      This style hides the category selection control.

### Derived from

`wxControl` (p. 285)

### Include files

<wx/richtext/richtextstyles.h>

### Data structures

## **wxRichTextStyleListCtrl::wxRichTextStyleListCtrl**

**wxRichTextStyleListCtrl(wxWindow\* parent, wxWindowID id = wxID\_ANY, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0)**

**wxRichTextStyleListCtrl()**

Constructors.

## **wxRichTextStyleListCtrl::Create**

**bool Create(wxWindow\* parent, wxWindowID id = wxID\_ANY, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0)**

Creates the windows.

## **wxRichTextStyleListCtrl::GetRichTextCtrl**

**wxRichTextCtrl\* GetRichTextCtrl() const**

Returns the associated rich text control, if any.

## **wxRichTextStyleListCtrl::GetStyleChoice**

**wxChoice\* GetStyleChoice() const**

Returns the wxChoice control used for selecting the style category.

### **wxRichTextStyleListCtrl::GetStyleListBox**

**wxRichTextStyleListBox\* GetStyleListBox() const**

Returns the wxListBox control used to view the style list.

### **wxRichTextStyleListCtrl::GetStyleSheet**

**wxRichTextStyleSheet\* GetStyleSheet() const**

Returns the associated style sheet, if any.

### **wxRichTextStyleListCtrl::GetStyleType**

**wxRichTextStyleListBox::wxRichTextStyleType GetStyleType() const**

Returns the type of style to show in the list box.

### **wxRichTextStyleListCtrl::SetRichTextCtrl**

**void SetRichTextCtrl(wxRichTextCtrl\* ctrl)**

Associates the control with a wxRichTextCtrl.

### **wxRichTextStyleListCtrl::SetStyleSheet**

**void SetStyleSheet(wxRichTextStyleSheet\* styleSheet)**

Associates the control with a style sheet.

### **wxRichTextStyleListCtrl::SetStyleType**

**void SetStyleType(wxRichTextStyleListBox::wxRichTextStyleType styleType)**

Sets the style type to display. One of  
wxRichTextStyleListBox::wxRICHTEXT\_STYLE\_ALL,  
wxRichTextStyleListBox::wxRICHTEXT\_STYLE\_PARAGRAPH,  
wxRichTextStyleListBox::wxRICHTEXT\_STYLE\_CHARACTER and  
wxRichTextStyleListBox::wxRICHTEXT\_STYLE\_LIST.

### **wxRichTextStyleListCtrl::UpdateStyles**

**void UpdateStyles()**

Updates the style list box.

## **wxRichTextStyleOrganiserDialog**

This class shows a style sheet and allows the user to edit, add and remove styles. It can also be used as a style browser, for example if the application is not using a permanent *wxRichTextStyleComboCtrl* (p. 1378) or *wxRichTextStyleListCtrl* (p. 1381) to present styles. **Derived from**

*wxDialog* (p. 496)

**Include files**

<wx/richtext/richtextstyledlg.h>

**wxRichTextStyleOrganiserDialog::wxRichTextStyleOrganiserDialog**

**wxRichTextStyleOrganiserDialog**(int *flags*, **wxRichTextStyleSheet\*** *sheet*, **wxRichTextCtrl\*** *ctrl*, **wxWindow\*** *parent*, **wxWindowID** *id* = *wxID\_ANY*, **const wxString&** *caption* = \_("Style Organiser"), **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = *wxDEFAULT\_DIALOG\_STYLE|wxRESIZE\_BORDER|wxSYSTEM\_MENU|wxCLOSE\_BOX*)

**wxRichTextStyleOrganiserDialog()**

Constructors.

To create a dialog, pass a bitlist of *flags* (see below), a style sheet, a text control to apply a selected style to (or NULL), followed by the usual window parameters.

To specify the operations available to the user, pass a combination of these values to *flags*:

**wxRICHTEXT\_ORGANISER\_DELETE\_STYLES** Provides a button for deleting styles.

**wxRICHTEXT\_ORGANISER\_CREATE\_STYLES** Provides buttons for creating styles.

**wxRICHTEXT\_ORGANISER\_APPLY\_STYLES** Provides a button for applying the currently selected style to the selection.

**wxRICHTEXT\_ORGANISER\_EDIT\_STYLES** Provides a button for editing styles.

**wxRICHTEXT\_ORGANISER\_RENAME\_STYLES** Provides a button for renaming styles.

**wxRICHTEXT\_ORGANISER\_OK\_CANCEL** Provides OK and Cancel buttons.

**wxRICHTEXT\_ORGANISER\_RENUMBER** Provides a checkbox for specifying that the selection should be renumbered.

The following flags determine what will be displayed in the style list:

**wxRICHTEXT\_ORGANISER\_SHOW\_CHARACTER** Displays character styles only.



**wxRICHTEXT\_ORGANISER\_SHOW\_PARAGRAPH**      Displays paragraph styles only.

**wxRICHTEXT\_ORGANISER\_SHOW\_LIST** Displays list styles only.

**wxRICHTEXT\_ORGANISER\_SHOW\_ALL** Displays all styles.

The following symbols define commonly-used combinations of flags:

**wxRICHTEXT\_ORGANISER\_ORGANISE** Enable all style editing operations so the dialog behaves as a style organiser.

**wxRICHTEXT\_ORGANISER\_BROWSE** Show a list of all styles and their previews, but only allow application of a style or cancellation of the dialog. This makes the dialog behave as a style browser.

**wxRICHTEXT\_ORGANISER\_BROWSE\_NUMBERING** Enables only list style browsing, plus a control to specify renumbering. This allows the dialog to be used for applying list styles to the selection.

### **wxRichTextStyleOrganiserDialog::ApplyStyle**

**bool ApplyStyle(wxRichTextCtrl\* ctrl = NULL)**

Applies the selected style to selection in the given control or the control passed to the constructor.

### **wxRichTextStyleOrganiserDialog::Create**

**bool Create(int flags, wxRichTextStyleSheet\* sheet, wxRichTextCtrl\* ctrl, wxWindow\* parent, wxWindowID id = wxID\_ANY, const wxString& caption = \_("Style Organiser"), const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxDEFAULT\_DIALOG\_STYLE|wxRESIZE\_BORDER|wxSYSTEM\_MENU|wxCLOSE\_BOX)**

Creates the dialog. See

**wxRichTextStyleOrganiserDialog::wxRichTextStyleOrganiserDialog**

wxrichtextstyleorganiserdialogwxrichtextstyleorganiserdialog for details.

### **wxRichTextStyleOrganiserDialog::GetFlags**

**int GetFlags() const**

Returns the flags used to control the interface presented to the user.

### **wxRichTextStyleOrganiserDialog::GetRestartNumbering**

**bool GetRestartNumbering() const**

Returns `true` if the user has opted to restart numbering.

**wxRichTextStyleOrganiserDialog::GetRichTextCtrl**

**wxRichTextCtrl\* GetRichTextCtrl() const**

Returns the associated rich text control (if any).

**wxRichTextStyleOrganiserDialog::GetSelectedStyle**

**wxString GetSelectedStyle() const**

Returns selected style name.

**wxRichTextStyleOrganiserDialog::GetSelectedStyleDefinition**

**wxRichTextStyleDefinition\* GetSelectedStyleDefinition() const**

Returns selected style definition.

**wxRichTextStyleOrganiserDialog::GetStyleSheet**

**wxRichTextStyleSheet\* GetStyleSheet() const**

Returns the associated style sheet.

**wxRichTextStyleOrganiserDialog::SetFlags**

**void SetFlags(int flags)**

Sets the flags used to control the interface presented to the user.

**wxRichTextStyleOrganiserDialog::SetRestartNumbering**

**void SetRestartNumbering(bool restartNumbering)**

Checks or unchecks the restart numbering checkbox.

**wxRichTextStyleOrganiserDialog::SetRichTextCtrl**

**void SetRichTextCtrl(wxRichTextCtrl\* ctrl)**

Sets the control to be associated with the dialog, for the purposes of applying a style to the selection.

**wxRichTextStyleOrganiserDialog::SetShowToolTips**

**void SetShowToolTips(bool show)**

Determines whether tooltips will be shown.

### **wxRichTextStyleOrganiserDialog::SetStyleSheet**

**void SetStyleSheet(wxRichTextStyleSheet\* sheet)**

Sets the associated style sheet.

## **wxRichTextStyleSheet**

A style sheet contains named paragraph and character styles that make it easy for a user to apply combinations of attributes to a *wxRichTextCtrl* (p. 1316).

You can use a *wxRichTextStyleListBox* (p. 1379) in your user interface to show available styles to the user, and allow application of styles to the control.

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/richtext/richtextstyles.h>

### **Data structures**

### **wxRichTextStyleSheet::wxRichTextStyleSheet**

**wxRichTextStyleSheet()**

Constructor.

### **wxRichTextStyleSheet::~~wxRichTextStyleSheet**

**~wxRichTextStyleSheet()**

Destructor.

### **wxRichTextStyleSheet::AddCharacterStyle**

**bool AddCharacterStyle(wxRichTextCharacterStyleDefinition\* def)**

Adds a definition to the character style list.

### **wxRichTextStyleSheet::AddListStyle**

**bool AddListStyle(wxRichTextListStyleDefinition\* def)**

Adds a definition to the list style list.

**wxRichTextStyleSheet::AddParagraphStyle****bool AddParagraphStyle(wxRichTextParagraphStyleDefinition\* def)**

Adds a definition to the paragraph style list.

**wxRichTextStyleSheet::AddStyle****bool AddStyle(wxRichTextStyleDefinition\* def)**

Adds a definition to the appropriate style list.

**wxRichTextStyleSheet::DeleteStyles****void DeleteStyles()**

Deletes all styles.

**wxRichTextStyleSheet::FindCharacterStyle****wxRichTextCharacterStyleDefinition\* FindCharacterStyle(const wxString& name)  
const**

Finds a character definition by name.

**wxRichTextStyleSheet::FindListStyle****wxRichTextListStyleDefinition\* FindListStyle(const wxString& name) const**

Finds a list definition by name.

**wxRichTextStyleSheet::FindParagraphStyle****wxRichTextParagraphStyleDefinition\* FindParagraphStyle(const wxString& name)  
const**

Finds a paragraph definition by name.

**wxRichTextStyleSheet::FindStyle****wxRichTextStyleDefinition\* FindStyle(const wxString& name) const**

Finds a style definition by name.

**wxRichTextStyleSheet::GetCharacterStyle****wxRichTextCharacterStyleDefinition\* GetCharacterStyle(size\_t n) const**

Returns the *n*th character style.

**wxRichTextStyleSheet::GetCharacterStyleCount****size\_t GetCharacterStyleCount() const**

Returns the number of character styles.

**wxRichTextStyleSheet::GetDescription****const wxString& GetDescription() const**

Returns the style sheet's description.

**wxRichTextStyleSheet::GetListStyle****wxRichTextListStyleDefinition\* GetListStyle(size\_t n) const**

Returns the *n*th list style.

**wxRichTextStyleSheet::GetListStyleCount****size\_t GetListStyleCount() const**

Returns the number of list styles.

**wxRichTextStyleSheet::GetName****const wxString& GetName() const**

Returns the style sheet's name.

**wxRichTextStyleSheet::GetParagraphStyle****wxRichTextParagraphStyleDefinition\* GetParagraphStyle(size\_t n) const**

Returns the *n*th paragraph style.

**wxRichTextStyleSheet::GetParagraphStyleCount****size\_t GetParagraphStyleCount() const**

Returns the number of paragraph styles.

**wxRichTextStyleSheet::RemoveCharacterStyle****bool RemoveCharacterStyle(wxRichTextStyleDefinition\* def, bool deleteStyle = false)**

Removes a character style.

**wxRichTextStyleSheet::RemoveListStyle**

**bool RemoveListStyle(wxRichTextStyleDefinition\* def, bool deleteStyle = false)**

Removes a list style.

### **wxRichTextStyleSheet::RemoveParagraphStyle**

**bool RemoveParagraphStyle(wxRichTextStyleDefinition\* def, bool deleteStyle = false)**

Removes a paragraph style.

### **wxRichTextStyleSheet::RemoveStyle**

**bool RemoveStyle(wxRichTextStyleDefinition\* def, bool deleteStyle = false)**

Removes a style.

### **wxRichTextStyleSheet::SetDescription**

**void SetDescription(const wxString& descr)**

Sets the style sheet's description.

### **wxRichTextStyleSheet::SetName**

**void SetName(const wxString& name)**

Sets the style sheet's name.

## **wxRichTextXMLHandler**

A handler for loading and saving content in an XML format specific to wxRichTextBuffer. You can either add the handler to the buffer and load and save through the buffer or control API, or you can create an instance of the handler on the stack and call its functions directly.

### **Handler flags**

The following flags can be used with this handler, via the handler's SetFlags function or the buffer or control's SetHandlerFlags function:

**wxRICHTEXT\_HANDLER\_INCLUDE\_STYLESHEET**      Include the style sheet in loading and saving operations.

### **Derived from**

*wxRichTextFileHandler* (p. 1350)

### **Include files**

<wx/richtext/richtextxml.h>

## Data structures

### **wxRichTextXMLHandler::wxRichTextXMLHandler**

**wxRichTextXMLHandler(const wxString& name = wxT("XML"), const wxString& ext = wxT("xml"), int type = wxRICHTEXT\_TYPE\_XML)**

Constructor.

### **wxRichTextXMLHandler::CanLoad**

**bool CanLoad() const**

Returns `true`.

### **wxRichTextXMLHandler::CanSave**

**bool CanSave() const**

Returns `true`.

### **wxRichTextXMLHandler::CreateStyle**

**wxString CreateStyle(const wxTextAttrEx& attr, bool isPara = false)**

Creates XML code for a given character or paragraph style.

### **wxRichTextXMLHandler::DoLoadFile**

**bool DoLoadFile(wxRichTextBuffer\* buffer, wxInputStream& stream)**

Loads buffer context from the given stream.

### **wxRichTextXMLHandler::DoSaveFile**

**bool DoSaveFile(wxRichTextBuffer\* buffer, wxOutputStream& stream)**

Saves buffer context to the given stream.

### **wxRichTextXMLHandler::ExportXML**

**bool ExportXML(wxOutputStream& stream, wxMBConv\* convMem, wxMBConv\* convFile, wxRichTextObject& obj, int level)**

Recursively exports an object to the stream.

### **wxRichTextXMLHandler::GetNodeContent**

**wxString GetNodeContent(wxXmlNode\* node)**

Helper function: gets node context.

**wxRichTextXMLHandler::GetParamNode**

**wxXmlNode\* GetParamNode(wxXmlNode\* node, const wxString& param)**

Helper function: gets a named parameter from the XML node.

**wxRichTextXMLHandler::GetParamValue**

**wxString GetParamValue(wxXmlNode\* node, const wxString& param)**

Helper function: gets a named parameter from the XML node.

**wxRichTextXMLHandler::GetStyle**

**bool GetStyle(wxTextAttrEx& attr, wxXmlNode\* node, bool isPara = false)**

Helper function: gets style parameters from the given XML node.

**wxRichTextXMLHandler::GetText**

**wxString GetText(wxXmlNode\* node, const wxString& param = wxEmptyString, bool translate = false)**

Helper function: gets text from the node.

**wxRichTextXMLHandler::HasParam**

**bool HasParam(wxXmlNode\* node, const wxString& param)**

Helper function: returns `true` if the node has the given parameter.

**wxRichTextXMLHandler::ImportXML**

**bool ImportXML(wxRichTextBuffer\* buffer, wxXmlNode\* node)**

Recursively imports an object.

## **wxSashEvent**

A sash event is sent when the sash of a *wxSashWindow* (p. 1397) has been dragged by the user.

**Derived from**

*wxCommandEvent* (p. 250)

*wxEvent* (p. 572)



*wxObject* (p. 1148)

### Include files

<wx/sashwin.h>

### Event table macros

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

**EVT\_SASH\_DRAGGED(id, func)** Process a `wxEVT_SASH_DRAGGED` event, when the user has finished dragging a sash.

**EVT\_SASH\_DRAGGED\_RANGE(id1, id2, func)** Process a `wxEVT_SASH_DRAGGED_RANGE` event, when the user has finished dragging a sash. The event handler is called when windows with ids in the given range have their sashes dragged.

### Data structures

```
enum wxSashDragStatus
{
    wxSASH_STATUS_OK,
    wxSASH_STATUS_OUT_OF_RANGE
};
```

### Remarks

When a sash belonging to a sash window is dragged by the user, and then released, this event is sent to the window, where it may be processed by an event table entry in a derived class, a plug-in event handler or an ancestor class.

Note that the `wxSashWindow` doesn't change the window's size itself. It relies on the application's event handler to do that. This is because the application may have to handle other consequences of the resize, or it may wish to veto it altogether. The event handler should look at the drag rectangle: see `wxSashEvent::GetDragRect` (p. 1394) to see what the new size of the window would be if the resize were to be applied. It should also call `wxSashEvent::GetDragStatus` (p. 1394) to see whether the drag was OK or out of the current allowed range.

### See also

`wxSashWindow` (p. 1397), *Event handling overview* (p. 2077)

### `wxSashEvent::wxSashEvent`

**wxSashEvent(int id = 0, wxSashEdgePosition edge = wxSASH\_NONE)**

Constructor.

### **wxSashEvent::GetEdge**

#### **wxSashEdgePosition GetEdge() const**

Returns the dragged edge. The return value is one of `wxSASH_TOP`, `wxSASH_RIGHT`, `wxSASH_BOTTOM`, `wxSASH_LEFT`.

### **wxSashEvent::GetDragRect**

#### **wxRect GetDragRect() const**

Returns the rectangle representing the new size the window would be if the resize was applied. It is up to the application to set the window size if required.

### **wxSashEvent::GetDragStatus**

#### **wxSashDragStatus GetDragStatus() const**

Returns the status of the sash: one of `wxSASH_STATUS_OK`, `wxSASH_STATUS_OUT_OF_RANGE`. If the drag caused the notional bounding box of the window to flip over, for example, the drag will be out of range.

## **wxSashLayoutWindow**

`wxSashLayoutWindow` responds to `OnCalculateLayout` events generated by `wxLayoutAlgorithm` (p. 961). It allows the application to use simple accessors to specify how the window should be laid out, rather than having to respond to events. The fact that the class derives from `wxSashWindow` allows sashes to be used if required, to allow the windows to be user-resizable.

The documentation for `wxLayoutAlgorithm` (p. 961) explains the purpose of this class in more detail.

### **Derived from**

`wxSashWindow` (p. 1397)

`wxWindow` (p. 1795)

`wxEvtHandler` (p. 576)

`wxObject` (p. 1148)

### **Include files**

<wx/laywin.h>

### **Window styles**

See `wxSashWindow` (p. 1397).

### **Event handling**

This class handles the `EVT_QUERY_LAYOUT_INFO` and `EVT_CALCULATE_LAYOUT` events for you. However, if you use sashes, see *wxSashWindow* (p. 1397) for relevant event information.

See also *wxLayoutAlgorithm* (p. 961) for information about the layout events.

### See also

*wxLayoutAlgorithm* (p. 961), *wxSashWindow* (p. 1397), *Event handling overview* (p. 2077)

## wxSashLayoutWindow::wxSashLayoutWindow

### wxSashLayoutWindow()

Default constructor.

**wxSashLayoutWindow**(**wxSashLayoutWindow\*** *parent*, **wxWindowID** *id*, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = *wxCLIP\_CHILDREN* | *wxSW\_3D*, **const wxString&** *name* = "layoutWindow")

Constructs a sash layout window, which can be a child of a frame, dialog or any other non-control window.

### Parameters

*parent*

Pointer to a parent window.

*id*

Window identifier. If -1, will automatically create an identifier.

*pos*

Window position. *wxDefaultPosition* is (-1, -1) which indicates that *wxSashLayoutWindows* should generate a default position for the window. If using the *wxSashLayoutWindow* class directly, supply an actual position.

*size*

Window size. *wxDefaultSize* is (-1, -1) which indicates that *wxSashLayoutWindows* should generate a default size for the window.

*style*

Window style. For window styles, please see *wxSashLayoutWindow* (p. 1394).

*name*

Window name.

**wxSashLayoutWindow::Create**

```
bool Create(wxSashLayoutWindow* parent, wxWindowID id, const wxPoint& pos =  
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxCLIP_CHILDREN | wxSW_3D, const wxString& name = "layoutWindow")
```

Initializes a sash layout window, which can be a child of a frame, dialog or any other non-control window.

**Parameters**

*parent*

Pointer to a parent window.

*id*

Window identifier. If -1, will automatically create an identifier.

*pos*

Window position. wxDefaultPosition is (-1, -1) which indicates that wxSashLayoutWindows should generate a default position for the window. If using the wxSashLayoutWindow class directly, supply an actual position.

*size*

Window size. wxDefaultSize is (-1, -1) which indicates that wxSashLayoutWindows should generate a default size for the window.

*style*

Window style. For window styles, please see *wxSashLayoutWindow* (p. 1394).

*name*

Window name.

**wxSashLayoutWindow::GetAlignment**

```
wxLayoutAlignment GetAlignment() const
```

Returns the alignment of the window: one of wxLAYOUT\_TOP, wxLAYOUT\_LEFT, wxLAYOUT\_RIGHT, wxLAYOUT\_BOTTOM.

**wxSashLayoutWindow::GetOrientation**

```
wxLayoutOrientation GetOrientation() const
```

Returns the orientation of the window: one of wxLAYOUT\_HORIZONTAL, wxLAYOUT\_VERTICAL.

**wxSashLayoutWindow::OnCalculateLayout**

**void OnCalculateLayout(wxCalculateLayoutEvent& event)**

The default handler for the event that is generated by wxLayoutAlgorithm. The implementation of this function calls wxCalculateLayoutEvent::SetRect to shrink the provided size according to how much space this window takes up. For further details, see *wxLayoutAlgorithm* (p. 961) and *wxCalculateLayoutEvent* (p. 167).

**wxSashLayoutWindow::OnQueryLayoutInfo**

**void OnQueryLayoutInfo(wxQueryLayoutInfoEvent& event)**

The default handler for the event that is generated by OnCalculateLayout to get size, alignment and orientation information for the window. The implementation of this function uses member variables as set by accessors called by the application. For further details, see *wxLayoutAlgorithm* (p. 961) and *wxQueryLayoutInfoEvent* (p. 1238).

**wxSashLayoutWindow::SetAlignment**

**void SetAlignment(wxLayoutAlignment alignment)**

Sets the alignment of the window (which edge of the available parent client area the window is attached to). *alignment* is one of wxLAYOUT\_TOP, wxLAYOUT\_LEFT, wxLAYOUT\_RIGHT, wxLAYOUT\_BOTTOM.

**wxSashLayoutWindow::SetDefaultSize**

**void SetDefaultSize(const wxSize& size)**

Sets the default dimensions of the window. The dimension other than the orientation will be fixed to this value, and the orientation dimension will be ignored and the window stretched to fit the available space.

**wxSashLayoutWindow::SetOrientation**

**void SetOrientation(wxLayoutOrientation orientation)**

Sets the orientation of the window (the direction the window will stretch in, to fill the available parent client area). *orientation* is one of wxLAYOUT\_HORIZONTAL, wxLAYOUT\_VERTICAL.

## wxSashWindow

wxSashWindow allows any of its edges to have a sash which can be dragged to resize the window. The actual content window will be created by the application as a child of wxSashWindow. The window (or an ancestor) will be notified of a drag via a *wxSashEvent* (p. 1392) notification.

**Derived from**

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### Include files

<wx/sashwin.h>

### Window styles

The following styles apply in addition to the normal *wxWindow* styles.

**wxSW\_3D**                      Draws a 3D effect sash and border.

**wxSW\_3DSASH**                Draws a 3D effect sash.

**wxSW\_3DBORDER**             Draws a 3D effect border.

**wxSW\_BORDER**                Draws a thin black border.

See also *window styles overview* (p. 2089).

### Event handling

**EVT\_SASH\_DRAGGED(id, func)**      Process a *wxEVT\_SASH\_DRAGGED* event, when the user has finished dragging a sash.

**EVT\_SASH\_DRAGGED\_RANGE(id1, id2, func)**      Process a *wxEVT\_SASH\_DRAGGED\_RANGE* event, when the user has finished dragging a sash. The event handler is called when windows with ids in the given range have their sashes dragged.

### Data types

```
enum wxSashEdgePosition {  
    wxSASH_TOP = 0,  
    wxSASH_RIGHT,  
    wxSASH_BOTTOM,  
    wxSASH_LEFT,  
    wxSASH_NONE = 100  
};
```

### See also

*wxSashEvent* (p. 1392), *wxSashLayoutWindow* (p. 1394), *Event handling overview* (p. 2077)

## **wxSashWindow::wxSashWindow**

**wxSashWindow()**

Default constructor.

```
wxSashWindow(wxWindow* parent, wxWindowID id, const wxPoint& pos =  
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxCLIP_CHILDREN | wxSW_3D, const wxString& name = "sashWindow")
```

Constructs a sash window, which can be a child of a frame, dialog or any other non-control window.

### Parameters

*parent*

Pointer to a parent window.

*id*

Window identifier. If -1, will automatically create an identifier.

*pos*

Window position. wxDefaultPosition is (-1, -1) which indicates that wxSashWindows should generate a default position for the window. If using the wxSashWindow class directly, supply an actual position.

*size*

Window size. wxDefaultSize is (-1, -1) which indicates that wxSashWindows should generate a default size for the window.

*style*

Window style. For window styles, please see *wxSashWindow* (p. 1397).

*name*

Window name.

### **wxSashWindow::~~wxSashWindow**

```
~wxSashWindow()
```

Destructor.

### **wxSashWindow::GetSashVisible**

```
bool GetSashVisible(wxSashEdgePosition edge) const
```

Returns true if a sash is visible on the given edge, false otherwise.

### Parameters

*edge*

Edge. One of wxSASH\_TOP, wxSASH\_RIGHT, wxSASH\_BOTTOM, wxSASH\_LEFT.

**See also**

*wxSashWindow::SetSashVisible* (p. 1401)

**wxSashWindow::GetMaximumSizeX**

**int GetMaximumSizeX() const**

Gets the maximum window size in the x direction.

**wxSashWindow::GetMaximumSizeY**

**int GetMaximumSizeY() const**

Gets the maximum window size in the y direction.

**wxSashWindow::GetMinimumSizeX**

**int GetMinimumSizeX()**

Gets the minimum window size in the x direction.

**wxSashWindow::GetMinimumSizeY**

**int GetMinimumSizeY() const**

Gets the minimum window size in the y direction.

**wxSashWindow::HasBorder**

**bool HasBorder(wxSashEdgePosition *edge*) const**

Returns true if the sash has a border, false otherwise. This function is obsolete since the sash border property is unused.

**Parameters**

*edge*

Edge. One of wxSASH\_TOP, wxSASH\_RIGHT, wxSASH\_BOTTOM, wxSASH\_LEFT.

**See also**

*wxSashWindow::SetSashBorder* (p. 1401)

**wxSashWindow::SetMaximumSizeX**



**void SetMaximumSizeX(int *min*)**

Sets the maximum window size in the x direction.

**wxSashWindow::SetMaximumSizeY**

**void SetMaximumSizeY(int *min*)**

Sets the maximum window size in the y direction.

**wxSashWindow::SetMinimumSizeX**

**void SetMinimumSizeX(int *min*)**

Sets the minimum window size in the x direction.

**wxSashWindow::SetMinimumSizeY**

**void SetMinimumSizeY(int *min*)**

Sets the minimum window size in the y direction.

**wxSashWindow::SetSashVisible**

**void SetSashVisible(wxSashEdgePosition *edge*, bool *visible*)**

Call this function to make a sash visible or invisible on a particular edge.

**Parameters**

*edge*

Edge to change. One of wxSASH\_TOP, wxSASH\_RIGHT, wxSASH\_BOTTOM, wxSASH\_LEFT.

*visible*

true to make the sash visible, false to make it invisible.

**See also**

*wxSashWindow::GetSashVisible* (p. 1399)

**wxSashWindow::SetSashBorder**

**void SetSashBorder(wxSashEdgePosition *edge*, bool *hasBorder*)**

Call this function to give the sash a border, or remove the border. This function is obsolete since the sash border property is unused.

**Parameters**

*edge*

Edge to change. One of wxSASH\_TOP, wxSASH\_RIGHT, wxSASH\_BOTTOM, wxSASH\_LEFT.

*hasBorder*

true to give the sash a border visible, false to remove it.

**See also**

*wxSashWindow::HasBorder* (p. 1400)

## wxScopedArray

This is a simple scoped smart pointer array implementation that is similar to the Boost (<http://www.boost.org>) smart pointers but rewritten to use macros instead.

### Example

Below is an example of using a wxWidgets scoped smart pointer and pointer array.

```
class MyClass { /* ... */ };

// declare a smart pointer to a MyClass called wxMyClassPtr
wxDECLARE_SCOPED_PTR(MyClass, wxMyClassPtr)
// declare a smart pointer to an array of chars
wxDECLARE_SCOPED_ARRAY(char, wxCharArray)

...

// define the first pointer class, must be complete
wxDEFINE_SCOPED_PTR(MyClass, wxMyClassPtr)
// define the second pointer class
wxDEFINE_SCOPED_ARRAY(char, wxCharArray)

// create an object with a new pointer to MyClass
wxMyClassPtr theObj(new MyClass());
// reset the pointer (deletes the previous one)
theObj.reset(new MyClass());

// access the pointer
theObj->MyFunc();

// create an object with a new array of chars
wxCharArray theCharObj(new char[100]);

// access the array
theCharObj[0] = "!";
```

### Declaring new smart pointer types

```
wxDECLAR_SCOPED_ARRAY( TYPE,          // type of the values
                      CLASSNAME ); // name of the class
```

A smart pointer holds a pointer to an object (which must be complete when `wxDEFINE_SCOPED_ARRAY()` is called). The memory used by the object is deleted when the smart pointer goes out of scope. The first argument of the macro is the pointer type, the second is the name of the new smart pointer class being created. Below we will use `wxScopedArray` to represent the scoped pointer array class, but the user may create the class with any legal name.

**Include files**

`<wx/ptr_scpd.h>`

**See also**

`wxScopedPtr` (p. 1404)

**wxScopedArray::wxScopedArray**

**wxScopedArray**(type \* *T* = *NULL*)

Creates the smart pointer with the given pointer or none if *NULL*. On compilers that support it, this uses the `explicit` keyword.

**wxScopedArray::reset**

**reset**(*T p* \* = *NULL*)

Deletes the currently held pointer and sets it to '*p*' or to *NULL* if no arguments are specified. This function does check to make sure that the pointer you are assigning is not the same pointer that is already stored.

**wxScopedArray::operator []**

**const T & operator []**(long *int*)

This operator acts like the standard `[]` indexing operator for C++ arrays. The function does not do bounds checking.

**wxScopedArray::get**

**const T\*** **get**()

This operator gets the pointer stored in the smart pointer or returns *NULL* if there is none.

**wxScopedArray::swap**

**swap**(**wxScopedPtr** & *ot*)

Swap the pointer inside the smart pointer with '*ot*'. The pointer being swapped must be of the same type (hence the same class name).

## wxScopedPtr

This is a simple scoped smart pointer implementation that is similar to the Boost (<http://www.boost.org/>) smart pointers but rewritten to use macros instead.

A smart pointer holds a pointer to an object. The memory used by the object is deleted when the smart pointer goes out of scope. This class is different from the `std::auto_ptr<>` in so far as it doesn't provide copy constructor nor assignment operator. This limits what you can do with it but is much less surprising than the "destructive copy" behaviour of the standard class.

### Example

Below is an example of using a wxWidgets scoped smart pointer and pointer array.

```
class MyClass { /* ... */ };

// declare a smart pointer to a MyClass called wxMyClassPtr
wxDECLARE_SCOPED_PTR(MyClass, wxMyClassPtr)
// declare a smart pointer to an array of chars
wxDECLARE_SCOPED_ARRAY(char, wxCharArray)

...

// define the first pointer class, must be complete
wxDEFINE_SCOPED_PTR(MyClass, wxMyClassPtr)
// define the second pointer class
wxDEFINE_SCOPED_ARRAY(char, wxCharArray)

// create an object with a new pointer to MyClass
wxMyClassPtr theObj(new MyClass());
// reset the pointer (deletes the previous one)
theObj.reset(new MyClass());

// access the pointer
theObj->MyFunc();

// create an object with a new array of chars
wxCharArray theCharObj(new char[100]);

// access the array
theCharObj[0] = "!";
```

### Declaring new smart pointer types

To declare the smart pointer class `CLASSNAME` containing pointers to a (possibly incomplete) type `TYPE` you should use

```
wxDECLARE_SCOPED_PTR( TYPE,          // type of the values
                     CLASSNAME ); // name of the class
```

And later, when `TYPE` is fully defined, you must also use

```
wxDEFINE_SCOPED_PTR( TYPE, CLASSNAME );
```

to implement the scoped pointer class.

The first argument of these macro is the pointer type, the second is the name of the new smart pointer class being created. Below we will use `wxScopedPtr` to represent the scoped pointer class, but the user may create the class with any legal name.

Alternatively, if you don't have to separate the point of declaration and definition of this class and if you accept the standard naming convention, that is that the scoped pointer for the class `Foo` is called `FooPtr`, you can use a single macro which replaces two macros above:

```
wxDEFINE_SCOPED_PTR_TYPE( TYPE );
```

Once again, in this case `CLASSNAME` will be `TYPEPtr`.

### Include files

<wx/ptr\_scpd.h>

### See also

*wxScopedArray* (p. 1402)

## **wxScopedPtr::wxScopedPtr**

**explicit wxScopedPtr**(type \* *T* = *NULL*)

Creates the smart pointer with the given pointer or none if `NULL`. On compilers that support it, this uses the `explicit` keyword.

## **wxScopedPtr::~~wxScopedPtr**

**~wxScopedPtr**()

Destructor frees the pointer held by this object if it is not `NULL`.

## **wxScopedPtr::release**

**T \* release**()

Returns the currently held pointer and resets the smart pointer object to `NULL`. After a call to this function the caller is responsible for deleting the pointer.

## **wxScopedPtr::reset**

**reset**(*T p* \* = *NULL*)

Deletes the currently held pointer and sets it to *p* or to `NULL` if no arguments are specified. This function does check to make sure that the pointer you are assigning is not the same pointer that is already stored.

**wxScopedPtr::operator \*****const T& operator \*()**

This operator works like the standard C++ pointer operator to return the object being pointed to by the pointer. If the pointer is NULL or invalid this will crash.

**wxScopedPtr::operator ->**

**const T\* operator ->()** This operator works like the standard C++ pointer operator to return the pointer in the smart pointer or NULL if it is empty.

**wxScopedPtr::get****const T\* get()**

This operator gets the pointer stored in the smart pointer or returns NULL if there is none.

**wxScopedPtr::swap****swap(wxScopedPtr & other)**

Swap the pointer inside the smart pointer with *other*. The pointer being swapped must be of the same type (hence the same class name).

**wxScopedTiedPtr**

This is a variation on the topic of *wxScopedPtr* (p. 1404). This class is also a smart pointer but in addition it "ties" the pointer value to another variable. In other words, during the life time of this class the value of that variable is set to be the same as the value of the pointer itself and it is reset to its old value when the object is destroyed. This class is especially useful when converting the existing code (which may already store the pointers value in some variable) to the smart pointers.

**Example****Derives from***wxScopedPtr* (p. 1404)**Include files**

&lt;wx/ptr\_scpd.h&gt;

**wxScopedTiedPtr::wxScopedTiedPtr****wxScopedTiedPtr(T \*\*ppTie, T \*ptr)**

Constructor creates a smart pointer initialized with *ptr* and stores *ptr* in the location

specified by *ppTie* which must not be `NULL`.

### **wxScopedTiedPtr::~~wxScopedTiedPtr**

#### **~wxScopedTiedPtr()**

Destructor frees the pointer held by this object and restores the value stored at the tied location (as specified in the *constructor* (p. 1406)) to the old value.

Warning: this location may now contain an uninitialized value if it hadn't been initialized previously, in particular don't count on it magically being `NULL`!

## **wxScreenDC**

A `wxScreenDC` can be used to paint on the screen. This should normally be constructed as a temporary stack object; don't store a `wxScreenDC` object.

### **Derived from**

*wxDC* (p. 456)

### **Include files**

<wx/dcscreen.h>

### **See also**

*wxDC* (p. 456), *wxMemoryDC* (p. 1069), *wxPaintDC* (p. 1164), *wxClientDC* (p. 193), *wxWindowDC* (p. 1855)

### **wxScreenDC::wxScreenDC**

#### **wxScreenDC()**

Constructor.

### **wxScreenDC::StartDrawingOnTop**

**bool StartDrawingOnTop(wxWindow\* window)**

**bool StartDrawingOnTop(wxRect\* rect = NULL)**

Use this in conjunction with *EndDrawingOnTop* (p. 1408) to ensure that drawing to the screen occurs on top of existing windows. Without this, some window systems (such as X) only allow drawing to take place underneath other windows.

By using the first form of this function, an application is specifying that the area that will be drawn on coincides with the given window.

By using the second form, an application can specify an area of the screen which is to be

drawn on. If NULL is passed, the whole screen is available.

It is recommended that an area of the screen is specified because with large regions, flickering effects are noticeable when destroying the temporary transparent window used to implement this feature.

You might use this pair of functions when implementing a drag feature, for example as in the *wxSplitterWindow* (p. 1508) implementation.

### Remarks

This function is probably obsolete since the X implementations allow drawing directly on the screen now. However, the fact that this function allows the screen to be refreshed afterwards, may be useful to some applications.

## **wxScreenDC::EndDrawingOnTop**

### **bool EndDrawingOnTop()**

Use this in conjunction with *StartDrawingOnTop* (p. 1407).

This function destroys the temporary window created to implement on-top drawing (X only).

## **wxScrollBar**

A *wxScrollBar* is a control that represents a horizontal or vertical scrollbar. It is distinct from the two scrollbars that some windows provide automatically, but the two types of scrollbar share the way events are received.

### Derived from

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/scrolbar.h>

### Remarks

A scrollbar has the following main attributes: *range*, *thumb size*, *page size*, and *position*.

The range is the total number of units associated with the view represented by the scrollbar. For a table with 15 columns, the range would be 15.

The thumb size is the number of units that are currently visible. For the table example, the window might be sized so that only 5 columns are currently visible, in which case the application would set the thumb size to 5. When the thumb size becomes the same as or greater than the range, the scrollbar will be automatically hidden on most platforms.



The page size is the number of units that the scrollbar should scroll by, when 'paging' through the data. This value is normally the same as the thumb size length, because it is natural to assume that the visible window size defines a page.

The scrollbar position is the current thumb position.

Most applications will find it convenient to provide a function called **AdjustScrollbars** which can be called initially, from an **OnSize** event handler, and whenever the application data changes in size. It will adjust the view, object and page size according to the size of the window and the size of the data.

### Window styles

**wxSB\_HORIZONTAL**            Specifies a horizontal scrollbar.

**wxSB\_VERTICAL**            Specifies a vertical scrollbar.

See also *window styles overview* (p. 2089).

### Event table macros

To process a scroll event, use these event handler macros to direct input to member functions that take a `wxScrollEvent` argument. You can use `EVT_COMMAND_SCROLL` . . . macros with window IDs for when intercepting scroll events from controls, or `EVT_SCROLL` . . . macros without window IDs for intercepting scroll events from the receiving window -- except for this, the macros behave exactly the same.

<b>EVT_SCROLL(func)</b>	Process all scroll events.
<b>EVT_SCROLL_TOP(func)</b>	Process <code>wxEVT_SCROLL_TOP</code> scroll-to-top events (minimum position).
<b>EVT_SCROLL_BOTTOM(func)</b>	Process <code>wxEVT_SCROLL_BOTTOM</code> scroll-to-bottom events (maximum position).
<b>EVT_SCROLL_LINEUP(func)</b>	Process <code>wxEVT_SCROLL_LINEUP</code> line up events.
<b>EVT_SCROLL_LINEDOWN(func)</b>	Process <code>wxEVT_SCROLL_LINEDOWN</code> line down events.
<b>EVT_SCROLL_PAGEUP(func)</b>	Process <code>wxEVT_SCROLL_PAGEUP</code> page up events.
<b>EVT_SCROLL_PAGEDOWN(func)</b>	Process <code>wxEVT_SCROLL_PAGEDOWN</code> page down events.
<b>EVT_SCROLL_THUMBTRACK(func)</b>	Process

	wxEVT_SCROLL_THUMBTRACK thumbtrack events (frequent events sent as the user drags the thumbtrack).
<b>EVT_SCROLL_THUMBRELEASE(func)</b>	Process wxEVT_SCROLL_THUMBRELEASE thumb release events.
<b>EVT_SCROLL_CHANGED(func)</b>	Process wxEVT_SCROLL_CHANGED end of scrolling events (MSW only).
<b>EVT_COMMAND_SCROLL(id, func)</b>	Process all scroll events.
<b>EVT_COMMAND_SCROLL_TOP(id, func)</b>	Process wxEVT_SCROLL_TOP scroll-to-top events (minimum position).
<b>EVT_COMMAND_SCROLL_BOTTOM(id, func)</b>	Process wxEVT_SCROLL_BOTTOM scroll-to-bottom events (maximum position).
<b>EVT_COMMAND_SCROLL_LINEUP(id, func)</b>	Process wxEVT_SCROLL_LINEUP line up events.
<b>EVT_COMMAND_SCROLL_LINEDOWN(id, func)</b>	Process wxEVT_SCROLL_LINEDOWN line down events.
<b>EVT_COMMAND_SCROLL_PAGEUP(id, func)</b>	Process wxEVT_SCROLL_PAGEUP page up events.
<b>EVT_COMMAND_SCROLL_PAGEDOWN(id, func)</b>	Process wxEVT_SCROLL_PAGEDOWN page down events.
<b>EVT_COMMAND_SCROLL_THUMBTRACK(id, func)</b>	Process wxEVT_SCROLL_THUMBTRACK thumbtrack events (frequent events sent as the user drags the thumbtrack).
<b>EVT_COMMAND_SCROLL_THUMBRELEASE(func)</b>	Process wxEVT_SCROLL_THUMBRELEASE thumb release events.
<b>EVT_COMMAND_SCROLL_CHANGED(func)</b>	Process wxEVT_SCROLL_CHANGED end of scrolling events (MSW only).

**The difference between EVT\_SCROLL\_THUMBRELEASE and EVT\_SCROLL\_CHANGED**

The EVT\_SCROLL\_THUMBRELEASE event is only emitted when actually dragging the thumb using the mouse and releasing it (This EVT\_SCROLL\_THUMBRELEASE event is also followed by an EVT\_SCROLL\_CHANGED event).

The EVT\_SCROLL\_CHANGED event also occurs when using the keyboard to change the thumb position, and when clicking next to the thumb (In all these cases the EVT\_SCROLL\_THUMBRELEASE event does not happen).

In short, the EVT\_SCROLL\_CHANGED event is triggered when scrolling/ moving has finished independently of the way it had started. Please see the widgets sample ("Slider" page) to see the difference between EVT\_SCROLL\_THUMBRELEASE and EVT\_SCROLL\_CHANGED in action.

**See also**

*Scrolling overview* (p. 2115), *Event handling overview* (p. 2077), *wxScrolledWindow* (p. 1414)

**wxScrollBar::wxScrollBar****wxScrollBar()**

Default constructor.

**wxScrollBar(wxWindow\* parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxSB\_HORIZONTAL, const wxValidator& validator = wxDefaultValidator, const wxString& name = "scrollBar")**

Constructor, creating and showing a scrollbar.

**Parameters**

*parent*

Parent window. Must be non-NULL.

*id*

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

*pos*

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

*size*

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

*style*

Window style. See *wxScrollBar* (p. 1408).

*validator*

Window validator.

*name*

Window name.

### See also

*wxScrollBar::Create* (p. 1412), *wxValidator* (p. 1767)

## **wxScrollBar::~wxScrollBar**

**void ~wxScrollBar()**

Destructor, destroying the scrollbar.

## **wxScrollBar::Create**

**bool Create**(*wxWindow\** parent, *wxWindowID* id, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = *wxSB\_HORIZONTAL*, **const wxValidator&** validator = *wxDefaultValidator*, **const wxString&** name = "scrollBar")

Scrollbar creation function called by the scrollbar constructor. See *wxScrollBar::wxScrollBar* (p. 1411) for details.

## **wxScrollBar::GetRange**

**int GetRange() const**

Returns the length of the scrollbar.

### See also

*wxScrollBar::SetScrollbar* (p. 1413)

## **wxScrollBar::GetPageSize**

**int GetPageSize() const**

Returns the page size of the scrollbar. This is the number of scroll units that will be scrolled when the user pages up or down. Often it is the same as the thumb size.

### See also

*wxScrollBar::SetScrollbar* (p. 1413)

## **wxScrollBar::GetThumbPosition**

**int GetThumbPosition() const**

Returns the current position of the scrollbar thumb.

**See also**

*wxScrollBar::SetThumbPosition* (p. 1413)

**wxScrollBar::GetThumbSize****int GetThumbSize() const**

Returns the thumb or 'view' size.

**See also**

*wxScrollBar::SetScrollbar* (p. 1413)

**wxScrollBar::SetThumbPosition****void SetThumbPosition(int viewStart)**

Sets the position of the scrollbar.

**Parameters**

*viewStart*

The position of the scrollbar thumb.

**See also**

*wxScrollBar::GetThumbPosition* (p. 1412)

**wxScrollBar::SetScrollbar****virtual void SetScrollbar(int position, int thumbSize, int range, int pageSize, const bool refresh = true)**

Sets the scrollbar properties.

**Parameters**

*position*

The position of the scrollbar in scroll units.

*thumbSize*

The size of the thumb, or visible portion of the scrollbar, in scroll units.

*range*

The maximum position of the scrollbar.

### *pageSize*

The size of the page size in scroll units. This is the number of units the scrollbar will scroll when it is paged up or down. Often it is the same as the thumb size.

### *refresh*

true to redraw the scrollbar, false otherwise.

### **Remarks**

Let's say you wish to display 50 lines of text, using the same font. The window is sized so that you can only see 16 lines at a time.

You would use:

```
scrollbar->SetScrollbar(0, 16, 50, 15);
```

The page size is 1 less than the thumb size so that the last line of the previous page will be visible on the next page, to help orient the user.

Note that with the window at this size, the thumb position can never go above 50 minus 16, or 34.

You can determine how many lines are currently visible by dividing the current view size by the character height in pixels.

When defining your own scrollbar behaviour, you will always need to recalculate the scrollbar settings when the window size changes. You could therefore put your scrollbar calculations and `SetScrollbar` call into a function named `AdjustScrollbars`, which can be called initially and also from a `wxSizeEvent` (p. 1443) event handler function.

### **See also**

*Scrolling overview* (p. 2115), `wxWindow::SetScrollbar` (p. 1843), `wxScrolledWindow` (p. 1414)

## **wxScrolledWindow**

The `wxScrolledWindow` class manages scrolling for its client area, transforming the coordinates according to the scrollbar positions, and setting the scroll positions, thumb sizes and ranges according to the area in view.

Starting from version 2.4 of `wxWidgets`, there are several ways to use a `wxScrolledWindow`. In particular, there are now three ways to set the size of the scrolling area:

One way is to set the scrollbars directly using a call to `wxScrolledWindow::SetScrollbars` (p. 1421). This is the way it used to be in any previous version of `wxWidgets` and it will be kept for backwards compatibility.

An additional method of manual control, which requires a little less computation of your own, is to set the total size of the scrolling area by calling either `wxWindow::SetVirtualSize` (p. 1848), or `wxWindow::FitInside` (p. 1808), and setting the scrolling increments for it by calling `wxScrolledWindow::SetScrollRate` (p. 1423). Scrolling in some orientation is enabled by setting a non-zero increment for it.

The most automatic and newest way is to simply let sizers determine the scrolling area. This is now the default when you set an interior sizer into a `wxScrolledWindow` with `wxWindow::SetSizer` (p. 1846). The scrolling area will be set to the size requested by the sizer and the scrollbars will be assigned for each orientation according to the need for them and the scrolling increment set by `wxScrolledWindow::SetScrollRate` (p. 1423). As above, scrolling is only enabled in orientations with a non-zero increment. You can influence the minimum size of the scrolled area controlled by a sizer by calling `wxWindow::SetVirtualSizeHints` (p. 1848). (calling `wxScrolledWindow::SetScrollbars` (p. 1421) has analogous effects in wxWidgets 2.4 -- in later versions it may not continue to override the sizer)

Note: if Maximum size hints are still supported by `SetVirtualSizeHints`, use them at your own dire risk. They may or may not have been removed for 2.4, but it really only makes sense to set minimum size hints here. We should probably replace `SetVirtualSizeHints` with `SetMinVirtualSize` or similar and remove it entirely in future.

As with all windows, an application can draw onto a `wxScrolledWindow` using a *device context* (p. 2120).

You have the option of handling the `OnPaint` handler or overriding the `OnDraw` (p. 1421) function, which is passed a pre-scrolled device context (prepared by `DoPrepareDC` (p. 1420)).

If you don't wish to calculate your own scrolling, you must call `DoPrepareDC` when not drawing from within `OnDraw`, to set the device origin for the device context according to the current scroll position.

A `wxScrolledWindow` will normally scroll itself and therefore its child windows as well. It might however be desired to scroll a different window than itself: e.g. when designing a spreadsheet, you will normally only have to scroll the (usually white) cell area, whereas the (usually grey) label area will scroll very differently. For this special purpose, you can call `SetTargetWindow` (p. 1423) which means that pressing the scrollbars will scroll a different window.

Note that the underlying system knows nothing about scrolling coordinates, so that all system functions (mouse events, expose events, refresh calls etc) as well as the position of subwindows are relative to the "physical" origin of the scrolled window. If the user insert a child window at position (10,10) and scrolls the window down 100 pixels (moving the child window out of the visible area), the child window will report a position of (10,-90).

### **Derived from**

`wxPanel` (p. 1170)  
`wxWindow` (p. 1795)  
`wxEvtHandler` (p. 576)  
`wxObject` (p. 1148)

**Include files**

<wx/scrolwin.h>

**Window styles**

**wxRETAINED** Uses a backing pixmap to speed refreshes. Motif only.

See also *window styles overview* (p. 2089).

**Remarks**

Use `wxScrolledWindow` for applications where the user scrolls by a fixed amount, and where a 'page' can be interpreted to be the current visible portion of the window. For more sophisticated applications, use the `wxScrolledWindow` implementation as a guide to build your own scroll behaviour.

**See also**

`wxScrollBar` (p. 1408), `wxClientDC` (p. 193),  
`wxPaintDC` (p. 1164), `wxVScrolledWindow` (p. 1790)

**wxScrolledWindow::wxScrolledWindow****wxScrolledWindow()**

Default constructor.

**wxScrolledWindow**(`wxWindow*` *parent*, `wxWindowID` *id* = -1, `const wxPoint&` *pos* = `wxDefaultPosition`, `const wxSize&` *size* = `wxDefaultSize`, `long` *style* = `wxHSCROLL | wxVSCROLL`, `const wxString&` *name* = "scrolledWindow")

Constructor.

**Parameters**

*parent*

Parent window.

*id*

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

*pos*

Window position. If a position of (-1, -1) is specified then a default position is chosen.

*size*

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

*style*



Window style. See *wxScrolledWindow* (p. 1414).

*name*

Window name.

### Remarks

The window is initially created without visible scrollbars. Call *wxScrolledWindow::SetScrollbars* (p. 1421) to specify how big the virtual window size should be.

### **wxScrolledWindow::~~wxScrolledWindow**

**~wxScrolledWindow()**

Destructor.

### **wxScrolledWindow::CalcScrolledPosition**

**void CalcScrolledPosition( int x, int y, int \*xx int \*yy) const**

Translates the logical coordinates to the device ones. For example, if a window is scrolled 10 pixels to the bottom, the device coordinates of the origin are (0, 0) (as always), but the logical coordinates are (0, 10) and so the call to *CalcScrolledPosition*(0, 10, &xx, &yy) will return 0 in yy.

### See also

*CalcUnscrolledPosition* (p. 1417)

**wxPython note:** The wxPython version of this methods accepts only two parameters and returns xx and yy as a tuple of values.

**wxPerl note:** In wxPerl this method takes two parameters and returns a 2-element list ( *xx*, *yy* ).

### **wxScrolledWindow::CalcUnscrolledPosition**

**void CalcUnscrolledPosition( int x, int y, int \*xx int \*yy) const**

Translates the device coordinates to the logical ones. For example, if a window is scrolled 10 pixels to the bottom, the device coordinates of the origin are (0, 0) (as always), but the logical coordinates are (0, 10) and so the call to *CalcUnscrolledPosition*(0, 0, &xx, &yy) will return 10 in yy.

### See also

*CalcScrolledPosition* (p. 1417)

**wxPython note:** The wxPython version of this methods accepts only two parameters and returns xx and yy as a tuple of values.

**wxPerl note:** In wxPerl this method takes two parameters and returns a 2-element list ( *xx*, *yy* ).

### **wxScrolledWindow::Create**

**bool Create**(*wxWindow\** parent, *wxWindowID* id = -1, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = *wxHSCROLL | wxVSCROLL*, **const wxString&** name = "scrolledWindow")

Creates the window for two-step construction. Derived classes should call or replace this function. See *wxScrolledWindow::wxScrolledWindow* (p. 1416) for details.

### **wxScrolledWindow::EnableScrolling**

**void EnableScrolling**(**const bool** xScrolling, **const bool** yScrolling)

Enable or disable physical scrolling in the given direction. Physical scrolling is the physical transfer of bits up or down the screen when a scroll event occurs. If the application scrolls by a variable amount (e.g. if there are different font sizes) then physical scrolling will not work, and you should switch it off. Note that you will have to reposition child windows yourself, if physical scrolling is disabled.

#### **Parameters**

*xScrolling*

If true, enables physical scrolling in the x direction.

*yScrolling*

If true, enables physical scrolling in the y direction.

#### **Remarks**

Physical scrolling may not be available on all platforms. Where it is available, it is enabled by default.

### **wxScrolledWindow::GetScrollPixelsPerUnit**

**void GetScrollPixelsPerUnit**(*int\** xUnit, *int\** yUnit) **const**

Get the number of pixels per scroll unit (line), in each direction, as set by *wxScrolledWindow::SetScrollbars* (p. 1421). A value of zero indicates no scrolling in that direction.

#### **Parameters**

*xUnit*

Receives the number of pixels per horizontal unit.

*yUnit*

Receives the number of pixels per vertical unit.

**See also**

*wxScrolledWindow::SetScrollbars* (p. 1421), *wxScrolledWindow::GetVirtualSize* (p. 1419)

**wxPython note:** The wxPython version of this methods accepts no parameters and returns a tuple of values for xUnit and yUnit.

**wxPerl note:** In wxPerl this method takes no parameters and returns a 2-element list ( xUnit, yUnit ).

**wxScrolledWindow::GetViewStart**

**void GetViewStart(int\* x, int\* y) const**

Get the position at which the visible portion of the window starts.

**Parameters**

*x*

Receives the first visible x position in scroll units.

*y*

Receives the first visible y position in scroll units.

**Remarks**

If either of the scrollbars is not at the home position, x and/or y will be greater than zero. Combined with *wxWindow::GetClientSize* (p. 1811), the application can use this function to efficiently redraw only the visible portion of the window. The positions are in logical scroll units, not pixels, so to convert to pixels you will have to multiply by the number of pixels per scroll increment.

**See also**

*wxScrolledWindow::SetScrollbars* (p. 1421)

**wxPython note:** The wxPython version of this methods accepts no parameters and returns a tuple of values for x and y.

**wxPerl note:** In wxPerl this method takes no parameters and returns a 2-element list ( x, y ).

**wxScrolledWindow::GetVirtualSize**

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

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

**Parameters**

*x*

Receives the length of the scrollable window, in pixels.

*y*

Receives the height of the scrollable window, in pixels.

### Remarks

Use *wxDC::DeviceToLogicalX* (p. 459) and *wxDC::DeviceToLogicalY* (p. 460) to translate these units to logical units.

### See also

*wxScrolledWindow::SetScrollbars* (p. 1421), *wxScrolledWindow::GetScrollPixelsPerUnit* (p. 1418)

**wxPython note:** The wxPython version of this methods accepts no parameters and returns a tuple of values for *x* and *y*.

**wxPerl note:** In wxPerl this method takes no parameters and returns a 2-element list ( *x*, *y* ).

### wxScrolledWindow::IsRetained

**bool IsRetained() const**

Motif only: true if the window has a backing bitmap.

### wxScrolledWindow::DoPrepareDC

**void DoPrepareDC(wxDC& dc)**

Call this function to prepare the device context for drawing a scrolled image. It sets the device origin according to the current scroll position.

DoPrepareDC is called automatically within the default *wxScrolledWindow::OnPaint* event handler, so your *wxScrolledWindow::OnDraw* (p. 1421) override will be passed a 'pre-scrolled' device context. However, if you wish to draw from outside of OnDraw (via OnPaint), or you wish to implement OnPaint yourself, you must call this function yourself. For example:

```
void MyWindow::OnEvent(wxMouseEvent& event)
{
    wxClientDC dc(this);
    DoPrepareDC(dc);

    dc.SetPen(*wxBLACK_PEN);
    float x, y;
    event.Position(&x, &y);
    if (xpos > -1 && ypos > -1 && event.Dragging())
    {
        dc.DrawLine(xpos, ypos, x, y);
    }
}
```

```
    }  
    xpos = x;  
    ypos = y;  
}
```

### **wxScrolledWindow::OnDraw**

**virtual void OnDraw(wxDC& dc)**

Called by the default paint event handler to allow the application to define painting behaviour without having to worry about calling *wxScrolledWindow::DoPrepareDC* (p. 1420).

Instead of overriding this function you may also just process the paint event in the derived class as usual, but then you will have to call *DoPrepareDC()* yourself.

### **wxScrolledWindow::PrepareDC**

**void PrepareDC(wxDC& dc)**

This function is for backwards compatibility only and simply calls *DoPrepareDC* (p. 1420) now. Notice that it is *not* called by the default paint event handle (*DoPrepareDC()* is), so overriding this method in your derived class is useless.

### **wxScrolledWindow::Scroll**

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

Scrolls a window so the view start is at the given point.

#### **Parameters**

*x*

The x position to scroll to, in scroll units.

*y*

The y position to scroll to, in scroll units.

#### **Remarks**

The positions are in scroll units, not pixels, so to convert to pixels you will have to multiply by the number of pixels per scroll increment. If either parameter is -1, that position will be ignored (no change in that direction).

#### **See also**

*wxScrolledWindow::SetScrollbars* (p. 1421), *wxScrolledWindow::GetScrollPixelsPerUnit* (p. 1418)

### **wxScrolledWindow::SetScrollbars**

**void SetScrollbars**(*int pixelsPerUnitX*, *int pixelsPerUnitY*, *int noUnitsX*, *int noUnitsY*, *int xPos = 0*, *int yPos = 0*, **bool noRefresh** = *false*)

Sets up vertical and/or horizontal scrollbars.

### Parameters

*pixelsPerUnitX*

Pixels per scroll unit in the horizontal direction.

*pixelsPerUnitY*

Pixels per scroll unit in the vertical direction.

*noUnitsX*

Number of units in the horizontal direction.

*noUnitsY*

Number of units in the vertical direction.

*xPos*

Position to initialize the scrollbars in the horizontal direction, in scroll units.

*yPos*

Position to initialize the scrollbars in the vertical direction, in scroll units.

*noRefresh*

Will not refresh window if true.

### Remarks

The first pair of parameters give the number of pixels per 'scroll step', i.e. amount moved when the up or down scroll arrows are pressed. The second pair gives the length of scrollbar in scroll steps, which sets the size of the virtual window.

*xPos* and *yPos* optionally specify a position to scroll to immediately.

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

```
window->SetScrollbars(20, 20, 50, 50);
```

`wxScrolledWindow` manages the page size itself, using the current client window size as the page size.

Note that for more sophisticated scrolling applications, for example where scroll steps may be variable according to the position in the document, it will be necessary to derive a new class from `wxWindow`, overriding **OnSize** and adjusting the scrollbars appropriately.

### See also

*wxWindow::SetVirtualSize* (p. 1848)

### **wxScrolledWindow::SetScrollRate**

**void SetScrollRate**(int *xstep*, int *ystep*)

Set the horizontal and vertical scrolling increment only. See the `pixelsPerUnit` parameter in `SetScrollbars`.

### **wxScrolledWindow::SetTargetWindow**

**void SetTargetWindow**(*wxWindow\** *window*)

Call this function to tell `wxScrolledWindow` to perform the actual scrolling on a different window (and not on itself).

## **wxScrollEvent**

A scroll event holds information about events sent from stand-alone *scrollbars* (p. 1408) and *sliders* (p. 1462). Note that starting from `wxWidgets 2.1`, scrolled windows send the *wxScrollWinEvent* (p. 1426) which does not derive from `wxCommandEvent`, but from `wxEvent` directly - don't confuse these two kinds of events and use the event table macros mentioned below only for the scrollbar-like controls.

### **Derived from**

*wxCommandEvent* (p. 250)

*wxEvent* (p. 572)

*wxObject* (p. 1148)

### **Include files**

<wx/event.h>

### **Event table macros**

To process a scroll event, use these event handler macros to direct input to member functions that take a `wxScrollEvent` argument. You can use `EVT_COMMAND_SCROLL` . . . macros with window IDs for when intercepting scroll events from controls, or `EVT_SCROLL` . . . macros without window IDs for intercepting scroll events from the receiving window -- except for this, the macros behave exactly the same.

<b>EVT_SCROLL(func)</b>	Process all scroll events.
<b>EVT_SCROLL_TOP(func)</b>	Process <code>wxEVT_SCROLL_TOP</code> scroll-to-top events (minimum position).
<b>EVT_SCROLL_BOTTOM(func)</b>	Process <code>wxEVT_SCROLL_BOTTOM</code> scroll-to-bottom events (maximum position).

	position).
<b>EVT_SCROLL_LINEUP(func)</b>	Process wxEVT_SCROLL_LINEUP line up events.
<b>EVT_SCROLL_LINEDOWN(func)</b>	Process wxEVT_SCROLL_LINEDOWN line down events.
<b>EVT_SCROLL_PAGEUP(func)</b>	Process wxEVT_SCROLL_PAGEUP page up events.
<b>EVT_SCROLL_PAGEDOWN(func)</b>	Process wxEVT_SCROLL_PAGEDOWN page down events.
<b>EVT_SCROLL_THUMBTRACK(func)</b>	Process wxEVT_SCROLL_THUMBTRACK thumbtrack events (frequent events sent as the user drags the thumbtrack).
<b>EVT_SCROLL_THUMBRELEASE(func)</b>	Process wxEVT_SCROLL_THUMBRELEASE thumb release events.
<b>EVT_SCROLL_CHANGED(func)</b>	Process wxEVT_SCROLL_CHANGED end of scrolling events (MSW only).
<b>EVT_COMMAND_SCROLL(id, func)</b>	Process all scroll events.
<b>EVT_COMMAND_SCROLL_TOP(id, func)</b>	Process wxEVT_SCROLL_TOP scroll-to-top events (minimum position).
<b>EVT_COMMAND_SCROLL_BOTTOM(id, func)</b>	Process wxEVT_SCROLL_BOTTOM scroll-to-bottom events (maximum position).
<b>EVT_COMMAND_SCROLL_LINEUP(id, func)</b>	Process wxEVT_SCROLL_LINEUP line up events.
<b>EVT_COMMAND_SCROLL_LINEDOWN(id, func)</b>	Process wxEVT_SCROLL_LINEDOWN line down events.
<b>EVT_COMMAND_SCROLL_PAGEUP(id, func)</b>	Process wxEVT_SCROLL_PAGEUP page up events.
<b>EVT_COMMAND_SCROLL_PAGEDOWN(id, func)</b>	Process



`wxEVT_SCROLL_PAGEDOWN`  
page down events.

**`EVT_COMMAND_SCROLL_THUMBTRACK(id, func)`**     Process  
`wxEVT_SCROLL_THUMBTRACK`  
thumbtrack events (frequent events  
sent as the user drags the  
thumbtrack).

**`EVT_COMMAND_SCROLL_THUMBRELEASE(func)`**     Process  
`wxEVT_SCROLL_THUMBRELEASE`  
thumb release events.

**`EVT_COMMAND_SCROLL_CHANGED(func)`**     Process  
`wxEVT_SCROLL_CHANGED` end  
of scrolling events (MSW only).

#### **The difference between `EVT_SCROLL_THUMBRELEASE` and `EVT_SCROLL_CHANGED`**

The `EVT_SCROLL_THUMBRELEASE` event is only emitted when actually dragging the thumb using the mouse and releasing it (This `EVT_SCROLL_THUMBRELEASE` event is also followed by an `EVT_SCROLL_CHANGED` event).

The `EVT_SCROLL_CHANGED` event also occurs when using the keyboard to change the thumb position, and when clicking next to the thumb (In all these cases the `EVT_SCROLL_THUMBRELEASE` event does not happen).

In short, the `EVT_SCROLL_CHANGED` event is triggered when scrolling/ moving has finished independently of the way it had started. Please see the widgets sample ("Slider" page) to see the difference between `EVT_SCROLL_THUMBRELEASE` and `EVT_SCROLL_CHANGED` in action.

#### **Remarks**

Note that unless specifying a scroll control identifier, you will need to test for scrollbar orientation with `wxScrollEvent::GetOrientation` (p. 1425), since horizontal and vertical scroll events are processed using the same event handler.

#### **See also**

`wxScrollBar` (p. 1408), `wxSlider` (p. 1462), `wxSpinButton` (p. 1496),  
`wxScrollWinEvent` (p. 1426), *Event handling overview* (p. 2077)

#### **`wxScrollEvent::wxScrollEvent`**

**`wxScrollEvent(WXTYPE commandType = 0, int id = 0, int pos = 0, int orientation = 0)`**

Constructor.

#### **`wxScrollEvent::GetOrientation`**

**int GetOrientation() const**

Returns wxHORIZONTAL or wxVERTICAL, depending on the orientation of the scrollbar.

**wxScrollEvent::GetPosition****int GetPosition() const**

Returns the position of the scrollbar.

**wxScrollWinEvent**

A scroll event holds information about events sent from scrolling windows.

**Derived from**

*wxEvent* (p. 572)

*wxObject* (p. 1148)

**Include files**

<wx/event.h>

**Event table macros**

To process a scroll window event, use these event handler macros to direct input to member functions that take a wxScrollWinEvent argument. You can use the EVT\_SCROLLWIN... macros for intercepting scroll window events from the receiving window.

<b>EVT_SCROLLWIN(func)</b>	Process all scroll events.
<b>EVT_SCROLLWIN_TOP(func)</b>	Process wxEVT_SCROLLWIN_TOP scroll-to-top events.
<b>EVT_SCROLLWIN_BOTTOM(func)</b>	Process wxEVT_SCROLLWIN_BOTTOM scroll-to-bottom events.
<b>EVT_SCROLLWIN_LINEUP(func)</b>	Process wxEVT_SCROLLWIN_LINEUP line up events.
<b>EVT_SCROLLWIN_LINEDOWN(func)</b>	Process wxEVT_SCROLLWIN_LINEDOWN line down events.
<b>EVT_SCROLLWIN_PAGEUP(func)</b>	Process wxEVT_SCROLLWIN_PAGEUP page up events.
<b>EVT_SCROLLWIN_PAGEDOWN(func)</b>	Process wxEVT_SCROLLWIN_PAGEDOWN page down events.
<b>EVT_SCROLLWIN_THUMBTRACK(func)</b>	Process wxEVT_SCROLLWIN_THUMBTRACK thumbtrack events (frequent events sent as the

user drags the thumbtrack).

**EVT\_SCROLLWIN\_THUMBRELEASE(func)**      Process  
wxEVT\_SCROLLWIN\_THUMBRELEASE  
thumb release events.

**See also**

*wxScrollEvent* (p. 1423), *Event handling overview* (p. 2077)

### **wxScrollWinEvent::wxScrollWinEvent**

**wxScrollWinEvent(WXTYPE *commandType* = 0, int *pos* = 0, int *orientation* = 0)**

Constructor.

### **wxScrollWinEvent::GetOrientation**

**int GetOrientation() const**

Returns wxHORIZONTAL or wxVERTICAL, depending on the orientation of the scrollbar.

### **wxScrollWinEvent::GetPosition**

**int GetPosition() const**

Returns the position of the scrollbar for the thumb track and release events. Note that this field can't be used for the other events, you need to query the window itself for the current position in that case.

## **wxSemaphore**

wxSemaphore is a counter limiting the number of threads concurrently accessing a shared resource. This counter is always between 0 and the maximum value specified during the semaphore creation. When the counter is strictly greater than 0, a call to *Wait* (p. 1429) returns immediately and decrements the counter. As soon as it reaches 0, any subsequent calls to *Wait* (p. 1429) block and only return when the semaphore counter becomes strictly positive again as the result of calling *Post* (p. 1428) which increments the counter.

In general, semaphores are useful to restrict access to a shared resource which can only be accessed by some fixed number of clients at the same time. For example, when modeling a hotel reservation system a semaphore with the counter equal to the total number of available rooms could be created. Each time a room is reserved, the semaphore should be acquired by calling *Wait* (p. 1429) and each time a room is freed it should be released by calling *Post* (p. 1428).

### **Derived from**

No base class

**Include files**

<wx/thread.h>

**wxSemaphore::wxSemaphore**

**wxSemaphore**(int *initialcount* = 0, int *maxcount* = 0)

Specifying a *maxcount* of 0 actually makes wxSemaphore behave as if there is no upper limit. If *maxcount* is 1, the semaphore behaves exactly as a mutex.

*initialcount* is the initial value of the semaphore which must be between 0 and *maxcount* (if it is not set to 0).

**wxSemaphore::~~wxSemaphore**

**~wxSemaphore**()

Destructor is not virtual, don't use this class polymorphically.

**wxSemaphore::Post**

**wxSemaError Post**()

Increments the semaphore count and signals one of the waiting threads in an atomic way. Returns wxSEMA\_OVERFLOW if the count would increase the counter past the maximum.

**Return value**

One of:

<b>wxSEMA_NO_ERROR</b>	There was no error.
<b>wxSEMA_INVALID</b>	Semaphore hasn't been initialized successfully.
<b>wxSEMA_OVERFLOW</b>	Post() would increase counter past the max.
<b>wxSEMA_MISC_ERROR</b>	Miscellaneous error.

**wxSemaphore::TryWait**

**wxSemaError TryWait**()

Same as *Wait*() (p. 1429), but returns immediately.

**Return value**

One of:

<b>wxSEMA_NO_ERROR</b>	There was no error.
------------------------	---------------------

<b>wxSEMA_INVALID</b>	Semaphore hasn't been initialized successfully.
<b>wxSEMA_BUSY</b>	Returned by TryWait() if Wait() would block, i.e. the count is zero.
<b>wxSEMA_MISC_ERROR</b>	Miscellaneous error.

### **wxSemaphore::Wait**

#### **wxSemaError Wait()**

Wait indefinitely until the semaphore count becomes strictly positive and then decrement it and return.

#### **Return value**

One of:

<b>wxSEMA_NO_ERROR</b>	There was no error.
<b>wxSEMA_INVALID</b>	Semaphore hasn't been initialized successfully.
<b>wxSEMA_MISC_ERROR</b>	Miscellaneous error.

### **wxSemaphore::WaitTimeout**

#### **wxSemaError WaitTimeout(unsigned long *timeout\_millis*)**

Same as *Wait()* (p. 1429), but with a timeout limit.

#### **Return value**

One of:

<b>wxSEMA_NO_ERROR</b>	There was no error.
<b>wxSEMA_INVALID</b>	Semaphore hasn't been initialized successfully.
<b>wxSEMA_TIMEOUT</b>	Timeout occurred without receiving semaphore.
<b>wxSEMA_MISC_ERROR</b>	Miscellaneous error.

## **wxSetCursorEvent**

A SetCursorEvent is generated when the mouse cursor is about to be set as a result of mouse motion. This event gives the application the chance to perform specific mouse cursor processing based on the current position of the mouse within the window. Use *SetCursor* (p. 1430) to specify the cursor you want to be displayed.

#### **Derived from**

*wxEvent* (p. 572)  
*wxObject* (p. 1148)

**Include files**

<wx/event.h>

**See also**

`::wxSetCursor` (p. 1946) `wxWindow::wxSetCursor` (p. 1837)

**Event table macros**

To process a set cursor event, use this event handler macro to direct input to a member function that takes a `wxSetCursorEvent` argument.

**EVT\_SET\_CURSOR(func)**                      Process a `wxEVT_SET_CURSOR` event.

**wxSetCursorEvent::wxSetCursorEvent**

**wxSetCursorEvent(wxCoord x = 0, wxCoord y = 0)**

Constructor, used by the library itself internally to initialize the event object.

**wxSetCursorEvent::GetCursor**

**wxCursor& GetCursor() const**

Returns a reference to the cursor specified by this event.

**wxSetCursorEvent::GetX**

**wxCoord GetX() const**

Returns the X coordinate of the mouse in client coordinates.

**wxSetCursorEvent::GetY**

**wxCoord GetY() const**

Returns the Y coordinate of the mouse in client coordinates.

**wxSetCursorEvent::HasCursor**

**bool HasCursor() const**

Returns `true` if the cursor specified by this event is a valid cursor.

**Remarks**

You cannot specify `wxNullCursor` with this event, as it is not considered a valid cursor.

**wxSetCursorEvent::SetCursor**

**void SetCursor(const wxCursor&cursor)**

Sets the cursor associated with this event.

## wxServer

A wxServer object represents the server part of a client-server DDE-like (Dynamic Data Exchange) conversation. The actual DDE-based implementation using wxDDEServer is available on Windows only, but a platform-independent, socket-based version of this API is available using wxTCPServer, which has the same API.

To create a server which can communicate with a suitable client, you need to derive a class from wxConnection and another from wxServer. The custom wxConnection class will intercept communications in a 'conversation' with a client, and the custom wxServer is required so that a user-overridden *wxServer::OnAcceptConnection* (p. 1431) member can return a wxConnection of the required class, when a connection is made. Look at the IPC sample and the *Interprocess communications overview* (p. 2175) for an example of how to do this.

### Derived from

wxServerBase

### Include files

<wx/ipc.h>

### See also

*wxClient* (p. 191), *wxConnection* (p. 478), *IPC overview* (p. 2175)

## wxServer::wxServer

**wxServer()**

Constructs a server object.

## wxServer::Create

**bool Create(const wxString& service)**

Registers the server using the given service name. Under Unix, the service name may be either an integer port identifier in which case an Internet domain socket will be used for the communications, or a valid file name (which shouldn't exist and will be deleted afterwards) in which case a Unix domain socket is created. false is returned if the call failed (for example, the port number is already in use).

## wxServer::OnAcceptConnection

**virtual wxConnectionBase \* OnAcceptConnection(const wxString& topic)**

When a client calls **MakeConnection**, the server receives the message and this member is called. The application should derive a member to intercept this message and return a connection object of either the standard `wxConnection` type, or (more likely) 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.

## wxSimpleHelpProvider

`wxSimpleHelpProvider` is an implementation of `wxHelpProvider` (p. 817) which supports only plain text help strings, and shows the string associated with the control (if any) in a tooltip.

### Derived from

`wxHelpProvider` (p. 817)

### Include files

<wx/cshelp.h>

### See also

`wxHelpProvider` (p. 817), `wxHelpControllerHelpProvider` (p. 815), `wxContextHelp` (p. 282), `wxWindow::SetHelpText` (p. 1841), `wxWindow::GetHelpTextAtPoint` (p. 1814)

## wxSearchCtrl

A search control is a composite control with a search button, a text control, and a cancel button.

### Derived from

`wxTextCtrl` (p. 1633)  
`streambuf`  
`wxControl` (p. 285)  
`wxWindow` (p. 1795)  
`wxEvtHandler` (p. 576)  
`wxObject` (p. 1148)

### Include files

<wx/srchctrl.h>

### Window styles

**wxTE\_PROCESS\_ENTER** The control will generate the event  
`wxEVT_COMMAND_TEXT_ENTER` (otherwise pressing



	Enter key is either processed internally by the control or used for navigation between dialog controls).
<b>wxTE_PROCESS_TAB</b>	The control will receive wxEVT_CHAR events for TAB pressed - normally, TAB is used for passing to the next control in a dialog instead. For the control created with this style, you can still use Ctrl-Enter to pass to the next control from the keyboard.
<b>wxTE_NOHIDESEL</b>	By default, the Windows text control doesn't show the selection when it doesn't have focus - use this style to force it to always show it. It doesn't do anything under other platforms.
<b>wxTE_LEFT</b>	The text in the control will be left-justified (default).
<b>wxTE_CENTRE</b>	The text in the control will be centered (currently wxMSW and wxGTK2 only).
<b>wxTE_RIGHT</b>	The text in the control will be right-justified (currently wxMSW and wxGTK2 only).
<b>wxTE_CAPITALIZE</b>	On PocketPC and Smartphone, causes the first letter to be capitalized.

See also *window styles overview* (p. 2089) and *wxSearchCtrl::wxSearchCtrl* (p. 1433).

### Event handling

To process input from a search control, use these event handler macros to direct input to member functions that take a *wxCommandEvent* (p. 250) argument. To retrieve actual search queries, use EVT\_TEXT and EVT\_TEXT\_ENTER events, just as you would with *wxTextCtrl* (p. 1633).

<b>EVT_SEARCHCTRL_SEARCH_BTN(id, func)</b>	Respond to a wxEVT_SEARCHCTRL_SEARCH_BTN event, generated when the search button is clicked. Note that this does not initiate a search.
<b>EVT_SEARCHCTRL_CANCEL_BTN(id, func)</b>	Respond to a wxEVT_SEARCHCTRL_CANCEL_BTN event, generated when the cancel button is clicked.

### wxSearchCtrl::wxSearchCtrl

**wxSearchCtrl()**

Default constructor.

**wxSearchCtrl(wxWindow\* parent, wxWindowID id, const wxString& value = "", const**

**wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = 0, **const wxValidator&** *validator* = *wxDefaultValidator*, **const wxString&** *name* = *wxSearchCtrlNameStr*)

Constructor, creating and showing a text control.

### Parameters

*parent*

Parent window. Should not be NULL.

*id*

Control identifier. A value of -1 denotes a default value.

*value*

Default text value.

*pos*

Text control position.

*size*

Text control size.

*style*

Window style. See *wxSearchCtrl* (p. 1432).

*validator*

Window validator.

*name*

Window name.

### See also

*wxTextCtrl::Create* (p. 1640), *wxValidator* (p. 1767)

### **wxSearchCtrl::~~wxSearchCtrl**

**~wxSearchCtrl()**

Destructor, destroying the search control.

### **wxSearchCtrl::SetMenu**

**virtual void SetMenu(wxMenu\* menu)**

Sets the search control's menu object. If there is already a menu associated with the

search control it is deleted.

### Parameters

*menu*

Menu to attach to the search control.

### **wxSearchCtrl::GetMenu**

**virtual wxMenu\* GetMenu()**

Returns a pointer to the search control's menu object or NULL if there is no menu attached.

### **wxSearchCtrl::ShowSearchButton**

**virtual void ShowSearchButton(bool show)**

Sets the search button visibility value on the search control. If there is a menu attached, the search button will be visible regardless of the search button visibility value.

This has no effect in Mac OS X v10.3

### **wxSearchCtrl::IsSearchButtonVisible**

**virtual bool IsSearchButtonVisible()**

Returns the search button visibility value. If there is a menu attached, the search button will be visible regardless of the search button visibility value.

This always returns false in Mac OS X v10.3

### **wxSearchCtrl::ShowCancelButton**

**virtual void ShowCancelButton(bool show)**

Shows or hides the cancel button.

### **wxSearchCtrl::IsCancelButtonVisible**

**virtual bool IsCancelButtonVisible()**

Indicates whether the cancel button is visible.

## **wxSingleChoiceDialog**

This class represents a dialog that shows a list of strings, and allows the user to select one. Double-clicking on a list item is equivalent to single-clicking and then pressing OK.

**Derived from**

*wxDialog* (p. 496)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

#### Include files

<wx/choicdlg.h>

#### See also

*wxSingleChoiceDialog* overview (p. 2131), *wxMultiChoiceDialog* (p. 1129)

### **wxSingleChoiceDialog::wxSingleChoiceDialog**

**wxSingleChoiceDialog**(*wxWindow\** parent, **const wxString&** message, **const wxString&** caption, **int** n, **const wxString\*** choices, **void\*\*** clientData = NULL, **long** style = wxCHOICEDLG\_STYLE, **const wxPoint&** pos = wxDefaultPosition)

**wxSingleChoiceDialog**(*wxWindow\** parent, **const wxString&** message, **const wxString&** caption, **const wxArrayString&** choices, **void\*\*** clientData = NULL, **long** style = wxCHOICEDLG\_STYLE, **const wxPoint&** pos = wxDefaultPosition)

Constructor, taking an array of wxString choices and optional client data.

#### Parameters

*parent*

Parent window.

*message*

Message to show on the dialog.

*caption*

The dialog caption.

*n*

The number of choices.

*choices*

An array of strings, or a string list, containing the choices.

*clientData*

An array of client data to be associated with the items. See *GetSelectionClientData* (p. 1437).

*style*

A dialog style (bitlist) containing flags chosen from standard dialog styles and the following:

<b>wxOK</b>	Show an OK button.
<b>wxCANCEL</b>	Show a Cancel button.
<b>wxCENTRE</b>	Centre the message. Not Windows.

The default value is equivalent to **wxDEFAULT\_DIALOG\_STYLE | wxRESIZE\_BORDER | wxOK | wxCANCEL | wxCENTRE**.  
*pos*

Dialog position. Not Windows.

### Remarks

Use *wxSingleChoiceDialog::ShowModal* (p. 1437) to show the dialog.

**wxPython note:** For Python the two parameters *n* and *choices* are collapsed into a single parameter *choices* which is expected to be a Python list of strings.

**wxPerl note:** In wxPerl there is just an array reference in place of *n* and *choices*, and the client data array, if present, must have the same length as the choices array.

### **wxSingleChoiceDialog::GetSelection**

**int GetSelection() const**

Returns the index of selected item.

### **wxSingleChoiceDialog::GetSelectionClientData**

**char\* GetSelectionClientData() const**

Returns the client data associated with the selection.

### **wxSingleChoiceDialog::GetStringSelection**

**wxString GetStringSelection() const**

Returns the selected string.

### **wxSingleChoiceDialog::SetSelection**

**void SetSelection(int selection) const**

Sets the index of the initially selected item.

### **wxSingleChoiceDialog::ShowModal**

**int ShowModal()**

Shows the dialog, returning either `wxID_OK` or `wxID_CANCEL`.

## wxSingleInstanceChecker

`wxSingleInstanceChecker` class allows to check that only a single instance of a program is running. To do it, you should create an object of this class. As long as this object is alive, calls to *IsAnotherRunning()* (p. 1439) from other processes will return `true`.

As the object should have the life span as big as possible, it makes sense to create it either as a global or in *wxApp::OnInit* (p. 52). For example:

```
bool MyApp::OnInit()
{
    const wxString name = wxString::Format("MyApp-%s",
wxGetUserId().c_str());
    m_checker = new wxSingleInstanceChecker(name);
    if ( m_checker->IsAnotherRunning() )
    {
        wxLogError(_("Another program instance is already running,
aborting."));

        return false;
    }

    ... more initializations ...

    return true;
}

int MyApp::OnExit()
{
    delete m_checker;

    return 0;
}
```

Note using *wxGetUserId()* (p. 1928) to construct the name: this allows different user to run the application concurrently which is usually the intended goal. If you don't use the user name in the `wxSingleInstanceChecker` name, only one user would be able to run the application at a time.

This class is implemented for Win32 and Unix platforms (supporting `fcntl()` system call, but almost all of modern Unix systems do) only.

### Derived from

No base class

### Include files

<wx/snglinst.h>

**wxSingleInstanceChecker::wxSingleInstanceChecker****wxSingleInstanceChecker()**

Default ctor, use *Create()* (p. 1439) after it.

**wxSingleInstanceChecker::wxSingleInstanceChecker****wxSingleInstanceChecker(const wxString& name, const wxString& path = wxEmptyString)**

Like *Create()* (p. 1439) but without error checking.

**wxSingleInstanceChecker::Create****bool Create(const wxString& name, const wxString& path = wxEmptyString)**

Initialize the object if it had been created using the default constructor. Note that you can't call *Create()* more than once, so calling it if the *non default ctor* (p. 1439) had been used is an error.

**Parameters**

*name*

must be given and be as unique as possible. It is used as the mutex name under Win32 and the lock file name under Unix. *GetAppName()* (p. 47) and *wxGetUserId()* (p. 1928) are commonly used to construct this parameter.

*path*

is optional and is ignored under Win32 and used as the directory to create the lock file in under Unix (default is *wxGetHomeDir()* (p. 1926))

**Return value**

Returns *false* if initialization failed, it doesn't mean that another instance is running - use *IsAnotherRunning()* (p. 1439) to check for it.

**Note**

One of possible reasons while *Create* may fail on Unix is that the lock file used for checking already exists but was not created by the user. Therefore applications shouldn't treat failure of this function as fatal condition, because doing so would open them to the possibility of a Denial of Service attack. Instead, they should alert the user about the problem and offer to continue execution without checking if another instance is running.

**wxSingleInstanceChecker::IsAnotherRunning****bool IsAnotherRunning() const**

Returns *true* if another copy of this program is already running, *false* otherwise.

**wxSingleInstanceChecker::~~wxSingleInstanceChecker****~wxSingleInstanceChecker()**

Destructor frees the associated resources.

Note that it is not virtual, this class is not meant to be used polymorphically

**wxSize**

A **wxSize** is a useful data structure for graphics operations. It simply contains integer *width* and *height* members.

wxSize is used throughout wxWidgets as well as wxPoint which, although almost equivalent to wxSize, has a different meaning: wxPoint represents a position while wxSize - the size.

**wxPython note:** wxPython defines aliases for the *x* and *y* members named *width* and *height* since it makes much more sense for sizes.

**Derived from**

None

**Include files**

<wx/gdicmn.h>

**See also**

*wxPoint* (p. 1193), *wxRealPoint* (p. 1251)

**wxSize::wxSize****wxSize()****wxSize(int width, int height)**

Creates a size object.

**wxSize::DecBy****void DecBy(const wxSize& size)****void DecBy(int dx, int dy)****void DecBy(int d)**

Decreases the size in x- and y- directions

1. By *size.x* and *size.y* for the first overload



2. By  $dx$  and  $dy$  for the second one
3. By  $d$  and  $d$  for the third one

**See also**

*IncBy* (p. 1441)

**wxSize::DecTo**

**void DecTo(const wxSize& size)**

Decrements this object so that both of its dimensions are not greater than the corresponding dimensions of the *size*.

**See also**

*IncTo* (p. 1442)

**wxSize::IsFullySpecified**

**bool IsFullySpecified() const**

Returns `true` if neither of the size object components is equal to -1, which is used as default for the size values in `wxWidgets` (hence the predefined `wxDefaultSize` has both of its components equal to -1).

This method is typically used before calling *SetDefaults* (p. 1442).

**wxSize::GetWidth**

**int GetWidth() const**

Gets the width member.

**wxSize::GetHeight**

**int GetHeight() const**

Gets the height member.

**wxSize::IncBy**

**void IncBy(const wxSize& size)**

**void IncBy(int dx, int dy)**

**void IncBy(int d)**

Increases the size in x- and y- directions

1. By *size.x* and *size.y* for the first overload

2. By  $dx$  and  $dy$  for the second one
3. By  $d$  and  $d$  for the third one

**See also**

*DecBy* (p. 1440)

**wxSize::IncTo**

**void IncTo(const wxSize& size)**

Increments this object so that both of its dimensions are not less than the corresponding dimensions of the *size*.

**See also**

*DecTo* (p. 1441)

**wxSize::Scale**

**wxSize& Scale(float xscale, float yscale)**

Scales the dimensions of this object by the given factors. If you want to scale both dimensions by the same factor you can also use the *operator \*=* (p. 1443)

Returns a reference to this object (so that you can concatenate other operations in the same line).

**wxSize::Set**

**void Set(int width, int height)**

Sets the width and height members.

**wxSize::SetDefaults**

**void SetDefaults(const wxSize& sizeDefault)**

Combine this size object with another one replacing the default (i.e. equal to -1) components of this object with those of the other. It is typically used like this: if

```
( !size.IsFullySpecified() )  
{  
    size.SetDefaults(GetDefaultSize());  
}
```

**See also**

*IsFullySpecified* (p. 1441)

**wxSize::SetHeight**

**void SetHeight(int height)**

Sets the height.

**wxSize::SetWidth**

**void SetWidth(int width)**

Sets the width.

## Operators

**void operator =(const wxSize& sz)**

Assignment operator.

**bool operator ==(const wxSize& sz) const**

**bool operator !=(const wxSize& sz) const**

**wxSize operator +(const wxSize& sz)**

**wxSize operator -(const wxSize& sz)**

**wxSize& operator +=(const wxSize& sz)**

**wxSize& operator -=(const wxSize& sz)**

Operators for comparison, sum and subtraction between *wxSize* (p. 1440) objects.

**wxSize operator /(int factor)**

**wxSize operator \*(int factor)**

**wxSize& operator /=(int factor)**

**wxSize& operator \*=(int factor)**

Operators for division and multiplication between a *wxSize* (p. 1440) object and an integer.

## wxSizeEvent

A size event holds information about size change events.

The EVT\_SIZE handler function will be called when the window has been resized.

You may wish to use this for frames to resize their child windows as appropriate.

Note that the size passed is of the whole window: call *wxWindow::GetClientSize* (p. 1811) for the area which may be used by the application.

When a window is resized, usually only a small part of the window is damaged and you may only need to repaint that area. However, if your drawing depends on the size of the

window, you may need to clear the DC explicitly and repaint the whole window. In which case, you may need to call `wxWindow::Refresh` (p. 1830) to invalidate the entire window.

**Derived from**

*wxEvt* (p. 572)

*wxObject* (p. 1148)

## Include files

&lt;wx/event.h&gt;

## Event table macros

To process a size event, use this event handler macro to direct input to a member function that takes a `wxSizeEvent` argument.

<b>EVT_SIZE(func)</b>	Process a wxEVT_SIZE event.
-----------------------	-----------------------------

## See also

*wxSize* (p. 1440), *Event handling overview* (p. 2077)

## wxSizeEvent::wxSizeEvent

**wxSizeEvent(const wxSize& sz, int id = 0)**

### Constructor.

## wxSizeEvent::GetSize

**wxSize GetSize() const**

Returns the entire size of the window generating the size change event.

## wxSizer

`wxSizer` is the abstract base class used for laying out subwindows in a window. You cannot use `wxSizer` directly; instead, you will have to use one of the sizer classes derived from it. Currently there are `wxBoxSizer` (p. 149), `wxStaticBoxSizer` (p. 1532), `wxGridSizer` (p. 797) `wxFlexGridSizer` (p. 652) and `wxGridBagSizer` (p. 772).

The layout algorithm used by sizers in wxWidgets is closely related to layout in other GUI toolkits, such as Java's AWT, the GTK toolkit or the Qt toolkit. It is based upon the idea of the individual subwindows reporting their minimal required size and their ability to get stretched if the size of the parent window has changed. This will most often mean that the programmer does not set the original size of a dialog in the beginning, rather the dialog will be assigned a sizer and this sizer will be queried about the recommended size. The sizer in turn will query its children, which can be normal windows, empty space or other sizers, so that a hierarchy of sizers can be constructed. Note that wxSizer does not derive from wxWindow and thus does not interfere with tab ordering and requires very little resources

compared to a real window on screen.

What makes sizers so well fitted for use in wxWidgets is the fact that every control reports its own minimal size and the algorithm can handle differences in font sizes or different window (dialog item) sizes on different platforms without problems. If e.g. the standard font as well as the overall design of Motif widgets requires more space than on Windows, the initial dialog size will automatically be bigger on Motif than on Windows.

Sizers may also be used to control the layout of custom drawn items on the window. The Add, Insert, and Prepend functions return a pointer to the newly added wxSizerItem. Just add empty space of the desired size and attributes, and then use the wxSizerItem::GetRect method to determine where the drawing operations should take place.

Please notice that sizers, like child windows, are owned by the library and will be deleted by it which implies that they must be allocated on the heap. However if you create a sizer and do not add it to another sizer or window, the library wouldn't be able to delete such an orphan sizer and in this, and only this, case it should be deleted explicitly.

**wxPython note:** If you wish to create a sizer class in wxPython you should derive the class from wxPySizer in order to get Python-aware capabilities for the various virtual methods.

#### **Derived from**

*wxObject* (p. 1148)

*wxClientDataContainer* (p. 194)

#### **Include files**

<wx/sizer.h>

#### **See also**

*Sizer overview* (p. 2098)

#### **wxSizer::wxSizer**

**wxSizer()**

The constructor. Note that wxSizer is an abstract base class and may not be instantiated.

#### **wxSizer::~~wxSizer**

**~wxSizer()**

The destructor.

#### **wxSizer::Add**

**wxSizerItem\* Add(wxWindow\* window, const wxSizerFlags& flags)**

**wxSizerItem\* Add(wxWindow\* window, int proportion = 0, int flag = 0, int border = 0, wxObject\* userData = NULL)**

**wxSizerItem\* Add(wxSizer\* sizer, const wxSizerFlags& flags)**

**wxSizerItem\* Add(wxSizer\* sizer, int proportion = 0, int flag = 0, int border = 0, wxObject\* userData = NULL)**

**wxSizerItem\* Add(int width, int height, int proportion = 0, int flag = 0, int border = 0, wxObject\* userData = NULL)**

Appends a child to the sizer. `wxSizer` itself is an abstract class, but the parameters are equivalent in the derived classes that you will instantiate to use it so they are described here:

*window*

The window to be added to the sizer. Its initial size (either set explicitly by the user or calculated internally when using `wxDefaultSize`) is interpreted as the minimal and in many cases also the initial size.

*sizer*

The (child-)sizer to be added to the sizer. This allows placing a child sizer in a sizer and thus to create hierarchies of sizers (typically a vertical box as the top sizer and several horizontal boxes on the level beneath).

*width and height*

The dimension of a spacer to be added to the sizer. Adding spacers to sizers gives more flexibility in the design of dialogs; imagine for example a horizontal box with two buttons at the bottom of a dialog: you might want to insert a space between the two buttons and make that space stretchable using the *proportion* flag and the result will be that the left button will be aligned with the left side of the dialog and the right button with the right side - the space in between will shrink and grow with the dialog.

*proportion*

Although the meaning of this parameter is undefined in `wxSizer`, it is used in `wxBoxSizer` to indicate if a child of a sizer can change its size in the main orientation of the `wxBoxSizer` - where 0 stands for not changeable and a value of more than zero is interpreted relative to the value of other children of the same `wxBoxSizer`. For example, you might have a horizontal `wxBoxSizer` with three children, two of which are supposed to change their size with the sizer. Then the two stretchable windows would get a value of 1 each to make them grow and shrink equally with the sizer's horizontal dimension.

*flag*

This parameter can be used to set a number of flags which can be combined using the binary OR operator `|`. Two main behaviours are defined using these flags. One is the border around a window: the *border* parameter determines the border width

whereas the flags given here determine which side(s) of the item that the border will be added. The other flags determine how the sizer item behaves when the space allotted to the sizer changes, and is somewhat dependent on the specific kind of sizer used.

**wxTOP****wxBOTTOM****wxLEFT****wxRIGHT**

**wxALL** These flags are used to specify which side(s) of the sizer item the *border* width will apply to.

**wxEXPAND**

The item will be expanded to fill the space assigned to the item.

**wxSHAPED**

The item will be expanded as much as possible while also maintaining its aspect ratio

**wxFIXED\_MINSIZE**

Normally wxSizers will use *GetAdjustedBestSize* (p. 1809) to determine what the minimal size of window items should be, and will use that size to calculate the layout. This allows layouts to adjust when an item changes and its *best size* becomes different. If you would rather have a window item stay the size it started with then use wxFIXED\_MINSIZE.

**wxRESERVE\_SPACE\_EVEN\_IF\_HIDDEN** Normally wxSizers don't allocate space for hidden windows or other items. This flag overrides this behavior so that sufficient space is allocated for the window even if it isn't visible. This makes it possible to dynamically show and hide controls without resizing parent dialog, for example. This function is new since wxWidgets version 2.8.8

**wxALIGN\_CENTER wxALIGN\_CENTRE****wxALIGN\_LEFT****wxALIGN\_RIGHT****wxALIGN\_TOP****wxALIGN\_BOTTOM****wxALIGN\_CENTER\_VERTICAL****wxALIGN\_CENTRE\_VERTICAL****wxALIGN\_CENTER\_HORIZONTAL**

**wxALIGN\_CENTRE\_HORIZONTAL** The wxALIGN flags allow you to specify the alignment of the item within the space allotted to it by the sizer, adjusted for the border if any.

*border*

Determines the border width, if the *flag* parameter is set to include any border flag.

*userData*

Allows an extra object to be attached to the sizer item, for use in derived classes when sizing information is more complex than the *proportion* and *flag* will allow for.

*flags*

A *wxSizerFlags* (p. 1455) object that enables you to specify most of the above parameters more conveniently.

### **wxSizer::AddSpacer**

**wxSizerItem\* AddSpacer(int size)**

Adds non-stretchable space to the sizer. More readable way of calling *Add* (p. 1445)(size, size, 0).

### **wxSizer::AddStretchSpacer**

**wxSizerItem\* AddStretchSpacer(int prop = 1)**

Adds stretchable space to the sizer. More readable way of calling *Add* (p. 1445)(0, 0, prop).

### **wxSizer::CalcMin**

**wxSize CalcMin()**

This method is abstract and has to be overwritten by any derived class. Here, the sizer will do the actual calculation of its children minimal sizes.

### **wxSizer::Clear**

**void Clear(bool delete\_windows = false)**

Detaches all children from the sizer. If *delete\_windows* is `true` then child windows will also be deleted.

### **wxSizer::ComputeFittingClientSize**

**wxSize ComputeFittingClientSize(wxWindow\* window)**

Computes client area size for *window* so that it matches the sizer's minimal size. Unlike *GetMinSize* (p. 1450), this method accounts for other constraints imposed on *window*, namely display's size (returned size will never be too large for the display) and maximum window size if previously set by *wxWindow::SetMaxSize* (p. 1842).

The returned value is suitable for passing to *wxWindow::SetClientSize* (p. 1836).

This function is new since wxWidgets version 2.8.8

### **See also**

*ComputeFittingWindowSize* (p. 1449), *Fit* (p. 1449)



**wxSizer::ComputeFittingWindowSize****wxSize ComputeFittingWindowSize(wxWindow\* window)**

Like *ComputeFittingClientSize* (p. 1448), but converts the result into *window* size.

The returned value is suitable for passing to *wxWindow::SetSize* (p. 1845) or *wxWindow::SetMinSize* (p. 1842).

This function is new since wxWidgets version 2.8.8

**See also**

*ComputeFittingClientSize* (p. 1448), *Fit* (p. 1449)

**wxSizer::Detach****bool Detach(wxWindow\* window)****bool Detach(wxSizer\* sizer)****bool Detach(size\_t index)**

Detach a child from the sizer without destroying it. *window* is the window to be detached, *sizer* is the equivalent sizer and *index* is the position of the child in the sizer, typically 0 for the first item. This method does not cause any layout or resizing to take place, call *wxSizer::Layout* (p. 1452) to update the layout "on screen" after detaching a child from the sizer.

Returns true if the child item was found and detached, false otherwise.

**See also**

*wxSizer::Remove* (p. 1453)

**wxSizer::Fit****wxSize Fit(wxWindow\* window)**

Tell the sizer to resize the *window* to match the sizer's minimal size. This is commonly done in the constructor of the window itself, see sample in the description of *wxBoxSizer* (p. 149). Returns the new size.

For a top level window this is the total window size, not client size.

**See also**

*ComputeFittingClientSize* (p. 1448), *ComputeFittingWindowSize* (p. 1449)

**wxSizer::FitInside****void FitInside(wxWindow\* window)**

Tell the sizer to resize the virtual size of the *window* to match the sizer's minimal size. This will not alter the on screen size of the window, but may cause the addition/removal/alteration of scrollbars required to view the virtual area in windows which manage it.

**See also**

*wxScrolledWindow::SetScrollbars* (p. 1421), *wxSizer::SetVirtualSizeHints* (p. 1454)

**wxSizer::GetChildren****wxSizerItemList& GetChildren()**

Returns the list of the items in this sizer. The elements of type-safe *wxList* (p. 966) *wxSizerItemList* are objects of type *wxSizerItem \** (p. 1458).

**wxSizer::GetContainingWindow****wxWindow \* GetContainingWindow() const**

Returns the window this sizer is used in or `NULL` if none.

**wxSizer::GetItem****wxSizerItem \* GetItem(wxWindow\* window, bool recursive = false)****wxSizerItem \* GetItem(wxSizer\* sizer, bool recursive = false)****wxSizerItem \* GetItem(size\_t index)**

Finds item of the sizer which holds given *window*, *sizer* or is located in sizer at position *index*. Use parameter *recursive* to search in subsizers too.

Returns pointer to item or `NULL`.

**wxSizer::GetSize****wxSize GetSize()**

Returns the current size of the sizer.

**wxSizer::GetPosition****wxPoint GetPosition()**

Returns the current position of the sizer.

**wxSizer::GetMinSize****wxSize GetMinSize()**

Returns the minimal size of the sizer. This is either the combined minimal size of all the children and their borders or the minimal size set by *SetMinSize* (p. 1454), depending on which is bigger.

Note that the returned value is *client* size, not window size. In particular, if you use the value to set toplevel window's minimal or actual size, you should convert it using *wxWindow::ClientToWindowSize* (p. 1802) before passing it to *wxWindow::SetMinSize* (p. 1842) or *wxWindow::SetSize* (p. 1845).

### **wxSizer::Hide**

**bool** Hide(**wxWindow\*** window, **bool** recursive = false)

**bool** Hide(**wxSizer\*** sizer, **bool** recursive = false)

**bool** Hide(**size\_t** index)

Hides the *window*, *sizer*, or item at *index*. To make a sizer item disappear, use *Hide()* followed by *Layout()* (p. 1452). Use parameter *recursive* to hide elements found in subsizers.

Returns *true* if the child item was found, *false* otherwise.

### **See also**

*wxSizer::IsShown* (p. 1452), *wxSizer::Show* (p. 1455)

### **wxSizer::Insert**

**wxSizerItem\*** Insert(**size\_t** index, **wxWindow\*** window, **const wxSizerFlags&** flags)

**wxSizerItem\*** Insert(**size\_t** index, **wxWindow\*** window, **int** proportion = 0, **int** flag = 0, **int** border = 0, **wxObject\*** userData = NULL)

**wxSizerItem\*** Insert(**size\_t** index, **wxSizer\*** sizer, **const wxSizerFlags&** flags)

**wxSizerItem\*** Insert(**size\_t** index, **wxSizer\*** sizer, **int** proportion = 0, **int** flag = 0, **int** border = 0, **wxObject\*** userData = NULL)

**wxSizerItem\*** Insert(**size\_t** index, **int** width, **int** height, **int** proportion = 0, **int** flag = 0, **int** border = 0, **wxObject\*** userData = NULL)

Insert a child into the sizer before any existing item at *index*.

*index*

The position this child should assume in the sizer.

See *wxSizer::Add* (p. 1445) for the meaning of the other parameters.

### **wxSizer::InsertSpacer**

**wxSizerItem\*** InsertSpacer(**size\_t** index, **int** size)

Inserts non-stretchable space to the sizer. More readable way of calling *Insert* (p. 1451)(size, size, 0).

### **wxSizer::InsertStretchSpacer**

**wxSizerItem\* InsertStretchSpacer(size\_t index, int prop = 1)**

Inserts stretchable space to the sizer. More readable way of calling *Insert* (p. 1451)(0, 0, prop).

### **wxSizer::IsShown**

**bool IsShown(wxWindow\* window) const**

**bool IsShown(wxSizer\* sizer) const**

**bool IsShown(size\_t index) const**

Returns `true` if the *window*, *sizer*, or item at *index* is shown.

### **See also**

*wxSizer::Hide* (p. 1451), *wxSizer::Show* (p. 1455)

### **wxSizer::Layout**

**void Layout()**

Call this to force layout of the children anew, e.g. after having added a child to or removed a child (window, other sizer or space) from the sizer while keeping the current dimension.

### **wxSizer::Prepend**

**wxSizerItem\* Prepend(wxWindow\* window, const wxSizerFlags& flags)**

**wxSizerItem\* Prepend(wxWindow\* window, int proportion = 0, int flag = 0, int border = 0, wxObject\* userData = NULL)**

**wxSizerItem\* Prepend(wxSizer\* sizer, const wxSizerFlags& flags)**

**wxSizerItem\* Prepend(wxSizer\* sizer, int proportion = 0, int flag = 0, int border = 0, wxObject\* userData = NULL)**

**wxSizerItem\* Prepend(int width, int height, int proportion = 0, int flag = 0, int border = 0, wxObject\* userData = NULL)**

Same as *wxSizer::Add* (p. 1445), but prepends the items to the beginning of the list of items (windows, subsizers or spaces) owned by this sizer.

### **wxSizer::PrependSpacer**

**wxSizerItem\* PrependSpacer(int size)**

Prepends non-stretchable space to the sizer. More readable way of calling *Prepend* (p. 1452)(size, size, 0).

### **wxSizer::PrependStretchSpacer**

**wxSizerItem\* PrependStretchSpacer(int prop = 1)**

Prepends stretchable space to the sizer. More readable way of calling *Prepend* (p. 1452)(0, 0, prop).

### **wxSizer::RecalcSizes**

**void RecalcSizes()**

This method is abstract and has to be overwritten by any derived class. Here, the sizer will do the actual calculation of its children's positions and sizes.

### **wxSizer::Remove**

**bool Remove(wxWindow\* window)**

**bool Remove(wxSizer\* sizer)**

**bool Remove(size\_t index)**

Removes a child from the sizer and destroys it if it is a sizer or a spacer, but not if it is a window (because windows are owned by their parent window, not the sizer). *sizer* is the *wxSizer* to be removed, *index* is the position of the child in the sizer, e.g. 0 for the first item. This method does not cause any layout or resizing to take place, call *wxSizer::Layout* (p. 1452) to update the layout "on screen" after removing a child from the sizer.

**NB:** The method taking a *wxWindow\** parameter is deprecated as it does not destroy the window as would usually be expected from *Remove*. You should use *wxSizer::Detach* (p. 1449) in new code instead. There is currently no *wxSizer* method that will both detach and destroy a *wxWindow* item.

Returns true if the child item was found and removed, false otherwise.

### **wxSizer::Replace**

**bool Replace(wxWindow\* oldwin, wxWindow\* newwin, bool recursive = false)**

**bool Replace(wxSizer\* oldsz, wxSizer\* newsz, bool recursive = false)**

**bool Remove(size\_t oldindex, wxSizerItem\* newitem)**

Detaches the given *oldwin*, *oldsz* child from the sizer and replaces it with the given window, sizer, or *wxSizerItem*.

The detached child is removed **only** if it is a sizer or a spacer (because windows are owned by their parent window, not the sizer).

Use parameter *recursive* to search the given element recursively in subsizers.

This method does not cause any layout or resizing to take place, call `wxSizer::Layout` (p. 1452) to update the layout "on screen" after replacing a child from the sizer.

Returns true if the child item was found and removed, false otherwise.

### **wxSizer::SetDimension**

**void SetDimension(int x, int y, int width, int height)**

Call this to force the sizer to take the given dimension and thus force the items owned by the sizer to resize themselves according to the rules defined by the parameter in the *Add* (p. 1445) and *Prepend* (p. 1452) methods.

### **wxSizer::SetMinSize**

**void SetMinSize(int width, int height)**

**void SetMinSize(const wxSize& size)**

Call this to give the sizer a minimal size. Normally, the sizer will calculate its minimal size based purely on how much space its children need. After calling this method *GetMinSize* (p. 1450) will return either the minimal size as requested by its children or the minimal size set here, depending on which is bigger.

### **wxSizer::SetItemMinSize**

**void SetItemMinSize(wxWindow\* window, int width, int height)**

**void SetItemMinSize(wxSizer\* sizer, int width, int height)**

**void SetItemMinSize(size\_t index, int width, int height)**

Set an item's minimum size by window, sizer, or position. The item will be found recursively in the sizer's descendants. This function enables an application to set the size of an item after initial creation.

### **wxSizer::SetSizeHints**

**void SetSizeHints(wxWindow\* window)**

This method first calls *wxSizer::Fit* (p. 1449) and then *SetSizeHints* (p. 1718) on the *window* passed to it. This only makes sense when *window* is actually a *wxTopLevelWindow* (p. 1713) such as a *wxFrame* or a *wxDialog*, since *SetSizeHints* only has any effect in these classes. It does nothing in normal windows or controls.

This method is commonly invoked in the constructor of a toplevel window itself (see the sample in the description of *wxBoxSizer* (p. 149)) if the toplevel window is resizable.

### **wxSizer::SetVirtualSizeHints**

**void SetVirtualSizeHints(wxWindow\* window)**

Tell the sizer to set the minimal size of the *window* virtual area to match the sizer's minimal size. For windows with managed scrollbars this will set them appropriately.

**See also**

*wxScrolledWindow::SetScrollbars* (p. 1421)

**wxSizer::Show**

**bool Show(wxWindow\* window, bool show = true, bool recursive = false)**

**bool Show(wxSizer\* sizer, bool show = true, bool recursive = false)**

**bool Show(size\_t index, bool show = true)**

Shows or hides the *window*, *sizer*, or item at *index*. To make a sizer item disappear or reappear, use *Show()* followed by *Layout()* (p. 1452). Use parameter *recursive* to show or hide elements found in subsizers.

Returns true if the child item was found, false otherwise.

**See also**

*wxSizer::Hide* (p. 1451), *wxSizer::IsShown* (p. 1452)

## wxSizerFlags

Normally, when you add an item to a sizer via *wxSizer::Add* (p. 1445), you have to specify a lot of flags and parameters which can be unwieldy. This is where *wxSizerFlags* comes in: it allows you to specify all parameters using the named methods instead. For example, instead of

```
sizer->Add(ctrl, 0, wxEXPAND | wxBORDER, 10);
```

you can now write

```
sizer->Add(ctrl, wxSizerFlags().Expand().Border(10));
```

This is more readable and also allows you to create *wxSizerFlags* objects which can be reused for several sizer items.

```
wxSizerFlags flagsExpand(1);  
flagsExpand.Expand().Border(10);
```

```
sizer->Add(ctrl1, flagsExpand);  
sizer->Add(ctrl2, flagsExpand);
```

Note that by specification, all methods of *wxSizerFlags* return the *wxSizerFlags* object itself to allowing chaining multiple methods calls like in the examples above.

**wxSizerFlags::wxSizerFlags**

**wxSizerFlags**(int *proportion* = 0)

Creates the wxSizer with the proportion specified by *proportion*.

### **wxSizerFlags::Align**

**wxSizerFlags& Align**(int *align* = 0)

Sets the alignment of this wxSizerFlags to *align*.

Note that if this method is not called, the wxSizerFlags has no specified alignment.

### **See also**

*Left* (p. 1457),  
*Right* (p. 1457),  
*Centre* (p. 1456)

### **wxSizerFlags::Border**

**wxSizerFlags& Border**(int *direction*, int *borderinpixels*)

**wxSizerFlags& Border**(int *direction* = wxALL)

Sets the wxSizerFlags to have a border of a number of pixels specified by *borderinpixels* with the directions specified by *direction*.

In the overloaded version without *borderinpixels* parameter, the border of default size, as returned by *GetDefaultBorder* (p. 1457), is used.

### **wxSizerFlags::Center**

**wxSizerFlags& Center**()

Sets the object of the wxSizerFlags to center itself in the area it is given.

### **wxSizerFlags::Centre**

**wxSizerFlags& Centre**()

*wxSizerFlags::Center* (p. 1456) for people with the other dialect of english.

### **wxSizerFlags::DoubleBorder**

**wxSizerFlags& DoubleBorder**(int *direction* = wxALL)

Sets the border in the given *direction* having twice the default border size.

### **wxSizerFlags::DoubleHorzBorder**

**wxSizerFlags& DoubleHorzBorder**()



Sets the border in left and right directions having twice the default border size.

### **wxSizerFlags::Expand**

#### **wxSizerFlags& Expand()**

Sets the object of the wxSizerFlags to expand to fill as much area as it can.

### **wxSizerFlags::GetDefaultBorder**

#### **static int GetDefaultBorder()**

Returns the border used by default in *Border* (p. 1456) method.

### **wxSizerFlags::Left**

#### **wxSizerFlags& Left()**

Aligns the object to the left, shortcut for `Align(wxALIGN_LEFT)`

#### **See also**

*Align* (p. 1456)

### **wxSizerFlags::FixedMinSize**

#### **wxSizerFlags& FixedMinSize()**

Set the `wxFIXED_MINSIZE` flag which indicates that the initial size of the window should be also set as its minimal size.

### **wxSizerFlags::Proportion**

#### **wxSizerFlags& Proportion(int proportion = 0)**

Sets the proportion of this wxSizerFlags to *proportion*

### **wxSizerFlags::ReserveSpaceEvenIfHidden**

#### **wxSizerFlags& ReserveSpaceEvenIfHidden()**

Set the `wxRESERVE_SPACE_EVEN_IF_HIDDEN` flag. Normally wxSizers don't allocate space for hidden windows or other items. This flag overrides this behavior so that sufficient space is allocated for the window even if it isn't visible. This makes it possible to dynamically show and hide controls without resizing parent dialog, for example. This function is new since wxWidgets version 2.8.8

### **wxSizerFlags::Right**

#### **wxSizerFlags& Right()**

Aligns the object to the right, shortcut for `Align(wxALIGN_RIGHT)`

**See also**

*Align* (p. 1456)

**wxSizerFlags::Shaped****wxSizerFlags& Shaped()**

Set the `wx_SHAPED` flag which indicates that the elements should always keep the fixed width to height ratio equal to its original value.

**wxSizerFlags::TripleBorder****wxSizerFlags& TripleBorder(int direction = wxALL)**

Sets the border in the given *direction* having thrice the default border size.

**wxSizerItem**

The `wxSizerItem` class is used to track the position, size and other attributes of each item managed by a `wxSizer` (p. 1444). It is not usually necessary to use this class because the sizer elements can also be identified by their positions or window or sizer pointers but sometimes it may be more convenient to use it directly.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/sizer.h>

**wxSizerItem::wxSizerItem**

**wxSizerItem(int width, int height, int proportion, int flag, int border, wxObject\* userData)**

Construct a sizer item for tracking a spacer.

**wxSizerItem(wxWindow\* window, const wxSizerFlags& flags)**

**wxSizerItem(wxWindow\* window, int proportion, int flag, int border, wxObject\* userData)**

Construct a sizer item for tracking a window.

**wxSizerItem(wxSizer\* window, const wxSizerFlags& flags)**

**wxSizerItem(wxSizer\* sizer, int proportion, int flag, int border, wxObject\* userData)**

Construct a sizer item for tracking a subsizer.

**wxSizerItem::~~wxSizerItem**

**~wxSizerItem()**

Deletes the user data and subsizer, if any.

**wxSizerItem::CalcMin**

**wxSize CalcMin()**

Calculates the minimum desired size for the item, including any space needed by borders.

**wxSizerItem::DeleteWindows**

**void DeleteWindows()**

Destroy the window or the windows in a subsizer, depending on the type of item.

**wxSizerItem::DetachSizer**

**void DetachSizer()**

Enable deleting the SizerItem without destroying the contained sizer.

**wxSizerItem::GetBorder**

**int GetBorder() const**

Return the border attribute.

**wxSizerItem::GetFlag**

**int GetFlag() const**

Return the flags attribute.

**wxSizerItem::GetMinSize**

**wxSize GetMinSize() const**

Get the minimum size needed for the item.

**wxSizerItem::GetPosition**

**wxPoint GetPosition() const**

What is the current position of the item, as set in the last Layout.

**wxSizerItem::GetProportion****int GetProportion() const**

Get the proportion item attribute.

**wxSizerItem::GetRatio****float GetRatio() const**

Get the ration item attribute.

**wxSizerItem::GetRect****wxRect GetRect()**

Get the rectangle of the item on the parent window, excluding borders.

**wxSizerItem::GetSize****wxSize GetSize() const**

Get the current size of the item, as set in the last Layout.

**wxSizerItem::GetSizer****wxSizer\* GetSizer() const**

If this item is tracking a sizer, return it. NULL otherwise.

**wxSizerItem::GetSpacer****const wxSize& GetSpacer() const**

If this item is tracking a spacer, return its size.

**wxSizerItem::GetUserData****wxObject\* GetUserData() const**

Get the userData item attribute.

**wxSizerItem::GetWindow****wxWindow\* GetWindow() const**

If this item is tracking a window then return it. NULL otherwise.

**wxSizerItem::IsSizer****bool IsSizer() const**

Is this item a sizer?

**wxSizerItem::IsShown****bool IsShown() const**

Returns `true` if this item is a window or a spacer and it is shown or if this item is a sizer and not all its elements are hidden. In other words, for sizer items, all of the child elements must be hidden for the sizer itself to be considered hidden.

**wxSizerItem::IsSpacer****bool IsSpacer() const**

Is this item a spacer?

**wxSizerItem::IsWindow****bool IsWindow() const**

Is this item a window?

**wxSizerItem::SetBorder****void SetBorder(int border)**

Set the border item attribute.

**wxSizerItem::SetDimension****void SetDimension(const wxPoint& pos, const wxSize& size)**

Set the position and size of the space allocated to the sizer, and adjust the position and size of the item to be within that space taking alignment and borders into account.

**wxSizerItem::SetFlag****void SetFlag(int flag)**

Set the flag item attribute.

**wxSizerItem::SetInitSize****void SetInitSize(int x, int y)****wxSizerItem::SetProportion**

**void SetProportion(int *proportion*)**

Set the proportion item attribute.

**wxSizerItem::SetRatio**

**void SetRatio(int *width*, int *height*)**

**void SetRatio(wxSize *size*)**

**void SetRatio(float *ratio*)**

Set the ratio item attribute.

**wxSizerItem::SetSizer**

**void SetSizer(wxSizer\* *sizer*)**

Set the sizer tracked by this item.

**wxSizerItem::SetSpacer**

**void SetSpacer(const wxSize& *size*)**

Set the size of the spacer tracked by this item.

**wxSizerItem::SetWindow**

**void SetWindow(wxWindow\* *window*)**

Set the window to be tracked by this item.

**wxSizerItem::Show**

**void Show(bool *show*)**

Set the show item attribute, which sizers use to determine if the item is to be made part of the layout or not. If the item is tracking a window then it is shown or hidden as needed.

## **wxSlider**

A slider is a control with a handle which can be pulled back and forth to change the value.

On Windows, the track bar control is used.

Slider events are handled in the same way as a scrollbar.

**Derived from**

*wxControl* (p. 285)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/slider.h>

### Window styles

<b>wxSL_HORIZONTAL</b>	Displays the slider horizontally (this is the default).
<b>wxSL_VERTICAL</b>	Displays the slider vertically.
<b>wxSL_AUTOTICKS</b>	Displays tick marks.
<b>wxSL_LABELS</b>	Displays minimum, maximum and value labels.
<b>wxSL_LEFT</b>	Displays ticks on the left and forces the slider to be vertical.
<b>wxSL_RIGHT</b>	Displays ticks on the right and forces the slider to be vertical.
<b>wxSL_TOP</b>	Displays ticks on the top.
<b>wxSL_BOTTOM</b>	Displays ticks on the bottom (this is the default).
<b>wxSL_SELRANGE</b>	Allows the user to select a range on the slider. Windows only.
<b>wxSL_INVERSE</b>	Inverses the minimum and maximum endpoints on the slider. Not compatible with <b>wxSL_SELRANGE</b> .

See also *window styles overview* (p. 2089).

### Event table macros

To process a scroll event, use these event handler macros to direct input to member functions that take a `wxScrollEvent` argument. You can use `EVT_COMMAND_SCROLL` . . . macros with window IDs for when intercepting scroll events from controls, or `EVT_SCROLL` . . . macros without window IDs for intercepting scroll events from the receiving window -- except for this, the macros behave exactly the same.

<b>EVT_SCROLL(func)</b>	Process all scroll events.
<b>EVT_SCROLL_TOP(func)</b>	Process <code>wxEVT_SCROLL_TOP</code> scroll-to-top events (minimum position).
<b>EVT_SCROLL_BOTTOM(func)</b>	Process <code>wxEVT_SCROLL_BOTTOM</code> scroll-to-bottom events (maximum position).
<b>EVT_SCROLL_LINEUP(func)</b>	Process <code>wxEVT_SCROLL_LINEUP</code> line up events.

<b>EVT_SCROLL_LINEDOWN(func)</b>	Process wxEVT_SCROLL_LINEDOWN line down events.
<b>EVT_SCROLL_PAGEUP(func)</b>	Process wxEVT_SCROLL_PAGEUP page up events.
<b>EVT_SCROLL_PAGEDOWN(func)</b>	Process wxEVT_SCROLL_PAGEDOWN page down events.
<b>EVT_SCROLL_THUMBTRACK(func)</b>	Process wxEVT_SCROLL_THUMBTRACK thumbtrack events (frequent events sent as the user drags the thumbtrack).
<b>EVT_SCROLL_THUMBRELEASE(func)</b>	Process wxEVT_SCROLL_THUMBRELEAS E thumb release events.
<b>EVT_SCROLL_CHANGED(func)</b>	Process wxEVT_SCROLL_CHANGED end of scrolling events (MSW only).
<b>EVT_COMMAND_SCROLL(id, func)</b>	Process all scroll events.
<b>EVT_COMMAND_SCROLL_TOP(id, func)</b>	Process wxEVT_SCROLL_TOP scroll-to-top events (minimum position).
<b>EVT_COMMAND_SCROLL_BOTTOM(id, func)</b>	Process wxEVT_SCROLL_BOTTOM scroll-to-bottom events (maximum position).
<b>EVT_COMMAND_SCROLL_LINEUP(id, func)</b>	Process wxEVT_SCROLL_LINEUP line up events.
<b>EVT_COMMAND_SCROLL_LINEDOWN(id, func)</b>	Process wxEVT_SCROLL_LINEDOWN line down events.
<b>EVT_COMMAND_SCROLL_PAGEUP(id, func)</b>	Process wxEVT_SCROLL_PAGEUP page up events.
<b>EVT_COMMAND_SCROLL_PAGEDOWN(id, func)</b>	Process wxEVT_SCROLL_PAGEDOWN page down events.
<b>EVT_COMMAND_SCROLL_THUMBTRACK(id, func)</b>	Process



`wxEVT_SCROLL_THUMBTRACK`  
thumbtrack events (frequent events sent as the user drags the thumbtrack).

**`EVT_COMMAND_SCROLL_THUMBRELEASE(func)`**      Process  
`wxEVT_SCROLL_THUMBRELEASE`  
thumb release events.

**`EVT_COMMAND_SCROLL_CHANGED(func)`**      Process  
`wxEVT_SCROLL_CHANGED` end  
of scrolling events (MSW only).

#### **The difference between `EVT_SCROLL_THUMBRELEASE` and `EVT_SCROLL_CHANGED`**

The `EVT_SCROLL_THUMBRELEASE` event is only emitted when actually dragging the thumb using the mouse and releasing it (This `EVT_SCROLL_THUMBRELEASE` event is also followed by an `EVT_SCROLL_CHANGED` event).

The `EVT_SCROLL_CHANGED` event also occurs when using the keyboard to change the thumb position, and when clicking next to the thumb (In all these cases the `EVT_SCROLL_THUMBRELEASE` event does not happen).

In short, the `EVT_SCROLL_CHANGED` event is triggered when scrolling/ moving has finished independently of the way it had started. Please see the widgets sample ("Slider" page) to see the difference between `EVT_SCROLL_THUMBRELEASE` and `EVT_SCROLL_CHANGED` in action.

#### **See also**

*Event handling overview* (p. 2077), *wxScrollBar* (p. 1408)

### **wxSlider::wxSlider**

**`wxSlider()`**

Default slider.

**`wxSlider(wxWindow* parent, wxWindowID id, int value, int minValue, int maxValue, const wxPoint& point = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxSL_HORIZONTAL, const wxValidator& validator = wxDefaultValidator, const wxString& name = "slider")`**

Constructor, creating and showing a slider.

#### **Parameters**

*parent*

Parent window. Must not be NULL.

*id*

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

*value*

Initial position for the slider.

*minValue*

Minimum slider position.

*maxValue*

Maximum slider position.

*size*

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

*style*

Window style. See *wxSlider* (p. 1462).

*validator*

Window validator.

*name*

Window name.

**See also**

*wxSlider::Create* (p. 1467), *wxValidator* (p. 1767)

**wxSlider::~~wxSlider****void ~wxSlider()**

Destructor, destroying the slider.

**wxSlider::ClearSel****void ClearSel()**

Clears the selection, for a slider with the **wxSL\_SELRange** style.

**Remarks**

Windows 95 only.

**wxSlider::ClearTicks**

**void ClearTicks()**

Clears the ticks.

**Remarks**

Windows 95 only.

**wxSlider::Create**

**bool Create**(wxWindow\* *parent*, wxWindowID *id*, int *value*, int *minValue*, int *maxValue*, const wxPoint& *point* = wxDefaultPosition, const wxSize& *size* = wxDefaultSize, long *style* = wxSL\_HORIZONTAL, const wxValidator& *validator* = wxDefaultValidator, const wxString& *name* = "slider")

Used for two-step slider construction. See *wxSlider::wxSlider* (p. 1465) for further details.

**wxSlider::GetLineSize**

**int GetLineSize() const**

Returns the line size.

**See also**

*wxSlider::SetLineSize* (p. 1469)

**wxSlider::GetMax**

**int GetMax() const**

Gets the maximum slider value.

**See also**

*wxSlider::GetMin* (p. 1467), *wxSlider::SetRange* (p. 1469)

**wxSlider::GetMin**

**int GetMin() const**

Gets the minimum slider value.

**See also**

*wxSlider::GetMin* (p. 1467), *wxSlider::SetRange* (p. 1469)

**wxSlider::GetPageSize**

**int GetPageSize() const**

Returns the page size.

**See also**

*wxSlider::SetPageSize* (p. 1469)

**wxSlider::GetSelEnd**

**int GetSelEnd() const**

Returns the selection end point.

**Remarks**

Windows 95 only.

**See also**

*wxSlider::GetSelStart* (p. 1468), *wxSlider::SetSelection* (p. 1470)

**wxSlider::GetSelStart**

**int GetSelStart() const**

Returns the selection start point.

**Remarks**

Windows 95 only.

**See also**

*wxSlider::GetSelEnd* (p. 1468), *wxSlider::SetSelection* (p. 1470)

**wxSlider::GetThumbLength**

**int GetThumbLength() const**

Returns the thumb length.

**Remarks**

Windows 95 only.

**See also**

*wxSlider::SetThumbLength* (p. 1470)

**wxSlider::GetTickFreq**

**int GetTickFreq() const**

Returns the tick frequency.

**Remarks**

Windows 95 only.

**See also**

*wxSlider::SetTickFreq* (p. 1471)

**wxSlider::GetValue**

**int GetValue() const**

Gets the current slider value.

**See also**

*wxSlider::GetMin* (p. 1467), *wxSlider::GetMax* (p. 1467), *wxSlider::SetValue* (p. 1471)

**wxSlider::SetLineSize**

**void SetLineSize(int *lineSize*)**

Sets the line size for the slider.

**Parameters**

*lineSize*

The number of steps the slider moves when the user moves it up or down a line.

**See also**

*wxSlider::GetLineSize* (p. 1467)

**wxSlider::SetPageSize**

**void SetPageSize(int *pageSize*)**

Sets the page size for the slider.

**Parameters**

*pageSize*

The number of steps the slider moves when the user pages up or down.

**See also**

*wxSlider::GetPageSize* (p. 1467)

**wxSlider::SetRange**

**void SetRange(int *minValue*, int *maxValue*)**

Sets the minimum and maximum slider values.

**See also**

*wxSlider::GetMin* (p. 1467), *wxSlider::GetMax* (p. 1467)

**wxSlider::SetSelection**

**void SetSelection(int startPos, int endPos)**

Sets the selection.

**Parameters**

*startPos*

The selection start position.

*endPos*

The selection end position.

**Remarks**

Windows 95 only.

**See also**

*wxSlider::GetSelStart* (p. 1468), *wxSlider::GetSelEnd* (p. 1468)

**wxSlider::SetThumbLength**

**void SetThumbLength(int len)**

Sets the slider thumb length.

**Parameters**

*len*

The thumb length.

**Remarks**

Windows 95 only.

**See also**

*wxSlider::GetThumbLength* (p. 1468)

**wxSlider::SetTick**

**void SetTick(int tickPos)**

Sets a tick position.

**Parameters***tickPos*

The tick position.

**Remarks**

Windows 95 only.

**See also**

*wxSlider::SetTickFreq* (p. 1471)

**wxSlider::SetTickFreq****void SetTickFreq(int *n*, int *pos*)**

Sets the tick mark frequency and position.

**Parameters***n*

Frequency. For example, if the frequency is set to two, a tick mark is displayed for every other increment in the slider's range.

*pos*

Position. Must be greater than zero. TODO: what is this for?

**Remarks**

Windows 95 only.

**See also**

*wxSlider::GetTickFreq* (p. 1468)

**wxSlider::SetValue****void SetValue(int *value*)**

Sets the slider position.

**Parameters***value*

The slider position.

**See also**

*wxSlider::GetValue* (p. 1469)

## **wxSocketAddress**

You are unlikely to need to use this class: only `wxSocketBase` uses it.

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

`<wx/socket.h>`

### **See also**

*wxSocketBase* (p. 1472) *wxIPAddress* (p. 944) *wxIPv4address* (p. 946)

## **wxSocketAddress::wxSocketAddress**

**wxSocketAddress()**

Default constructor.

## **wxSocketAddress::~~wxSocketAddress**

**~wxSocketAddress()**

Default destructor.

## **wxSocketAddress::Clear**

**void Clear()**

Delete all informations about the address.

## **wxSocketBase**

`wxSocketBase` is the base class for all socket-related objects, and it defines all basic IO functionality.

Note: (Workaround for implementation limitation for `wxWidgets` up to 2.5.x) If you want to use sockets or derived classes such as `wxFTP` in a secondary thread, call `wxSocketBase::Initialize()` (undocumented) from the main thread before creating any sockets - in `wxApp::OnInit` for example. See <http://wiki.wxwidgets.org/wiki.pl?WxSocket> or <http://www.litwindow.com/knowhow/knowhow.html> for more details.

### **Derived from**

*wxObject* (p. 1148)



**Include files**

<wx/socket.h>

**wxSocket errors**

<b>wxSOCKET_NOERROR</b>	No error happened.
<b>wxSOCKET_INVOP</b>	Invalid operation.
<b>wxSOCKET_IOERR</b>	Input/Output error.
<b>wxSOCKET_INVADDR</b>	Invalid address passed to wxSocket.
<b>wxSOCKET_INVSOCK</b>	Invalid socket (uninitialized).
<b>wxSOCKET_NOHOST</b>	No corresponding host.
<b>wxSOCKET_INVPORT</b>	Invalid port.
<b>wxSOCKET_WOULDBLOCK</b>	The socket is non-blocking and the operation would block.
<b>wxSOCKET_TIMEOUT</b>	The timeout for this operation expired.
<b>wxSOCKET_MEMERR</b>	Memory exhausted.
<b>wxSocket events</b>	
<b>wxSOCKET_INPUT</b>	There is data available for reading.
<b>wxSOCKET_OUTPUT</b>	The socket is ready to be written to.
<b>wxSOCKET_CONNECTION</b>	Incoming connection request (server), or successful connection establishment (client).
<b>wxSOCKET_LOST</b>	The connection has been closed.

A brief note on how to use these events:

The **wxSOCKET\_INPUT** event will be issued whenever there is data available for reading. This will be the case if the input queue was empty and new data arrives, or if the application has read some data yet there is still more data available. This means that the application does not need to read all available data in response to a **wxSOCKET\_INPUT** event, as more events will be produced as necessary.

The **wxSOCKET\_OUTPUT** event is issued when a socket is first connected with *Connect* (p. 1489) or accepted with *Accept* (p. 1493). After that, new events will be generated only after an output operation fails with **wxSOCKET\_WOULDBLOCK** and buffer space becomes available again. This means that the application should assume that it can write data to the socket until an **wxSOCKET\_WOULDBLOCK** error occurs; after this, whenever the socket becomes writable again the application will be notified with another **wxSOCKET\_OUTPUT** event.

The **wxSOCKET\_CONNECTION** event is issued when a delayed connection request

completes successfully (client) or when a new connection arrives at the incoming queue (server).

The **wxSOCKET\_LOST** event is issued when a close indication is received for the socket. This means that the connection broke down or that it was closed by the peer. Also, this event will be issued if a connection request fails.

### Event handling

To process events coming from a socket object, use the following event handler macro to direct events to member functions that take a *wxSocketEvent* (p. 1490) argument.

**EVT\_SOCKET(id, func)**                      Process a wxEVT\_SOCKET event.

### See also

*wxSocketEvent* (p. 1490), *wxSocketClient* (p. 1488), *wxSocketServer* (p. 1492), *Sockets sample* (p. 2037)

## Construction and destruction

*wxSocketBase* (p. 1475)  
*~wxSocketBase* (p. 1475)  
*Destroy* (p. 1476)

## Socket state

Functions to retrieve current state and miscellaneous info.

*Error* (p. 1476)  
*GetLocal* (p. 1477)  
*GetPeer* (p. 1477) *IsConnected* (p. 1477)  
*IsData* (p. 1478)  
*IsDisconnected* (p. 1478)  
*LastCount* (p. 1478)  
*LastError* (p. 1478)  
*IsOk* (p. 1478)  
*SaveState* (p. 1479)  
*RestoreState* (p. 1479)

## Basic IO

Functions that perform basic IO functionality.

*Close* (p. 1475)  
*Discard* (p. 1476)  
*Peek* (p. 1482)  
*Read* (p. 1482)  
*ReadMsg* (p. 1483)  
*Unread* (p. 1484)

*Write* (p. 1487)  
*WriteMsg* (p. 1487)

Functions that perform a timed wait on a certain IO condition.

*InterruptWait* (p. 1477)  
*Wait* (p. 1484)  
*WaitForLost* (p. 1485)  
*WaitForRead* (p. 1486)  
*WaitForWrite* (p. 1486)

and also:

*wxSocketServer::WaitForAccept* (p. 1494)  
*wxSocketClient::WaitOnConnect* (p. 1489)

Functions that allow applications to customize socket IO as needed.

*GetFlags* (p. 1477)  
*SetFlags* (p. 1480)  
*SetTimeout* (p. 1482)  
*SetLocal* (p. 1481)

## Handling socket events

Functions that allow applications to receive socket events.

*Notify* (p. 1478)  
*SetNotify* (p. 1481)  
*GetClientData* (p. 1476)  
*SetClientData* (p. 1479)  
*SetEventHandler* (p. 1479)

## **wxSocketBase::wxSocketBase**

**wxSocketBase()**

Default constructor. Don't use it directly; instead, use *wxSocketClient* (p. 1488) to construct a socket client, or *wxSocketServer* (p. 1492) to construct a socket server.

## **wxSocketBase::~~wxSocketBase**

**~wxSocketBase()**

Destructor. Do not destroy a socket using the delete operator directly; use *Destroy* (p. 1476) instead. Also, do not create socket objects in the stack.

## **wxSocketBase::Close**

**void Close()**

This function shuts down the socket, disabling further transmission and reception of data; it also disables events for the socket and frees the associated system resources. Upon socket destruction, `Close` is automatically called, so in most cases you won't need to do it yourself, unless you explicitly want to shut down the socket, typically to notify the peer that you are closing the connection.

**Remark/Warning**

Although `Close` immediately disables events for the socket, it is possible that event messages may be waiting in the application's event queue. The application must therefore be prepared to handle socket event messages even after calling `Close`.

**wxSocketBase::Destroy****bool Destroy()**

Destroys the socket safely. Use this function instead of the delete operator, since otherwise socket events could reach the application even after the socket has been destroyed. To prevent this problem, this function appends the `wxSocket` to a list of object to be deleted on idle time, after all events have been processed. For the same reason, you should avoid creating socket objects in the stack.

`Destroy` calls `Close` (p. 1475) automatically.

**Return value**

Always true.

**wxSocketBase::Discard****wxSocketBase& Discard()**

This function simply deletes all bytes in the incoming queue. This function always returns immediately and its operation is not affected by IO flags.

Use `LastCount` (p. 1478) to verify the number of bytes actually discarded.

If you use `Error` (p. 1476), it will always return false.

**wxSocketBase::Error****bool Error() const**

Returns true if an error occurred in the last IO operation.

Use this function to check for an error condition after one of the following calls: `Discard`, `Peek`, `Read`, `ReadMsg`, `Unread`, `Write`, `WriteMsg`.

**wxSocketBase::GetClientData**

**void \* GetClientData() const**

Returns a pointer of the client data for this socket, as set with *SetClientData* (p. 1479)

**wxSocketBase::GetLocal**

**bool GetLocal(wxSockAddress& addr) const**

This function returns the local address field of the socket. The local address field contains the complete local address of the socket (local address, local port, ...).

**Return value**

true if no error happened, false otherwise.

**wxSocketBase::GetFlags**

**wxSocketFlags GetFlags() const**

Returns current IO flags, as set with *SetFlags* (p. 1480)

**wxSocketBase::GetPeer**

**bool GetPeer(wxSockAddress& addr) const**

This function returns the peer address field of the socket. The peer address field contains the complete peer host address of the socket (address, port, ...).

**Return value**

true if no error happened, false otherwise.

**wxSocketBase::InterruptWait**

**void InterruptWait()**

Use this function to interrupt any wait operation currently in progress. Note that this is not intended as a regular way to interrupt a *Wait* call, but only as an escape mechanism for exceptional situations where it is absolutely necessary to use it, for example to abort an operation due to some exception or abnormal problem. *InterruptWait* is automatically called when you *Close* (p. 1475) a socket (and thus also upon socket destruction), so you don't need to use it in these cases.

*wxSocketBase::Wait* (p. 1484), *wxSocketServer::WaitForAccept* (p. 1494),  
*wxSocketBase::WaitForLost* (p. 1485), *wxSocketBase::WaitForRead* (p. 1486),  
*wxSocketBase::WaitForWrite* (p. 1486), *wxSocketClient::WaitOnConnect* (p. 1489)

**wxSocketBase::IsConnected**

**bool IsConnected() const**

Returns true if the socket is connected.

**wxSocketBase::IsData****bool IsData() const**

This function waits until the socket is readable. This might mean that queued data is available for reading or, for streamed sockets, that the connection has been closed, so that a read operation will complete immediately without blocking (unless the **wxSOCKET\_WAITALL** flag is set, in which case the operation might still block).

**wxSocketBase::IsDisconnected****bool IsDisconnected() const**

Returns true if the socket is not connected.

**wxSocketBase::LastCount****wxUInt32 LastCount() const**

Returns the number of bytes read or written by the last IO call.

Use this function to get the number of bytes actually transferred after using one of the following IO calls: Discard, Peek, Read, ReadMsg, Unread, Write, WriteMsg.

**wxSocketBase::LastError****wxSocketError LastError() const**

Returns the last wxSocket error. See *wxSocket errors* (p. 1472).

Please note that this function merely returns the last error code, but it should not be used to determine if an error has occurred (this is because successful operations do not change the LastError value). Use *Error* (p. 1476) first, in order to determine if the last IO call failed. If this returns true, use LastError to discover the cause of the error.

**wxSocketBase::Notify****void Notify(bool notify)**

According to the *notify* value, this function enables or disables socket events. If *notify* is true, the events configured with *SetNotify* (p. 1481) will be sent to the application. If *notify* is false; no events will be sent.

**wxSocketBase::IsOk****bool IsOk() const**

Returns true if the socket is initialized and ready and false in other cases.

**Remark/Warning**

For *wxSocketClient* (p. 1488), *Ok* won't return true unless the client is connected to a server.

For *wxSocketServer* (p. 1492), *Ok* will return true if the server could bind to the specified address and is already listening for new connections.

*Ok* does not check for IO errors; use *Error* (p. 1476) instead for that purpose.

### **wxSocketBase::RestoreState**

#### **void RestoreState()**

This function restores the previous state of the socket, as saved with *SaveState* (p. 1479)

Calls to *SaveState* and *RestoreState* can be nested.

#### **See also**

*wxSocketBase::SaveState* (p. 1479)

### **wxSocketBase::SaveState**

#### **void SaveState()**

This function saves the current state of the socket in a stack. Socket state includes flags, as set with *SetFlags* (p. 1480), event mask, as set with *SetNotify* (p. 1481) and *Notify* (p. 1478), user data, as set with *SetClientData* (p. 1479).

Calls to *SaveState* and *RestoreState* can be nested.

#### **See also**

*wxSocketBase::RestoreState* (p. 1479)

### **wxSocketBase::SetClientData**

#### **void SetClientData(void \*data)**

Sets user-supplied client data for this socket. All socket events will contain a pointer to this data, which can be retrieved with the *wxSocketEvent::GetClientData* (p. 1491) function.

### **wxSocketBase::SetEventHandler**

#### **void SetEventHandler(wxEvtHandler& handler, int id = -1)**

Sets an event handler to be called when a socket event occurs. The handler will be called for those events for which notification is enabled with *SetNotify* (p. 1481) and *Notify* (p. 1478).

#### **Parameters**

*handler*

Specifies the event handler you want to use.

*id*

The id of socket event.

### See also

*wxSocketBase::SetNotify* (p. 1481), *wxSocketBase::Notify* (p. 1478), *wxSocketEvent* (p. 1490), *wxEvtHandler* (p. 576)

## wxSocketBase::SetFlags

**void SetFlags(wxSocketFlags flags)**

Use SetFlags to customize IO operation for this socket. The *flags* parameter may be a combination of flags ORed together. The following flags can be used:

<b>wxSOCKET_NONE</b>	Normal functionality.
<b>wxSOCKET_NOWAIT</b>	Read/write as much data as possible and return immediately.
<b>wxSOCKET_WAITALL</b>	Wait for all required data to be read/written unless an error occurs.
<b>wxSOCKET_BLOCK</b>	Block the GUI (do not yield) while reading/writing data.
<b>wxSOCKET_REUSEADDR</b>	Allows the use of an in-use port (wxServerSocket only)

A brief overview on how to use these flags follows.

If no flag is specified (this is the same as **wxSOCKET\_NONE**), IO calls will return after some data has been read or written, even when the transfer might not be complete. This is the same as issuing exactly one blocking low-level call to *recv()* or *send()*. Note that *blocking* here refers to when the function returns, not to whether the GUI blocks during this time.

If **wxSOCKET\_NOWAIT** is specified, IO calls will return immediately. Read operations will retrieve only available data. Write operations will write as much data as possible, depending on how much space is available in the output buffer. This is the same as issuing exactly one nonblocking low-level call to *recv()* or *send()*. Note that *nonblocking* here refers to when the function returns, not to whether the GUI blocks during this time.

If **wxSOCKET\_WAITALL** is specified, IO calls won't return until ALL the data has been read or written (or until an error occurs), blocking if necessary, and issuing several low level calls if necessary. This is the same as having a loop which makes as many blocking low-level calls to *recv()* or *send()* as needed so as to transfer all the data. Note that *blocking* here refers to when the function returns, not to whether the GUI blocks during this time.

The **wxSOCKET\_BLOCK** flag controls whether the GUI blocks during IO operations. If



this flag is specified, the socket will not yield during IO calls, so the GUI will remain blocked until the operation completes. If it is not used, then the application must take extra care to avoid unwanted reentrance.

The **wxSOCKET\_REUSEADDR** flag controls the use of the `SO_REUSEADDR` standard `setsockopt()` flag. This flag allows the socket to bind to a port that is already in use. This is mostly used on UNIX-based systems to allow rapid starting and stopping of a server - otherwise you may have to wait several minutes for the port to become available. **wxSOCKET\_REUSEADDR** can also be used with socket clients to (re)bind to a particular local port for an outgoing connection. This option can have surprising platform dependent behavior, so check the documentation for your platform's implementation of `setsockopt()`. Note that on BSD-based systems (e.g. Mac OS X), use of **wxSOCKET\_REUSEADDR** implies `SO_REUSEPORT` in addition to `SO_REUSEADDR` to be consistent with Windows.

So:

**wxSOCKET\_NONE** will try to read at least SOME data, no matter how much.

**wxSOCKET\_NOWAIT** will always return immediately, even if it cannot read or write ANY data.

**wxSOCKET\_WAITALL** will only return when it has read or written ALL the data.

**wxSOCKET\_BLOCK** has nothing to do with the previous flags and it controls whether the GUI blocks.

**wxSOCKET\_REUSEADDR** controls special platform-specific behavior for reusing local addresses/ports.

### **wxSocketBase::SetLocal**

**bool SetLocal(wxIPV4address& local)**

This function allows you to set the local address and port, useful when an application needs to reuse a particular port. When a local port is set for a *wxSocketClient* (p. 1488), **bind** will be called before **connect**.

### **wxSocketBase::SetNotify**

**void SetNotify(wxSocketEventFlags flags)**

**SetNotify** specifies which socket events are to be sent to the event handler. The *flags* parameter may be combination of flags ORed together. The following flags can be used:

<b>wxSOCKET_INPUT_FLAG</b>	to receive <b>wxSOCKET_INPUT</b>
<b>wxSOCKET_OUTPUT_FLAG</b>	to receive <b>wxSOCKET_OUTPUT</b>
<b>wxSOCKET_CONNECTION_FLAG</b>	to receive <b>wxSOCKET_CONNECTION</b>
<b>wxSOCKET_LOST_FLAG</b>	to receive <b>wxSOCKET_LOST</b>

For example:

```
sock.SetNotify(wxSOCKET_INPUT_FLAG | wxSOCKET_LOST_FLAG);  
sock.Notify(true);
```

In this example, the user will be notified about incoming socket data and whenever the connection is closed.

For more information on socket events see *wxSocket events* (p. 1472).

### **wxSocketBase::SetTimeout**

**void SetTimeout(int seconds)**

This function sets the default socket timeout in seconds. This timeout applies to all IO calls, and also to the *Wait* (p. 1484) family of functions if you don't specify a wait interval. Initially, the default timeout is 10 minutes.

### **wxSocketBase::Peek**

**wxSocketBase& Peek(void \* buffer, wxUint32 nbytes)**

This function peeks a buffer of *nbytes* bytes from the socket. Peeking a buffer doesn't delete it from the socket input queue.

Use *LastCount* (p. 1478) to verify the number of bytes actually peeked.

Use *Error* (p. 1476) to determine if the operation succeeded.

#### **Parameters**

*buffer*

Buffer where to put peeked data.

*nbytes*

Number of bytes.

#### **Return value**

Returns a reference to the current object.

#### **Remark/Warning**

The exact behaviour of *wxSocketBase::Peek* depends on the combination of flags being used. For a detailed explanation, see *wxSocketBase::SetFlags* (p. 1480)

#### **See also**

*wxSocketBase::Error* (p. 1476), *wxSocketBase::LastError* (p. 1478),  
*wxSocketBase::LastCount* (p. 1478), *wxSocketBase::SetFlags* (p. 1480)

### **wxSocketBase::Read**

**wxSocketBase& Read(void \* buffer, wxUint32 nbytes)**

This function reads a buffer of *nbytes* bytes from the socket.

Use *LastCount* (p. 1478) to verify the number of bytes actually read.

Use *Error* (p. 1476) to determine if the operation succeeded.

**Parameters**

*buffer*

Buffer where to put read data.

*nbytes*

Number of bytes.

**Return value**

Returns a reference to the current object.

**Remark/Warning**

The exact behaviour of `wxSocketBase::Read` depends on the combination of flags being used. For a detailed explanation, see `wxSocketBase::SetFlags` (p. 1480).

**See also**

`wxSocketBase::Error` (p. 1476), `wxSocketBase::LastError` (p. 1478),  
`wxSocketBase::LastCount` (p. 1478), `wxSocketBase::SetFlags` (p. 1480)

**wxSocketBase::ReadMsg****wxSocketBase& ReadMsg(void \* buffer, wxUint32 nbytes)**

This function reads a buffer sent by *WriteMsg* (p. 1487) on a socket. If the buffer passed to the function isn't big enough, the remaining bytes will be discarded. This function always waits for the buffer to be entirely filled, unless an error occurs.

Use *LastCount* (p. 1478) to verify the number of bytes actually read.

Use *Error* (p. 1476) to determine if the operation succeeded.

**Parameters**

*buffer*

Buffer where to put read data.

*nbytes*

Size of the buffer.

**Return value**

Returns a reference to the current object.

### Remark/Warning

`wxSocketBase::ReadMsg` will behave as if the `wxSOCKET_WAITALL` flag was always set and it will always ignore the `wxSOCKET_NOWAIT` flag. The exact behaviour of `ReadMsg` depends on the `wxSOCKET_BLOCK` flag. For a detailed explanation, see `wxSocketBase::SetFlags` (p. 1480).

### See also

`wxSocketBase::Error` (p. 1476), `wxSocketBase::LastError` (p. 1478),  
`wxSocketBase::LastCount` (p. 1478), `wxSocketBase::SetFlags` (p. 1480),  
`wxSocketBase::WriteMsg` (p. 1487)

## wxSocketBase::Unread

**wxSocketBase& Unread(const void \* *buffer*, wxUint32 *nbytes*)**

This function unread a buffer. That is, the data in the buffer is put back in the incoming queue. This function is not affected by `wxSocket` flags.

If you use `LastCount` (p. 1478), it will always return *nbytes*.

If you use `Error` (p. 1476), it will always return false.

### Parameters

*buffer*

Buffer to be unread.

*nbytes*

Number of bytes.

### Return value

Returns a reference to the current object.

### See also

`wxSocketBase::Error` (p. 1476), `wxSocketBase::LastCount` (p. 1478),  
`wxSocketBase::LastError` (p. 1478)

## wxSocketBase::Wait

**bool Wait(long *seconds* = -1, long *millisecond* = 0)**

This function waits until any of the following conditions is true:

- The socket becomes readable.
- The socket becomes writable.

- An ongoing connection request has completed (*wxSocketClient* (p. 1488) only)
- An incoming connection request has arrived (*wxSocketServer* (p. 1492) only)
- The connection has been closed.

Note that it is recommended to use the individual Wait functions to wait for the required condition, instead of this one.

### Parameters

*seconds*

Number of seconds to wait. If -1, it will wait for the default timeout, as set with *SetTimeout* (p. 1482).

*millisecond*

Number of milliseconds to wait.

### Return value

Returns true when any of the above conditions is satisfied, false if the timeout was reached.

### See also

*wxSocketBase::InterruptWait* (p. 1477), *wxSocketServer::WaitForAccept* (p. 1494), *wxSocketBase::WaitForLost* (p. 1485), *wxSocketBase::WaitForRead* (p. 1486), *wxSocketBase::WaitForWrite* (p. 1486), *wxSocketClient::WaitOnConnect* (p. 1489)

### **wxSocketBase::WaitForLost**

**bool Wait(long seconds = -1, long millisecond = 0)**

This function waits until the connection is lost. This may happen if the peer gracefully closes the connection or if the connection breaks.

### Parameters

*seconds*

Number of seconds to wait. If -1, it will wait for the default timeout, as set with *SetTimeout* (p. 1482).

*millisecond*

Number of milliseconds to wait.

### Return value

Returns true if the connection was lost, false if the timeout was reached.

### See also

*wxSocketBase::InterruptWait* (p. 1477), *wxSocketBase::Wait* (p. 1484)

### **wxSocketBase::WaitForRead**

**bool WaitForRead(long seconds = -1, long millisecond = 0)**

This function waits until the socket is readable. This might mean that queued data is available for reading or, for streamed sockets, that the connection has been closed, so that a read operation will complete immediately without blocking (unless the **wxSOCKET\_WAITALL** flag is set, in which case the operation might still block).

#### **Parameters**

*seconds*

Number of seconds to wait. If -1, it will wait for the default timeout, as set with *SetTimeout* (p. 1482).

*millisecond*

Number of milliseconds to wait.

#### **Return value**

Returns true if the socket becomes readable, false on timeout.

#### **See also**

*wxSocketBase::InterruptWait* (p. 1477), *wxSocketBase::Wait* (p. 1484)

### **wxSocketBase::WaitForWrite**

**bool WaitForWrite(long seconds = -1, long millisecond = 0)**

This function waits until the socket becomes writable. This might mean that the socket is ready to send new data, or for streamed sockets, that the connection has been closed, so that a write operation is guaranteed to complete immediately (unless the **wxSOCKET\_WAITALL** flag is set, in which case the operation might still block).

#### **Parameters**

*seconds*

Number of seconds to wait. If -1, it will wait for the default timeout, as set with *SetTimeout* (p. 1482).

*millisecond*

Number of milliseconds to wait.

#### **Return value**

Returns true if the socket becomes writable, false on timeout.

**See also**

*wxSocketBase::InterruptWait* (p. 1477), *wxSocketBase::Wait* (p. 1484)

**wxSocketBase::Write**

**wxSocketBase& Write(const void \* buffer, wxUint32 nbytes)**

This function writes a buffer of *nbytes* bytes to the socket.

Use *LastCount* (p. 1478) to verify the number of bytes actually written.

Use *Error* (p. 1476) to determine if the operation succeeded.

**Parameters**

*buffer*

Buffer with the data to be sent.

*nbytes*

Number of bytes.

**Return value**

Returns a reference to the current object.

**Remark/Warning**

The exact behaviour of *wxSocketBase::Write* depends on the combination of flags being used. For a detailed explanation, see *wxSocketBase::SetFlags* (p. 1480).

**See also**

*wxSocketBase::Error* (p. 1476), *wxSocketBase::LastError* (p. 1478),  
*wxSocketBase::LastCount* (p. 1478), *wxSocketBase::SetFlags* (p. 1480)

**wxSocketBase::WriteMsg**

**wxSocketBase& WriteMsg(const void \* buffer, wxUint32 nbytes)**

This function writes a buffer of *nbytes* bytes from the socket, but it writes a short header before so that *ReadMsg* (p. 1483) knows how much data should it actually read. So, a buffer sent with *WriteMsg* **must** be read with *ReadMsg*. This function always waits for the entire buffer to be sent, unless an error occurs.

Use *LastCount* (p. 1478) to verify the number of bytes actually written.

Use *Error* (p. 1476) to determine if the operation succeeded.

**Parameters**

*buffer*

Buffer with the data to be sent.

*nbytes*

Number of bytes to send.

### Return value

Returns a reference to the current object.

### Remark/Warning

`wxSocketBase::WriteMsg` will behave as if the **wxSOCKET\_WAITALL** flag was always set and it will always ignore the **wxSOCKET\_NOWAIT** flag. The exact behaviour of `WriteMsg` depends on the **wxSOCKET\_BLOCK** flag. For a detailed explanation, see `wxSocketBase::SetFlags` (p. 1480).

### See also

`wxSocketBase::Error` (p. 1476), `wxSocketBase::LastError` (p. 1478),  
`wxSocketBase::LastCount` (p. 1478), `wxSocketBase::SetFlags` (p. 1480),  
`wxSocketBase::ReadMsg` (p. 1483)

## wxSocketClient

### Derived from

`wxSocketBase` (p. 1472)

### Include files

<wx/socket.h>

### wxSocketClient::wxSocketClient

**wxSocketClient**(**wxSocketFlags** *flags* = `wxSOCKET_NONE`)

Constructor.

### Parameters

*flags*

Socket flags (See `wxSocketBase::SetFlags` (p. 1480))

### wxSocketClient::~~wxSocketClient

**~wxSocketClient**()

Destructor. Please see `wxSocketBase::Destroy` (p. 1476).



**wxSocketClient::Connect****bool Connect**(wxSockAddress& *address*, **bool** *wait* = *true*)**bool Connect**(wxSockAddress& *address*, wxSockAddress& *local*, **bool** *wait* = *true*)

Connects to a server using the specified address.

If *wait* is true, Connect will wait until the connection completes. **Warning:** This will block the GUI.

If *wait* is false, Connect will try to establish the connection and return immediately, without blocking the GUI. When used this way, even if Connect returns false, the connection request can be completed later. To detect this, use *WaitOnConnect* (p. 1489), or catch **wxSOCKET\_CONNECTION** events (for successful establishment) and **wxSOCKET\_LOST** events (for connection failure).

**Parameters***address*

Address of the server.

*local*

Bind to the specified local address and port before connecting. The local address and port can also be set using *SetLocal* (p. 1481), and then using the 2-parameter Connect method.

*wait*

If true, waits for the connection to complete.

**Return value**

Returns true if the connection is established and no error occurs.

If *wait* was true, and Connect returns false, an error occurred and the connection failed.

If *wait* was false, and Connect returns false, you should still be prepared to handle the completion of this connection request, either with *WaitOnConnect* (p. 1489) or by watching **wxSOCKET\_CONNECTION** and **wxSOCKET\_LOST** events.

**See also**

*wxSocketClient::WaitOnConnect* (p. 1489), *wxSocketBase::SetNotify* (p. 1481),  
*wxSocketBase::Notify* (p. 1478)

**wxSocketClient::WaitOnConnect****bool WaitOnConnect**(**long** *seconds* = -1, **long** *milliseconds* = 0)

Wait until a connection request completes, or until the specified timeout elapses. Use this function after issuing a call to *Connect* (p. 1489) with *wait* set to false.

**Parameters***seconds*

Number of seconds to wait. If -1, it will wait for the default timeout, as set with *SetTimeout* (p. 1482).

*millisecond*

Number of milliseconds to wait.

**Return value**

*WaitOnConnect* returns true if the connection request completes. This does not necessarily mean that the connection was successfully established; it might also happen that the connection was refused by the peer. Use *IsConnected* (p. 1477) to distinguish between these two situations.

If the timeout elapses, *WaitOnConnect* returns false.

These semantics allow code like this:

```
// Issue the connection request
client->Connect(addr, false);

// Wait until the request completes or until we decide to give up
bool waitmore = true;
while ( !client->WaitOnConnect(seconds, millis) && waitmore )
{
    // possibly give some feedback to the user,
    // and update waitmore as needed.
}
bool success = client->IsConnected();
```

**See also**

*wxSocketClient::Connect* (p. 1489), *wxSocketBase::InterruptWait* (p. 1477),  
*wxSocketBase::IsConnected* (p. 1477)

## **wxSocketEvent**

This event class contains information about socket events.

**Derived from**

*wxEvent* (p. 572)

**Include files**

<wx/socket.h>

**Event table macros**

To process a socket event, use these event handler macros to direct input to member

functions that take a `wxSocketEvent` argument.

**EVT\_SOCKET(id, func)**                      Process a socket event, supplying the member function.

**See also**

*wxSocketBase* (p. 1472), *wxSocketClient* (p. 1488), *wxSocketServer* (p. 1492)

## **wxSocketEvent::wxSocketEvent**

**wxSocketEvent(int id = 0)**

Constructor.

## **wxSocketEvent::GetClientData**

**void \* GetClientData()**

Gets the client data of the socket which generated this event, as set with *wxSocketBase::SetClientData* (p. 1479).

## **wxSocketEvent::GetSocket**

**wxSocketBase \* GetSocket() const**

Returns the socket object to which this event refers to. This makes it possible to use the same event handler for different sockets.

## **wxSocketEvent::GetSocketEvent**

**wxSocketNotify GetSocketEvent() const**

Returns the socket event type.

## **wxSocketInputStream**

This class implements an input stream which reads data from a connected socket. Note that this stream is purely sequential and it does not support seeking.

**Derived from**

*wxInputStream* (p. 941)

**Include files**

<wx/sckstrm.h>

**See also**

*wxSocketBase* (p. 1472)

### **wxSocketInputStream::wxSocketInputStream**

**wxSocketInputStream(wxSocketBase& s)**

Creates a new read-only socket stream using the specified initialized socket connection.

## **wxSocketOutputStream**

This class implements an output stream which writes data from a connected socket. Note that this stream is purely sequential and it does not support seeking.

### **Derived from**

*wxOutputStream* (p. 1156)

### **Include files**

<wx/sckstrm.h>

### **See also**

*wxSocketBase* (p. 1472)

### **wxSocketOutputStream::wxSocketOutputStream**

**wxSocketOutputStream(wxSocketBase& s)**

Creates a new write-only socket stream using the specified initialized socket connection.

## **wxSocketServer**

### **Derived from**

*wxSocketBase* (p. 1472)

### **Include files**

<wx/socket.h>

### **wxSocketServer::wxSocketServer**

**wxSocketServer(const wxSockAddress& address, wxSocketFlags flags = wxSOCKET\_NONE)**

Constructs a new server and tries to bind to the specified *address*. Before trying to accept new connections, test whether it succeeded with *wxSocketBase::IsOk* (p. 1478).

### Parameters

*address*

Specifies the local address for the server (e.g. port number).

*flags*

Socket flags (See *wxSocketBase::SetFlags* (p. 1480))

### **wxSocketServer::~wxSocketServer**

**~wxSocketServer()**

Destructor (it doesn't close the accepted connections).

### **wxSocketServer::Accept**

**wxSocketBase \* Accept**(**bool** *wait* = *true*)

Accepts an incoming connection request, and creates a new *wxSocketBase* (p. 1472) object which represents the server-side of the connection.

If *wait* is true and there are no pending connections to be accepted, it will wait for the next incoming connection to arrive. **Warning:** This will block the GUI.

If *wait* is false, it will try to accept a pending connection if there is one, but it will always return immediately without blocking the GUI. If you want to use *Accept* in this way, you can either check for incoming connections with *WaitForAccept* (p. 1494) or catch **wxSOCKET\_CONNECTION** events, then call *Accept* once you know that there is an incoming connection waiting to be accepted.

### Return value

Returns an opened socket connection, or NULL if an error occurred or if the *wait* parameter was false and there were no pending connections.

### See also

*wxSocketServer::WaitForAccept* (p. 1494), *wxSocketBase::SetNotify* (p. 1481), *wxSocketBase::Notify* (p. 1478), *wxSocketServer::AcceptWith* (p. 1493)

### **wxSocketServer::AcceptWith**

**bool AcceptWith**(**wxSocketBase&** *socket*, **bool** *wait* = *true*)

Accept an incoming connection using the specified socket object.

### Parameters

*socket*

Socket to be initialized

### Return value

Returns true on success, or false if an error occurred or if the *wait* parameter was false and there were no pending connections.

*wxSocketServer::WaitForAccept* (p. 1494), *wxSocketBase::SetNotify* (p. 1481),  
*wxSocketBase::Notify* (p. 1478), *wxSocketServer::Accept* (p. 1493)

## wxSocketServer::WaitForAccept

**bool WaitForAccept(long seconds = -1, long millisecond = 0)**

This function waits for an incoming connection. Use it if you want to call *Accept* (p. 1493) or *AcceptWith* (p. 1493) with *wait* set to false, to detect when an incoming connection is waiting to be accepted.

### Parameters

*seconds*

Number of seconds to wait. If -1, it will wait for the default timeout, as set with *SetTimeout* (p. 1482).

*millisecond*

Number of milliseconds to wait.

### Return value

Returns true if an incoming connection arrived, false if the timeout elapsed.

### See also

*wxSocketServer::Accept* (p. 1493), *wxSocketServer::AcceptWith* (p. 1493), *wxSocketBase::InterruptWait* (p. 1477)

## wxSound

This class represents a short sound (loaded from Windows WAV file), that can be stored in memory and played. Currently this class is implemented on Windows and Unix (and uses either Open Sound System (<http://www.opensound.com/oss.html>) or Simple DirectMedia Layer (<http://www.libsdl.org/>)).

### Derived from

*wxObject* (p. 1148)

### Include files

<wx/sound.h>

## **wxSound::wxSound**

### **wxSound()**

Default constructor.

### **wxSound(const wxString& fileName, bool isResource = false)**

Constructs a wave object from a file or, under Windows, from a Windows resource. Call *wxSound::IsOk* (p. 1495) to determine whether this succeeded.

### **Parameters**

*fileName*

The filename or Windows resource.

*isResource*

true if *fileName* is a resource, false if it is a filename.

## **wxSound::~~wxSound**

### **~wxSound()**

Destroys the wxSound object.

## **wxSound::Create**

### **bool Create(const wxString& fileName, bool isResource = false)**

Constructs a wave object from a file or resource.

### **Parameters**

*fileName*

The filename or Windows resource.

*isResource*

true if *fileName* is a resource, false if it is a filename.

### **Return value**

true if the call was successful, false otherwise.

## **wxSound::IsOk**

**bool IsOk() const**

Returns `true` if the object contains a successfully loaded file or resource, `false` otherwise.

**wxSound::IsPlaying****static bool IsPlaying() const**

Returns `true` if a sound is played at the moment.

This method is currently not implemented under Windows.

**wxSound::Play****bool Play(unsigned flags = wxSOUND\_ASYNC) const****static bool Play(const wxString& filename, unsigned flags = wxSOUND\_ASYNC)**

Plays the sound file. If another sound is playing, it will be interrupted. Returns `true` on success, `false` otherwise. Note that in general it is possible to delete the object which is being asynchronously played any time after calling this function and the sound would continue playing, however this currently doesn't work under Windows for sound objects loaded from memory data.

The possible values for *flags* are:

<code>wxSOUND_SYNC</code>	<code>Play</code> will block and wait until the sound is replayed.
<code>wxSOUND_ASYNC</code>	Sound is played asynchronously, <code>Play</code> returns immediately
<code>wxSOUND_ASYNC   wxSOUND_LOOP</code>	Sound is played asynchronously and loops until another sound is played, <code>wxSound::Stop</code> (p. 1496) is called or the program terminates.

The static form is shorthand for this code:

```
wxSound(filename).Play(flags);
```

**wxSound::Stop****static void Stop()**

If a sound is played, this function stops it.

**wxSpinButton**

A `wxSpinButton` has two small up and down (or left and right) arrow buttons. It is often used next to a text control for increment and decrementing a value. Portable programs



should try to use *wxSpinCtrl* (p. 1500) instead as *wxSpinButton* is not implemented for all platforms but *wxSpinCtrl* is as it degenerates to a simple *wxTextCtrl* (p. 1633) on such platforms.

**NB:** the range supported by this control (and *wxSpinCtrl*) depends on the platform but is at least `-0x8000` to `0x7fff`. Under GTK and Win32 with sufficiently new version of `comctl32.dll` (at least 4.71 is required, 5.80 is recommended) the full 32 bit range is supported.

#### Derived from

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

#### See also

*wxSpinCtrl* (p. 1500)

#### Include files

<wx/spinbutt.h>

#### Window styles

<b>wxSP_HORIZONTAL</b>	Specifies a horizontal spin button (note that this style is not supported in wxGTK).
<b>wxSP_VERTICAL</b>	Specifies a vertical spin button.
<b>wxSP_ARROW_KEYS</b>	The user can use arrow keys to change the value.
<b>wxSP_WRAP</b>	The value wraps at the minimum and maximum.

See also *window styles overview* (p. 2089).

#### Event handling

To process input from a spin button, use one of these event handler macros to direct input to member functions that take a *wxSpinEvent* (p. 1503) argument:

<b>EVT_SPIN(id, func)</b>	Generated whenever an arrow is pressed.
<b>EVT_SPIN_UP(id, func)</b>	Generated when left/up arrow is pressed.
<b>EVT_SPIN_DOWN(id, func)</b>	Generated when right/down arrow is pressed.

Note that if you handle both SPIN and UP or DOWN events, you will be notified about each of them twice: first the UP/DOWN event will be received and then, if it wasn't vetoed, the SPIN event will be sent. **See also** *Event handling overview* (p. 2077)

**wxSpinButton::wxSpinButton****wxSpinButton()**

Default constructor.

```
wxSpinButton(wxWindow* parent, wxWindowID id, const wxPoint& pos =  
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxSP_HORIZONTAL, const wxString& name = "spinButton")
```

Constructor, creating and showing a spin button.

**Parameters***parent*

Parent window. Must not be NULL.

*id*

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

*pos*

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

*size*

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

*style*

Window style. See *wxSpinButton* (p. 1496).

*name*

Window name.

**See also**

*wxSpinButton::Create* (p. 1498)

**wxSpinButton::~~wxSpinButton****void ~wxSpinButton()**

Destructor, destroys the spin button control.

**wxSpinButton::Create**

```
bool Create(wxWindow* parent, wxWindowID id, const wxPoint& pos =  
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxSP_HORIZONTAL, const wxString& name = "spinButton")
```

Scrollbar creation function called by the spin button constructor. See *wxSpinButton::wxSpinButton* (p. 1498) for details.

### **wxSpinButton::GetMax**

**int GetMax() const**

Returns the maximum permissible value.

#### **See also**

*wxSpinButton::SetRange* (p. 1499)

### **wxSpinButton::GetMin**

**int GetMin() const**

Returns the minimum permissible value.

#### **See also**

*wxSpinButton::SetRange* (p. 1499)

### **wxSpinButton::GetValue**

**int GetValue() const**

Returns the current spin button value.

#### **See also**

*wxSpinButton::SetValue* (p. 1500)

### **wxSpinButton::SetRange**

**void SetRange(int *min*, int *max*)**

Sets the range of the spin button.

#### **Parameters**

*min*

The minimum value for the spin button.

*max*

The maximum value for the spin button.

#### **See also**

*wxSpinButton::GetMin* (p. 1499), *wxSpinButton::GetMax* (p. 1499)

**wxSpinButton::SetValue****void SetValue**(int *value*)

Sets the value of the spin button.

**Parameters***value*

The value for the spin button.

**See also**

*wxSpinButton::GetValue* (p. 1499)

**wxSpinCtrl**

*wxSpinCtrl* combines *wxTextCtrl* (p. 1633) and *wxSpinButton* (p. 1496) in one control.

**Derived from***wxControl* (p. 285)*wxWindow* (p. 1795)*wxEvtHandler* (p. 576)*wxObject* (p. 1148)**Include files**

<wx/spinctrl.h>

**Window styles**

**wxSP\_ARROW\_KEYS**      The user can use arrow keys to change the value.

**wxSP\_WRAP**            The value wraps at the minimum and maximum.

**Event handling**

To process input from a spin button, use one of these event handler macros to direct input to member functions that take a *wxSpinEvent* (p. 1503) argument:

**EVT\_SPINCTRL(id, func)**      Generated whenever the numeric value of the spinctrl is updated

You may also use the *wxSpinButton* (p. 1496) event macros, however the corresponding events will not be generated under all platforms. Finally, if the user modifies the text in the edit part of the spin control directly, the **EVT\_TEXT** is generated, like for the *wxTextCtrl* (p. 1633).

When the user enters text into the text area, the text is not validated until the control loses focus (e.g. by using the TAB key). The value is then adjusted to the range and a *wxSpinEvent* (p. 1503) sent then if the value is different from the last value sent.

**See also**

*Event handling overview* (p. 2077), *wxSpinButton* (p. 1496), *wxControl* (p. 285)

**wxSpinCtrl::wxSpinCtrl****wxSpinCtrl()**

Default constructor.

```
wxSpinCtrl(wxWindow* parent, wxWindowID id = -1, const wxString& value =  
wxEmptyString, const wxPoint& pos = wxDefaultPosition, const wxSize& size =  
wxDefaultSize, long style = wxSP_ARROW_KEYS, int min = 0, int max = 100, int initial =  
0, const wxString& name = _T("wxSpinCtrl"))
```

Constructor, creating and showing a spin control.

**Parameters**

*parent*

Parent window. Must not be NULL.

*value*

Default value.

*id*

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

*pos*

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

*size*

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

*style*

Window style. See *wxSpinButton* (p. 1496).

*min*

Minimal value.

*max*

Maximal value.

*initial*

Initial value.

*name*

Window name.

### See also

*wxSpinCtrl::Create* (p. 1502)

### **wxSpinCtrl::Create**

```
bool Create(wxWindow* parent, wxWindowID id = -1, const wxString& value =  
wxEmptyString, const wxPoint& pos = wxDefaultPosition, const wxSize& size =  
wxDefaultSize, long style = wxSP_ARROW_KEYS, int min = 0, int max = 100, int initial =  
0, const wxString& name = _T("wxSpinCtrl"))
```

Creation function called by the spin control constructor.

See *wxSpinCtrl::wxSpinCtrl* (p. 1501) for details.

### **wxSpinCtrl::SetValue**

```
void SetValue(const wxString& text)
```

```
void SetValue(int value)
```

Sets the value of the spin control.

### **wxSpinCtrl::GetValue**

```
int GetValue() const
```

Gets the value of the spin control.

### **wxSpinCtrl::SetRange**

```
void SetRange(int minVal, int maxVal)
```

Sets range of allowable values.

### **wxSpinCtrl::SetSelection**

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

Select the text in the text part of the control between positions *from* (inclusive) and *to* (exclusive). This is similar to *wxTextCtrl::SetSelection* (p. 1650).

**NB:** this is currently only implemented for Windows and generic versions of the control.

### **wxSpinCtrl::GetMin**

**int GetMin() const**

Gets minimal allowable value.

**wxSpinCtrl::GetMax****int GetMax() const**

Gets maximal allowable value.

**wxSpinEvent**

This event class is used for the events generated by *wxSpinButton* (p. 1496) and *wxSpinCtrl* (p. 1500).

**Derived from**

*wxNotifyEvent* (p. 1146)  
*wxCommandEvent* (p. 250)  
*wxEvent* (p. 572)  
*wxObject* (p. 1148)

**Include files**

<wx/spinbutt.h> or <wx/spinctrl.h>

**Event handling**

To process input from a spin button, use one of these event handler macros to direct input to member functions that take a *wxSpinEvent* (p. 1503) argument:

<b>EVT_SPIN(id, func)</b>	Generated whenever an arrow is pressed.
<b>EVT_SPIN_UP(id, func)</b>	Generated when left/up arrow is pressed.
<b>EVT_SPIN_DOWN(id, func)</b>	Generated when right/down arrow is pressed.

Note that if you handle both SPIN and UP or DOWN events, you will be notified about each of them twice: first the UP/DOWN event will be received and then, if it wasn't vetoed, the SPIN event will be sent. **See also** *wxSpinButton* (p. 1496) and *wxSpinCtrl* (p. 1500)

**wxSpinEvent::wxSpinEvent**

**wxSpinEvent**(*wxEvtType* *commandType* = *wxEVT\_NULL*, *int* *id* = 0)

The constructor is not normally used by the user code.

**wxSpinEvent::GetPosition**

**int GetPosition() const**

Retrieve the current spin button or control value.

**wxSpinEvent::SetPosition****void SetPosition(int pos)**

Set the value associated with the event.

## wxSplashScreen

wxSplashScreen shows a window with a thin border, displaying a bitmap describing your application. Show it in application initialisation, and then either explicitly destroy it or let it time-out.

Example usage:

```
wxBitmap bitmap;
if (bitmap.LoadFile("splash16.png", wxBITMAP_TYPE_PNG))
{
    wxSplashScreen* splash = new wxSplashScreen(bitmap,
        wxSPLASH_CENTRE_ON_SCREEN|wxSPLASH_TIMEOUT,
        6000, NULL, -1, wxDefaultPosition, wxDefaultSize,
        wxSIMPLE_BORDER|wxSTAY_ON_TOP);
}
wxYield();
```

**Derived from**

*wxFrame* (p. 682)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

**Include files**

<wx/splash.h>

**wxSplashScreen::wxSplashScreen**

```
wxSplashScreen(const wxBitmap& bitmap, long splashStyle, int milliseconds,  
wxWindow* parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition, const  
wxSize& size = wxDefaultSize, long style =  
wxSIMPLE_BORDER|wxFRAME_NO_TASKBAR|wxSTAY_ON_TOP)
```

Construct the splash screen passing a bitmap, a style, a timeout, a window id, optional position and size, and a window style.

*splashStyle* is a bitlist of some of the following:



- `wxSPLASH_CENTRE_ON_PARENT`
- `wxSPLASH_CENTRE_ON_SCREEN`
- `wxSPLASH_NO_CENTRE`
- `wxSPLASH_TIMEOUT`
- `wxSPLASH_NO_TIMEOUT`

*milliseconds* is the timeout in milliseconds.

### **`wxSplashScreen::~~wxSplashScreen`**

**`~wxSplashScreen()`**

Destroys the splash screen.

### **`wxSplashScreen::OnCloseWindow`**

**`void OnCloseWindow(wxCloseEvent& event)`**

Reimplement this event handler if you want to set an application variable on window destruction, for example.

### **`wxSplashScreen::GetSplashStyle`**

**`long GetSplashStyle() const`**

Returns the splash style (see *`wxSplashScreen::wxSplashScreen`* (p. 1504) for details).

### **`wxSplashScreen::GetSplashWindow`**

**`wxSplashScreenWindow* GetSplashWindow() const`**

Returns the window used to display the bitmap.

### **`wxSplashScreen::GetTimeout`**

**`int GetTimeout() const`**

Returns the timeout in milliseconds.

## **`wxSplitterEvent`**

This class represents the events generated by a splitter control. Also there is only one event class, the data associated to the different events is not the same and so not all accessor functions may be called for each event. The documentation mentions the kind of event(s) for which the given accessor function makes sense: calling it for other types of events will result in assert failure (in debug mode) and will return meaningless results.

**Derived from**

*wxNotifyEvent* (p. 1146)  
*wxCommandEvent* (p. 250)  
*wxEvent* (p. 572)  
*wxObject* (p. 1148)

**Include files**

<wx/splitter.h>

**Event table macros**

To process a splitter event, use these event handler macros to direct input to member functions that take a `wxSplitterEvent` argument.

<b>EVT_SPLITTER_SASH_POS_CHANGING(id, func)</b>	The sash position is in the process of being changed. You may prevent this change from happening by calling <i>Veto</i> (p. 1147) or you may also modify the position of the tracking bar to properly reflect the position that would be set if the drag were to be completed at this point. Processes a <code>wxEVT_COMMAND_SPLITTER_SASH_POS_CHANGING</code> event.
<b>EVT_SPLITTER_SASH_POS_CHANGED(id, func)</b>	The sash position was changed. This event is generated after the user releases the mouse after dragging the splitter. Processes a <code>wxEVT_COMMAND_SPLITTER_SASH_POS_CHANGED</code> event.
<b>EVT_SPLITTER_UNSPLIT(id, func)</b>	The splitter has been just unsplit. Processes a <code>wxEVT_COMMAND_SPLITTER_UNSPLIT</code> event. This event can't be vetoed.
<b>EVT_SPLITTER_DCLICK(id, func)</b>	The sash was double clicked. The default behaviour is to unsplit the window when this happens (unless the minimum pane size has been set to a value greater than zero). This

won't happen if you veto this event. Processes a `wxEVT_COMMAND_SPLITTER_DOUBLECLICKED` event.

**See also**

*wxSplitterWindow* (p. 1508), *Event handling overview* (p. 2077)

**wxSplitterEvent::wxSplitterEvent**

**wxSplitterEvent(wxEventType eventType = wxEVT\_NULL, wxSplitterWindow \* splitter = NULL)**

Constructor. Used internally by wxWidgets only.

**wxSplitterEvent::GetSashPosition**

**int GetSashPosition() const**

Returns the new sash position.

May only be called while processing `wxEVT_COMMAND_SPLITTER_SASH_POS_CHANGING` and `wxEVT_COMMAND_SPLITTER_SASH_POS_CHANGED` events.

**wxSplitterEvent::GetX**

**int GetX() const**

Returns the x coordinate of the double-click point.

May only be called while processing `wxEVT_COMMAND_SPLITTER_DOUBLECLICKED` events.

**wxSplitterEvent::GetY**

**int GetY() const**

Returns the y coordinate of the double-click point.

May only be called while processing `wxEVT_COMMAND_SPLITTER_DOUBLECLICKED` events.

**wxSplitterEvent::GetWindowBeingRemoved**

**wxWindow\* GetWindowBeingRemoved() const**

Returns a pointer to the window being removed when a splitter window is unsplit.

May only be called while processing `wxEVT_COMMAND_SPLITTER_UNSPLOT` events.

### **wxSplitterEvent::SetSashPosition**

**void SetSashPosition(int pos)**

In the case of `wxEVT_COMMAND_SPLITTER_SASH_POS_CHANGED` events, sets the new sash position. In the case of `wxEVT_COMMAND_SPLITTER_SASH_POS_CHANGING` events, sets the new tracking bar position so visual feedback during dragging will represent that change that will actually take place. Set to -1 from the event handler code to prevent repositioning.

May only be called while processing `wxEVT_COMMAND_SPLITTER_SASH_POS_CHANGING` and `wxEVT_COMMAND_SPLITTER_SASH_POS_CHANGED` events.

#### **Parameters**

*pos*

New sash position.

## **wxSplitterWindow**

*wxSplitterWindow overview* (p. 2123)

This class manages up to two subwindows. The current view can be split into two programmatically (perhaps from a menu command), and unsplit either programmatically or via the `wxSplitterWindow` user interface.

#### **Window styles**

<b>wxSP_3D</b>	Draws a 3D effect border and sash.
<b>wxSP_3DSASH</b>	Draws a 3D effect sash.
<b>wxSP_3DBORDER</b>	Synonym for <code>wxSP_BORDER</code> .
<b>wxSP_BORDER</b>	Draws a standard border.
<b>wxSP_NOBORDER</b>	No border (default).
<b>wxSP_NO_XP_THEME</b>	Under Windows XP, switches off the attempt to draw the splitter using Windows XP theming, so the borders and sash will take on the pre-XP look.
<b>wxSP_PERMIT_UNSPLOT</b>	Always allow to unsplit, even

with the minimum pane size other than zero.

## **wxSP\_LIVE\_UPDATE**

Don't draw XOR line but resize the child windows immediately.

See also *window styles overview* (p. 2089).

### **Derived from**

*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### **Include files**

<wx/splitter.h>

### **Event handling**

To process input from a splitter control, use the following event handler macros to direct input to member functions that take a *wxSplitterEvent* (p. 1505) argument.

#### **EVT\_SPLITTER\_SASH\_POS\_CHANGING(id, func)**

The sash position is in the process of being changed. May be used to modify the position of the tracking bar to properly reflect the position that would be set if the drag were to be completed at this point. Processes a `wxEVT_COMMAND_SPLITTER_SASH_POS_CHANGING` event.

#### **EVT\_SPLITTER\_SASH\_POS\_CHANGED(id, func)**

The sash position was changed. May be used to modify the sash position before it is set, or to prevent the change from taking place. Processes a `wxEVT_COMMAND_SPLITTER_SASH_POS_CHANGED` event.

#### **EVT\_SPLITTER\_UNSPLOT(id, func)**

The splitter has been just unsplit. Processes a `wxEVT_COMMAND_SPLITTER_UNSPLOT` event.

#### **EVT\_SPLITTER\_DCLICK(id, func)**

The sash was double clicked. The default behaviour is to

unsplit the window when this happens (unless the minimum pane size has been set to a value greater than zero).  
Processes a  
`wxEVT_COMMAND_SPLITTER_DOUBLECLICKED`  
event.

**See also**

*wxSplitterEvent* (p. 1505)

**wxSplitterWindow::wxSplitterWindow****wxSplitterWindow()**

Default constructor.

**wxSplitterWindow**(*wxWindow\** parent, *wxWindowID* id, **const** *wxPoint&* point = *wxDefaultPosition*, **const** *wxSize&* size = *wxDefaultSize*, **long** style=*wxSP\_3D*, **const** *wxString&* name = *"splitterWindow"*)

Constructor for creating the window.

**Parameters**

*parent*

The parent of the splitter window.

*id*

The window identifier.

*pos*

The window position.

*size*

The window size.

*style*

The window style. See *wxSplitterWindow* (p. 1508).

*name*

The window name.

**Remarks**

After using this constructor, you must create either one or two subwindows with the splitter

window as parent, and then call one of `wxSplitterWindow::Initialize` (p. 1512), `wxSplitterWindow::SplitVertically` (p. 1517) and `wxSplitterWindow::SplitHorizontally` (p. 1516) in order to set the pane(s).

You can create two windows, with one hidden when not being shown; or you can create and delete the second pane on demand.

#### See also

`wxSplitterWindow::Initialize` (p. 1512), `wxSplitterWindow::SplitVertically` (p. 1517), `wxSplitterWindow::SplitHorizontally` (p. 1516), `wxSplitterWindow::Create` (p. 1511)

### **wxSplitterWindow::~~wxSplitterWindow**

**~wxSplitterWindow()**

Destroys the `wxSplitterWindow` and its children.

### **wxSplitterWindow::Create**

**bool Create(wxWindow\* parent, wxWindowID id, const wxPoint& point = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style=wxSP\_3D, const wxString& name = "splitterWindow")**

Creation function, for two-step construction. See `wxSplitterWindow::wxSplitterWindow` (p. 1510) for details.

### **wxSplitterWindow::GetMinimumPaneSize**

**int GetMinimumPaneSize() const**

Returns the current minimum pane size (defaults to zero).

#### See also

`wxSplitterWindow::SetMinimumPaneSize` (p. 1515)

### **wxSplitterWindow::GetSashGravity**

**double GetSashGravity()**

Returns the current sash gravity.

#### See also

`wxSplitterWindow::SetSashGravity` (p. 1514)

### **wxSplitterWindow::GetSashPosition**

**int GetSashPosition()**

Returns the current sash position.

**See also**

*wxSplitterWindow::SetSashPosition* (p. 1515)

**wxSplitterWindow::GetSplitMode**

**int GetSplitMode() const**

Gets the split mode.

**See also**

*wxSplitterWindow::SetSplitMode* (p. 1516), *wxSplitterWindow::SplitVertically* (p. 1517), *wxSplitterWindow::SplitHorizontally* (p. 1516).

**wxSplitterWindow::GetWindow1**

**wxWindow\* GetWindow1() const**

Returns the left/top or only pane.

**wxSplitterWindow::GetWindow2**

**wxWindow\* GetWindow2() const**

Returns the right/bottom pane.

**wxSplitterWindow::Initialize**

**void Initialize(wxWindow\* window)**

Initializes the splitter window to have one pane. The child window is shown if it is currently hidden.

**Parameters**

*window*

The pane for the unsplit window.

**Remarks**

This should be called if you wish to initially view only a single pane in the splitter window.

**See also**

*wxSplitterWindow::SplitVertically* (p. 1517), *wxSplitterWindow::SplitHorizontally* (p. 1516)

**wxSplitterWindow::IsSplit**

**bool IsSplit() const**



Returns true if the window is split, false otherwise.

### **wxSplitterWindow::OnDoubleClickSash**

**virtual void OnDoubleClickSash(int x, int y)**

Application-overrideable function called when the sash is double-clicked with the left mouse button.

#### **Parameters**

*x*

The x position of the mouse cursor.

*y*

The y position of the mouse cursor.

#### **Remarks**

The default implementation of this function calls *Unsplit* (p. 1518) if the minimum pane size is zero.

#### **See also**

*wxSplitterWindow::Unsplit* (p. 1518)

### **wxSplitterWindow::OnUnsplit**

**virtual void OnUnsplit(wxWindow\* removed)**

Application-overrideable function called when the window is unsplit, either programmatically or using the *wxSplitterWindow* user interface.

#### **Parameters**

*removed*

The window being removed.

#### **Remarks**

The default implementation of this function simply hides *removed*. You may wish to delete the window.

### **wxSplitterWindow::OnSashPositionChange**

**virtual bool OnSashPositionChange(int newSashPosition)**

Application-overrideable function called when the sash position is changed by user. It may return false to prevent the change or true to allow it.

#### **Parameters**

*newSashPosition*

The new sash position (always positive or zero)

#### Remarks

The default implementation of this function verifies that the sizes of both panes of the splitter are greater than minimum pane size.

### **wxSplitterWindow::ReplaceWindow**

**bool ReplaceWindow(wxWindow \* winOld, wxWindow \* winNew)**

This function replaces one of the windows managed by the wxSplitterWindow with another one. It is in general better to use it instead of calling `Unsplit()` and then resplitting the window back because it will provoke much less flicker (if any). It is valid to call this function whether the splitter has two windows or only one.

Both parameters should be non-NULL and *winOld* must specify one of the windows managed by the splitter. If the parameters are incorrect or the window couldn't be replaced, false is returned. Otherwise the function will return true, but please notice that it will not delete the replaced window and you may wish to do it yourself.

#### See also

*wxSplitterWindow::GetMinimumPaneSize* (p. 1511)

#### See also

*wxSplitterWindow::Unsplit* (p. 1518)

*wxSplitterWindow::SplitVertically* (p. 1517)

*wxSplitterWindow::SplitHorizontally* (p. 1516)

### **wxSplitterWindow::SetSashGravity**

**void SetSashGravity(double gravity)**

Sets the sash gravity.

#### Parameters

*gravity*

The sash gravity. Value between 0.0 and 1.0.

**Remarks** Gravity is real factor which controls position of sash while resizing wxSplitterWindow. Gravity tells wxSplitterWindow how much will left/top window grow while resizing.

Example values:

- 0.0 - only the bottom/right window is automatically resized
- 0.5 - both windows grow by equal size

- 1.0 - only left/top window grows

Gravity should be a real value between 0.0 and 1.0.

Default value of sash gravity is 0.0. That value is compatible with previous (before gravity was introduced) behaviour of `wxSplitterWindow`.

**See also**

`wxSplitterWindow::GetSashGravity` (p. 1511)

**`wxSplitterWindow::SetSashPosition`**

**`void SetSashPosition(int position, const bool redraw = true)`**

Sets the sash position.

**Parameters**

*position*

The sash position in pixels.

*redraw*

If true, resizes the panes and redraws the sash and border.

**Remarks**

Does not currently check for an out-of-range value.

**See also**

`wxSplitterWindow::GetSashPosition` (p. 1511)

**`wxSplitterWindow::SetSashSize`**

**`void SetSashSize(int size)`**

Sets the sash size. Normally, the sash size is determined according to the metrics of each platform, but the application can override this, for example to show a thin sash that the user is not expected to drag. If *size* is more -1, the custom sash size will be used.

**`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. If the `wxSP_PERMIT_UNSPPLIT` style is used when a splitter window is created, the window may be unsplit even if minimum size is non-zero.

**See also**

*wxSplitterWindow::GetMinimumPaneSize* (p. 1511)

**wxSplitterWindow::SetSplitMode**

**void SetSplitMode(int mode)**

Sets the split mode.

**Parameters**

*mode*

Can be `wxSPLIT_VERTICAL` or `wxSPLIT_HORIZONTAL`.

**Remarks**

Only sets the internal variable; does not update the display.

**See also**

*wxSplitterWindow::GetSplitMode* (p. 1512), *wxSplitterWindow::SplitVertically* (p. 1517), *wxSplitterWindow::SplitHorizontally* (p. 1516).

**wxSplitterWindow::SplitHorizontally**

**bool SplitHorizontally(wxWindow\* window1, wxWindow\* window2, int sashPosition = 0)**

Initializes the top and bottom panes of the splitter window. The child windows are shown if they are currently hidden.

**Parameters**

*window1*

The top pane.

*window2*

The bottom pane.

*sashPosition*

The initial position of the sash. If this value is positive, it specifies the size of the

upper pane. If it is negative, its absolute value gives the size of the lower pane. Finally, specify 0 (default) to choose the default position (half of the total window height).

**Return value**

true if successful, false otherwise (the window was already split).

**Remarks**

This should be called if you wish to initially view two panes. It can also be called at any subsequent time, but the application should check that the window is not currently split using *IsSplit* (p. 1512).

**See also**

*wxSplitterWindow::SplitVertically* (p. 1517), *wxSplitterWindow::IsSplit* (p. 1512), *wxSplitterWindow::Unsplit* (p. 1518)

**wxSplitterWindow::SplitVertically**

**bool SplitVertically(wxWindow\* window1, wxWindow\* window2, int sashPosition = 0)**

Initializes the left and right panes of the splitter window. The child windows are shown if they are currently hidden.

**Parameters**

*window1*

The left pane.

*window2*

The right pane.

*sashPosition*

The initial position of the sash. If this value is positive, it specifies the size of the left pane. If it is negative, its absolute value gives the size of the right pane. Finally, specify 0 (default) to choose the default position (half of the total window width).

**Return value**

true if successful, false otherwise (the window was already split).

**Remarks**

This should be called if you wish to initially view two panes. It can also be called at any subsequent time, but the application should check that the window is not currently split using *IsSplit* (p. 1512).

**See also**

*wxSplitterWindow::SplitHorizontally* (p. 1516), *wxSplitterWindow::IsSplit* (p. 1512),

*wxSplitterWindow::Unsplit* (p. 1518).

## **wxSplitterWindow::Unsplit**

**bool Unsplit(wxWindow\* toRemove = NULL)**

Unsplits the window.

### **Parameters**

*toRemove*

The pane to remove, or NULL to remove the right or bottom pane.

### **Return value**

true if successful, false otherwise (the window was not split).

### **Remarks**

This call will not actually delete the pane being removed; it calls *OnUnsplit* (p. 1513) which can be overridden for the desired behaviour. By default, the pane being removed is hidden.

### **See also**

*wxSplitterWindow::SplitHorizontally* (p. 1516), *wxSplitterWindow::SplitVertically* (p. 1517), *wxSplitterWindow::IsSplit* (p. 1512), *wxSplitterWindow::OnUnsplit* (p. 1513)

## **wxSplitterWindow::UpdateSize**

**void UpdateSize()**

Causes any pending sizing of the sash and child panes to take place immediately.

Such resizing normally takes place in idle time, in order to wait for layout to be completed. However, this can cause unacceptable flicker as the panes are resized after the window has been shown. To work around this, you can perform window layout (for example by sending a size event to the parent window), and then call this function, before showing the top-level window.

## **wxSplitterRenderParams**

This is just a simple `struct` used as a return value of *wxRendererNative::GetSplitterParams* (p. 1280).

It doesn't have any methods and all of its fields are constant and so can be only examined but not modified.

### **Include files**

<wx/renderer.h>

**wxEvent::widthSash****const wxCoord widthSash**

The width of the splitter sash.

**wxSplitterRenderParams::border****const wxCoord border**

The width of the border drawn by the splitter inside it, may be 0.

**wxSplitterRenderParams::isHotSensitive****const bool isHotSensitive**

`true` if the sash changes appearance when the mouse passes over it, `false` otherwise.

**wxStackFrame**

`wxStackFrame` represents a single stack frame, or a single function in the call stack, and is used exclusively together with `wxStackWalker` (p. 1521), see there for a more detailed discussion.

**Derived from**

No base class

**Include files**

<wx/stackwalk.h>

Only available if `wxUSE_STACKWALKER` is 1, currently only implemented for Win32 and Unix versions using recent version of GNU libc.

**See also**

`wxStackWalker` (p. 1521)

**wxStackFrame::GetAddress****void\* GetAddress() const**

Return the address of this frame.

**wxStackFrame::GetFileName**

**wxString GetFileName() const**

Return the name of the file containing this frame, empty if unavailable (typically because debug info is missing).

Use *HasSourceLocation* (p. 1521) to check whether the file name is available.

**wxStackFrame::GetLevel****size\_t GetLevel() const**

Get the level of this frame (deepest/innermost one is 0).

**wxStackFrame::GetLine****size\_t GetLine() const**

Return the line number of this frame, 0 if unavailable.

**See also**

*GetFileName* (p. 1519)

**wxStackFrame::GetModule****wxString GetModule() const**

Get the module this function belongs to (empty if not available).

**wxStackFrame::GetName****wxString GetName() const**

Return the unmangled (if possible) name of the function containing this frame.

**wxStackFrame::GetOffset****size\_t GetOffset() const**

Return the return address of this frame.

**wxStackFrame::GetParam****bool GetParam(size\_t n, wxString \* type, wxString \* name, wxString \* value) const**

Get the name, type and value (in text form) of the given parameter. Any pointer may be `NULL` if you're not interested in the corresponding value.

Return `true` if at least some values could be retrieved.

This function currently is only implemented under Win32 and requires a PDB file.



**wxStackFrame::GetParamCount****size\_t GetParamCount() const**

Return the number of parameters of this function (may return 0 if we can't retrieve the parameters info even although the function does have parameters).

**wxStackFrame::HasSourceLocation****bool HasSourceLocation() const**

Return `true` if we have the file name and line number for this frame.

**wxStackWalker**

`wxStackWalker` allows an application to enumerate, or walk, the stack frames (the function callstack). It is mostly useful in only two situations: inside `wxApp::OnFatalException` (p. 51) function to programmatically get the location of the crash and, in debug builds, in `wxApp::OnAssertFailure` (p. 49) to report the caller of the failed assert.

`wxStackWalker` works by repeatedly calling the `OnStackFrame` (p. 1522) method for each frame in the stack, so to use it you must derive your own class from it and override this method.

This class will not return anything except raw stack frame addresses if the debug information is not available. Under Win32 this means that the PDB file matching the program being executed should be present. Note that if you use Microsoft Visual C++ compiler, you can create PDB files even for the programs built in release mode and it doesn't affect the program size (at least if you don't forget to add `/opt:ref` option which is suppressed by using `/debug` linker option by default but should be always enabled for release builds). Under Unix, you need to compile your program with debugging information (usually using `-g` compiler and linker options) to get the file and line numbers information, however function names should be available even without it. Of course, all this is only true if you build using a recent enough version of GNU libc which provides the `backtrace()` function needed to walk the stack.

*debugging overview* (p. 2072) for how to make it available.

**Derived from**

No base class

**Include files**

<wx/stackwalk.h>

Only available if `wxUSE_STACKWALKER` is 1, currently only implemented for Win32 and Unix versions using recent version of GNU libc.

**See also**

*wxStackFrame* (p. 1519)

### **wxStackWalker::wxStackWalker**

**wxStackWalker()**

Constructor does nothing, use *Walk()* (p. 1522) to walk the stack.

### **wxStackWalker::~~wxStackWalker**

**~wxStackWalker()**

Destructor does nothing neither but should be virtual as this class is used as a base one.

### **wxStackWalker::OnStackFrame**

**void OnStackFrame(const wxStackFrame& frame)**

This function must be overridden to process the given frame.

### **wxStackWalker::Walk**

**void Walk(size\_t skip = 1, size\_t maxDepth = 200)**

Enumerate stack frames from the current location, skipping the initial number of them (this can be useful when *Walk()* is called from some known location and you don't want to see the first few frames anyhow; also notice that *Walk()* frame itself is not included if *skip*  $\geq$  1).

Up to *maxDepth* frames are walked from the innermost to the outermost one.

### **wxStackWalker::WalkFromException**

**void WalkFromException()**

Enumerate stack frames from the location of uncaught exception. This method can only be called from *wxApp::OnFatalException()* (p. 51).

## **wxStandardPaths**

*wxStandardPaths* returns the standard locations in the file system and should be used by applications to find their data files in a portable way.

In the description of the methods below, the example return values are given for the Unix, Windows and Mac OS X systems, however please note that these are just the examples and the actual values may differ. For example, under Windows: the system administrator may change the standard directories locations, i.e. the Windows directory may be named *W:\Win2003* instead of the default *C:\Windows*.

The strings *appname* and *username* should be replaced with the value returned by `wxApp::GetAppName` (p. 47) and the name of the currently logged in user, respectively. The string *prefix* is only used under Unix and is `/usr/local` by default but may be changed using `SetInstallPrefix` (p. 1527).

The directories returned by the methods of this class may or may not exist. If they don't exist, it's up to the caller to create them, `wxStandardPaths` doesn't do it.

Finally note that these functions only work with standardly packaged applications. I.e. under Unix you should follow the standard installation conventions and under Mac you should create your application bundle according to the Apple guidelines. Again, this class doesn't help you to do it.

This class is MT-safe: its methods may be called concurrently from different threads without additional locking.

### **Derived from**

No base class

### **Include files**

<wx/stdpaths.h>

## **wxStandardPaths::Get**

**static wxStandardPathsBase& Get()**

Returns reference to the unique global standard paths object.

## **wxStandardPaths::GetConfigDir**

**wxString GetConfigDir() const**

Return the directory containing the system config files.

Example return values:

- Unix: `/etc`
- Windows: `C:\Documents and Settings\All Users\Application Data`
- Mac: `/Library/Preferences`

### **See also**

`wxFileConfig` (p. 598)

## **wxStandardPaths::GetDataDir**

**wxString GetDataDir() const**

Return the location of the applications global, i.e. not user-specific, data files.

Example return values:

- Unix: *prefix/share/appname*
- Windows: the directory where the executable file is located
- Mac: *appname.app/Contents/SharedSupport* bundle subdirectory

#### **See also**

*GetLocalDataDir* (p. 1525)

### **wxStandardPaths::GetDocumentsDir**

#### **wxString GetDocumentsDir() const**

Return the directory containing the current user's documents.

Example return values:

- Unix: *~* (the home directory)
- Windows: *C:\Documents and Settings\username\Documents*
- Mac: *~/Documents*

This function is new since wxWidgets version 2.7.0

### **wxStandardPaths::GetExecutablePath**

#### **wxString GetExecutablePath() const**

Return the directory and the filename for the current executable.

Example return values:

- Unix: */usr/local/bin/exename*
- Windows: *C:\Programs\AppFolder\exename.exe*
- Mac: */Programs/exename*

### **wxStandardPaths::GetInstallPrefix**

#### **wxString GetInstallPrefix() const**

**Note:** This function is only available under Unix.

Return the program installation prefix, e.g. */usr*, */opt* or */home/zeitlin*.

If the prefix had been previously by *SetInstallPrefix* (p. 1527), returns that value, otherwise

tries to determine it automatically (Linux only right now) and finally returns the default `/usr/local` value if it failed.

### **wxStandardPaths::GetLocalDataDir**

#### **wxString GetLocalDataDir() const**

Return the location for application data files which are host-specific and can't, or shouldn't, be shared with the other machines.

This is the same as *GetDataDir()* (p. 1523) except under Unix where it returns */etc/appname*.

### **wxStandardPaths::GetLocalizedResourcesDir**

#### **wxString GetLocalizedResourcesDir(const wxChar\* lang, ResourceCat category = ResourceCat\_None) const**

Return the localized resources directory containing the resource files of the specified category for the given language.

In general this is just the same as *lang* subdirectory of *GetResourcesDir()* (p. 1525) (*orlang.lproj* under Mac OS X) but is something quite different for message catalog category under Unix where it returns the *standardprefix/share/locale/lang/LC\_MESSAGES* directory.

This function is new since wxWidgets version 2.7.0

### **wxStandardPaths::GetPluginsDir**

#### **wxString GetPluginsDir() const**

Return the directory where the loadable modules (plugins) live.

Example return values:

- Unix: *prefix/lib/appname*
- Windows: the directory of the executable file
- Mac: *appname.app/Contents/PlugIns* bundle subdirectory

#### **See also**

*wxDynamicLibrary* (p. 564)

### **wxStandardPaths::GetResourcesDir**

#### **wxString GetResourcesDir() const**

Return the directory where the application resource files are located. The resources are the auxiliary data files needed for the application to run and include, for example, image

and sound files it might use.

This function is the same as *GetDataDir* (p. 1523) for all platforms except Mac OS X.

Example return values:

- Unix: *prefix/share/appname*
- Windows: the directory where the executable file is located
- Mac: *appname.app/Contents/Resources* bundle subdirectory

This function is new since wxWidgets version 2.7.0

### See also

*GetLocalizedResourcesDir* (p. 1525)

## **wxStandardPaths::GetTempDir**

### **wxString GetTempDir() const**

Return the directory for storing temporary files. To create unique temporary files, it is best to use *wxFileName::CreateTempFileName* (p. 615) for correct behaviour when multiple processes are attempting to create temporary files.

This function is new since wxWidgets version 2.7.2

## **wxStandardPaths::GetUserConfigDir**

### **wxString GetUserConfigDir() const**

Return the directory for the user config files:

- Unix: *~* (the home directory)
- Windows: *C:\Documents and Settings\username\Application Data*
- Mac: *~/Library/Preferences*

Only use this method if you have a single configuration file to put in this directory, otherwise *GetUserDataDir* (p. 1526) is more appropriate.

## **wxStandardPaths::GetUserDataDir**

### **wxString GetUserDataDir() const**

Return the directory for the user-dependent application data files:

- Unix: *~/ .appname*
- Windows: *C:\Documents and Settings\username\Application Data\appname*

- Mac: ~/Library/Application Support/*appname*

### **wxStandardPaths::GetUserLocalDataDir**

#### **wxString GetUserLocalDataDir() const**

Return the directory for user data files which shouldn't be shared with the other machines.

This is the same as *GetUserDataDir()* (p. 1526) for all platforms except Windows where it returns `SC:\Documents and Settings\username\Local Settings\Application Data\appname`

### **wxStandardPaths::SetInstallPrefix**

#### **void SetInstallPrefix(const wxString& prefix)**

**Note:** This function is only available under Unix.

Lets *wxStandardPaths* know about the real program installation prefix on a Unix system. By default, the value returned by *GetInstallPrefix* (p. 1524) is used.

Although under Linux systems the program prefix may usually be determined automatically, portable programs should call this function. Usually the prefix is set during program configuration if using GNU autotools and so it is enough to pass its value defined in `config.h` to this function.

## **wxStaticBitmap**

A static bitmap control displays a bitmap. It is meant for display of the small icons in the dialog boxes and is not meant to be a general purpose image display control. In particular, under Windows 9x the size of bitmap is limited to 64\*64 pixels and thus you should use your own control if you want to display larger images portably.

### **Derived from**

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### **Include files**

<wx/statbmp.h>

### **Window styles**

There are no special styles for this control.

See also *window styles overview* (p. 2089).

### **See also**

*wxStaticBitmap* (p. 1527), *wxStaticBox* (p. 1530)

### Remarks

The bitmap to be displayed should have a small number of colours, such as 16, to avoid palette problems.

## **wxStaticBitmap::wxStaticBitmap**

### **wxStaticBitmap()**

Default constructor.

**wxStaticBitmap**(*wxWindow\** *parent*, *wxWindowID* *id*, **const** *wxBitmap&* *label*, **const** *wxPoint&* *pos* = *wxDefaultPosition*, **const** *wxSize&* *size* = *wxDefaultSize*, **long** *style* = 0, **const** *wxString&* *name* = "*staticBitmap*")

Constructor, creating and showing a static bitmap control.

### Parameters

*parent*

Parent window. Should not be NULL.

*id*

Control identifier. A value of -1 denotes a default value.

*label*

Bitmap label.

*pos*

Window position.

*size*

Window size.

*style*

Window style. See *wxStaticBitmap* (p. 1527).

*name*

Window name.

### See also

*wxStaticBitmap::Create* (p. 1529)



**wxStaticBitmap::Create**

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

Creation function, for two-step construction. For details see *wxStaticBitmap::wxStaticBitmap* (p. 1528).

**wxStaticBitmap::GetBitmap**

**wxBitmap GetBitmap()** const

Returns the bitmap currently used in the control. Notice that this method can be called even if *SetIcon* (p. 1529) had been used.

**See also**

*wxStaticBitmap::SetBitmap* (p. 1529)

**wxStaticBitmap::GetIcon**

**wxIcon GetIcon()** const

Returns the icon currently used in the control. Notice that this method can only be called if *SetIcon* (p. 1529) had been used: an icon can't be retrieved from the control if a bitmap had been set (using *SetBitmap* (p. 1529)).

**See also**

*wxStaticBitmap::SetIcon* (p. 1529)

**wxStaticBitmap::SetBitmap**

**virtual void SetBitmap**(const wxBitmap& *label*)

Sets the bitmap label.

**Parameters**

*label*

The new bitmap.

**See also**

*wxStaticBitmap::GetBitmap* (p. 1529)

**wxStaticBitmap::SetIcon**

**virtual void SetIcon**(const wxIcon& *label*)

Sets the label to the given icon.

### Parameters

*label*

The new icon.

### See also

*wxStaticBitmap::GetIcon* (p. 1529)

## wxStaticBox

A static box is a rectangle drawn around other panel items to denote a logical grouping of items.

Please note that a static box should **not** be used as the parent for the controls it contains, instead they should be siblings of each other. Although using a static box as a parent might work in some versions of wxWidgets, it results in a crash under, for example, wxGTK.

Also, please note that because of this, the order in which you create new controls is important. Create your wxStaticBox control **before** any siblings that are to appear inside the wxStaticBox in order to preserve the correct Z-Order of controls.

### Derived from

*wxControl* (p. 285)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### Include files

<wx/statbox.h>

### Window styles

There are no special styles for this control.

See also *window styles overview* (p. 2089).

### See also

*wxStaticText* (p. 1534)

## wxStaticBox::wxStaticBox

**wxStaticBox()**

Default constructor.

```
wxStaticBox(wxWindow* parent, wxWindowID id, const wxString& label, const  
wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0,  
const wxString& name = "staticBox")
```

Constructor, creating and showing a static box.

### Parameters

*parent*

Parent window. Must not be NULL.

*id*

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

*label*

Text to be displayed in the static box, the empty string for no label.

*pos*

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

*size*

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

*style*

Window style. See *wxStaticBox* (p. 1530).

*name*

Window name.

### See also

*wxStaticBox::Create* (p. 1531)

### **wxStaticBox::~wxStaticBox**

```
void ~wxStaticBox()
```

Destructor, destroying the group box.

### **wxStaticBox::Create**

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

Creates the static box for two-step construction. See *wxStaticBox::wxStaticBox* (p. 1530)

for further details.

## **wxStaticBoxSizer**

`wxStaticBoxSizer` is a sizer derived from `wxBoxSizer` but adds a static box around the sizer. This static box may be either created independently or the sizer may create it itself as a convenience. In any case, the sizer owns the `wxStaticBox` (p. 1530) control and will delete it if it is deleted.

### **Derived from**

`wxBoxSizer` (p. 149)  
`wxSizer` (p. 1444)  
`wxObject` (p. 1148)

### **Include files**

<wx/sizer.h>

### **See also**

`wxSizer` (p. 1444), `wxStaticBox` (p. 1530), `wxBoxSizer` (p. 149), *Sizer overview* (p. 2098)

## **wxStaticBoxSizer::wxStaticBoxSizer**

**wxStaticBoxSizer**(**wxStaticBox\*** box, **int** orient)

**wxStaticBoxSizer**(**int** orient, **wxWindow** \*parent, **const wxString&** label = `wxEmptyString`)

The first constructor uses an already existing static box. It takes the associated static box and the orientation *orient*, which can be either `wxVERTICAL` or `wxHORIZONTAL` as parameters.

The second one creates a new static box with the given label and parent window.

## **wxStaticBoxSizer::GetStaticBox**

**wxStaticBox\*** GetStaticBox()

Returns the static box associated with the sizer.

## **wxStaticLine**

A static line is just a line which may be used in a dialog to separate the groups of controls. The line may be only vertical or horizontal.

### **Derived from**

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/statline.h>

**Window styles**

**wxLI\_HORIZONTAL**           Creates a horizontal line.

**wxLI\_VERTICAL**           Creates a vertical line.

**See also**

*wxStaticBox* (p. 1530)

**wxStaticLine::wxStaticLine****wxStaticLine()**

Default constructor.

**wxStaticLine**(**wxWindow\*** *parent*, **wxWindowID** *id* = *wxID\_ANY*, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = *wxLI\_HORIZONTAL*, **const wxString&** *name* = "staticLine")

Constructor, creating and showing a static line.

**Parameters***parent*

Parent window. Must not be NULL.

*id*

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

*pos*

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

*size*

Size. Note that either the height or the width (depending on whether the line is horizontal or vertical) is ignored.

*style*

Window style (either *wxLI\_HORIZONTAL* or *wxLI\_VERTICAL*).

*name*

Window name.

### See also

*wxStaticLine::Create* (p. 1534)

### **wxStaticLine::Create**

**bool Create**(*wxWindow\* parent*, **wxWindowID** *id* = *wxID\_ANY*, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **long** *style* = 0, **const wxString&** *name* = "staticLine")

Creates the static line for two-step construction. See *wxStaticLine::wxStaticLine* (p. 1533) for further details.

### **wxStaticLine::IsVertical**

**bool IsVertical()** **const**

Returns true if the line is vertical, false if horizontal.

### **wxStaticLine::GetDefaultSize**

**int GetDefaultSize()**

This static function returns the size which will be given to the smaller dimension of the static line, i.e. its height for a horizontal line or its width for a vertical one.

## **wxStaticText**

A static text control displays one or more lines of read-only text.

### Derived from

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/stattext.h>

### Window styles

<b>wxALIGN_LEFT</b>	Align the text to the left
<b>wxALIGN_RIGHT</b>	Align the text to the right
<b>wxALIGN_CENTRE</b>	Center the text (horizontally)

**wxST\_NO\_AUTORESIZE** By default, the control will adjust its size to exactly fit to the size of the text when *SetLabel* (p. 1536) is called. If this style flag is given, the control will not change its size (this style is especially useful with controls which also have *wxALIGN\_RIGHT* or *CENTER* style because otherwise they won't make sense any longer after a call to *SetLabel*)

See also *window styles overview* (p. 2089).

#### See also

*wxStaticBitmap* (p. 1527), *wxStaticBox* (p. 1530)

### **wxStaticText::wxStaticText**

#### **wxStaticText()**

Default constructor.

**wxStaticText(wxWindow\* parent, wxWindowID id, const wxString& label, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0, const wxString& name = "staticText")**

Constructor, creating and showing a text control.

#### Parameters

*parent*

Parent window. Should not be NULL.

*id*

Control identifier. A value of -1 denotes a default value.

*label*

Text label.

*pos*

Window position.

*size*

Window size.

*style*

Window style. See *wxStaticText* (p. 1534).

*name*

Window name.

### See also

*wxStaticText::Create* (p. 1536)

### **wxStaticText::Create**

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

Creation function, for two-step construction. For details see *wxStaticText::wxStaticText* (p. 1535).

### **wxStaticText::GetLabel**

**wxString GetLabel()** **const**

Returns the contents of the control.

### **wxStaticText::SetLabel**

**virtual void SetLabel**(**const wxString&** label)

Sets the static text label and updates the controls size to exactly fit the label unless the control has *wxST\_NO\_AUTORESIZE* flag.

### Parameters

*label*

The new label to set. It may contain newline characters.

### **wxStaticText::Wrap**

**void Wrap**(*int width*)

This functions wraps the controls label so that each of its lines becomes at most *width* pixels wide if possible (the lines are broken at words boundaries so it might not be the case if words are too long). If *width* is negative, no wrapping is done.

This function is new since *wxWidgets* version 2.6.2

## **wxStatusBar**

A status bar is a narrow window that can be placed along the bottom of a frame to give small amounts of status information. It can contain one or more fields, one or more of which can be variable length according to the size of the window.

*wxWindow* (p. 1795)



*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### Derived from

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### Include files

<wx/statusbr.h>

### Window styles

**wxST\_SIZEGRIP**                      On Windows 95, displays a gripper at right-hand side of the status bar.

See also *window styles overview* (p. 2089).

### Remarks

It is possible to create controls and other windows on the status bar. Position these windows from an **OnSize** event handler.

### See also

*wxFrame* (p. 682), *Status bar sample* (p. 2038)

## **wxStatusBar::wxStatusBar**

### **wxStatusBar()**

Default constructor.

**wxStatusBar**(*wxWindow\** parent, *wxWindowID* id = *wxID\_ANY*, *long* style = *wxST\_SIZEGRIP*, *const wxString&* name = "statusBar")

Constructor, creating the window.

### Parameters

*parent*

The window parent, usually a frame.

*id*

The window identifier. It may take a value of -1 to indicate a default value.

*style*

The window style. See *wxStatusBar* (p. 1536).

*name*

The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

**See also**

*wxStatusBar::Create* (p. 1538)

**wxStatusBar::~~wxStatusBar**

**void ~wxStatusBar()**

Destructor.

**wxStatusBar::Create**

**bool Create**(*wxWindow\* parent*, **wxWindowID** *id* = *wxID\_ANY*, **long** *style* = *wxST\_SIZEGRIP*, **const wxString&** *name* = "statusBar")

Creates the window, for two-step construction.

See *wxStatusBar::wxStatusBar* (p. 1537) for details.

**wxStatusBar::GetFieldRect**

**virtual bool GetFieldRect**(**int** *i*, **wxRect&** *rect*) **const**

Returns the size and position of a field's internal bounding rectangle.

**Parameters**

*i*

The field in question.

*rect*

The rectangle values are placed in this variable.

**Return value**

true if the field index is valid, false otherwise.

**See also**

*wxRect* (p. 1252)

**wxPerl note:** In wxPerl this function returns a `Wx::Rect` if the field index is valid, `undef` otherwise.

**wxStatusBar::GetFieldsCount**

**int GetFieldsCount() const**

Returns the number of fields in the status bar.

**wxStatusBar::GetStatusText****virtual wxString GetStatusText(int *i* = 0) const**

Returns the string associated with a status bar field.

**Parameters**

*i*

The number of the status field to retrieve, starting from zero.

**Return value**

The status field string if the field is valid, otherwise the empty string.

**See also**

*wxStatusBar::SetStatusText* (p. 1540)

**wxStatusBar::PopStatusText****void PopStatusText(int *field* = 0)**

Sets the field text to the top of the stack, and pops the stack of saved strings.

**See also**

*wxStatusBar::PushStatusText* (p. 1539)

**wxStatusBar::PushStatusText****void PushStatusText(const wxString& *string*, int *field* = 0)**

Saves the current field text in a per field stack, and sets the field text to the string passed as argument.

**wxStatusBar::SetFieldsCount****virtual void SetFieldsCount(int *number* = 1, int\* *widths* = NULL)**

Sets the number of fields, and optionally the field widths.

**wxPython note:** Only the first parameter is accepted. Use *SetStatusWidths* to set the widths of the fields.

**wxPerl note:** In wxPerl this function accepts only the *n* parameter. Use *SetStatusWidths* to set the field widths.

**Parameters***number*

The number of fields.

*widths*

An array of *n* integers interpreted in the same way as in *SetStatusWidths* (p. 1540)

**wxStatusBar::SetMinHeight****void SetMinHeight(int height)**

Sets the minimal possible height for the status bar. The real height may be bigger than the height specified here depending on the size of the font used by the status bar.

**wxStatusBar::SetStatusText****virtual void SetStatusText(const wxString& text, int i = 0)**

Sets the text for one field.

**Parameters***text*

The text to be set. Use an empty string ("" ) to clear the field.

*i*

The field to set, starting from zero.

**See also**

*wxStatusBar::GetStatusText* (p. 1539), *wxFrame::SetStatusText* (p. 691)

**wxStatusBar::SetStatusWidths****virtual void SetStatusWidths(int n, int \*widths)**

Sets the widths of the fields in the status line. There are two types of fields: fixed widths one and variable width fields. For the fixed width fields you should specify their (constant) width in pixels. For the variable width fields, specify a negative number which indicates how the field should expand: the space left for all variable width fields is divided between them according to the absolute value of this number. A variable width field with width of -2 gets twice as much of it as a field with width -1 and so on.

For example, to create one fixed width field of width 100 in the right part of the status bar and two more fields which get 66% and 33% of the remaining space correspondingly, you should use an array containing -2, -1 and 100.

**Parameters**

*n*

The number of fields in the status bar. Must be equal to the number passed to *SetFieldsCount* (p. 1539) the last time it was called.

*widths*

Contains an array of *n* integers, each of which is either an absolute status field width in pixels if positive or indicates a variable width field if negative.

### Remarks

The widths of the variable fields are calculated from the total width of all fields, minus the sum of widths of the non-variable fields, divided by the number of variable fields.

### See also

*wxStatusBar::SetFieldsCount* (p. 1539), *wxFrame::SetStatusWidths* (p. 691)

**wxPython note:** Only a single parameter is required, a Python list of integers.

**wxPerl note:** In wxPerl this method takes as parameters the field widths.

## **wxStatusBar::SetStatusStyles**

**virtual void SetStatusStyles(int *n*, int \**styles*)**

Sets the styles of the fields in the status line which can make fields appear flat or raised instead of the standard sunken 3D border.

### Parameters

*n*

The number of fields in the status bar. Must be equal to the number passed to *SetFieldsCount* (p. 1539) the last time it was called.

*styles*

Contains an array of *n* integers with the styles for each field. There are three possible styles:

<b>wxSB_NORMAL</b>	(default) The field appears sunken with a standard 3D border.
<b>wxSB_FLAT</b>	No border is painted around the field so that it appears flat.
<b>wxSB_RAISED</b>	A raised 3D border is painted around the field.

## **wxStdDialogButtonSizer**

This class creates button layouts which conform to the standard button spacing and ordering defined by the platform or toolkit's user interface guidelines (if such things exist). By using this class, you can ensure that all your standard dialogs look correct on all major platforms. Currently it conforms to the Windows, GTK+ and Mac OS X human interface guidelines.

When there aren't interface guidelines defined for a particular platform or toolkit, `wxStdDialogButtonSizer` reverts to the Windows implementation.

To use this class, first add buttons to the sizer by calling `AddButton` (or `SetAffirmativeButton`, `SetNegativeButton`, or `SetCancelButton`) and then call `Realize` in order to create the actual button layout used. Other than these special operations, this sizer works like any other sizer.

If you add a button with `wxID_SAVE`, on Mac OS X the button will be renamed to "Save" and the `wxID_NO` button will be renamed to "Don't Save" in accordance with the Mac OS X Human Interface Guidelines.

### Derived from

*wxBoxSizer* (p. 149)  
*wxSizer* (p. 1444)  
*wxObject* (p. 1148)

### Include files

<wx/sizer.h>

### See also

*wxSizer* (p. 1444), *Sizer overview* (p. 2098), *wxDialog::CreateButtonSizer* (p. 499)

## **wxStdDialogButtonSizer::wxStdDialogButtonSizer**

### **wxStdDialogButtonSizer()**

Constructor for a `wxStdDialogButtonSizer`.

## **wxStdDialogButtonSizer::AddButton**

### **void AddButton(wxButton\* button)**

Adds a button to the `wxStdDialogButtonSizer`. The button must have one of the following identifiers:

- `wxID_OK`
- `wxID_YES`
- `wxID_SAVE`
- `wxID_APPLY`

- `wxID_NO`
- `wxID_CANCEL`
- `wxID_HELP`
- `wxID_CONTEXT_HELP`

### **`wxStdDialogButtonSizer::Realize`**

#### **`void Realize()`**

Rearranges the buttons and applies proper spacing between buttons to make them match the platform or toolkit's interface guidelines.

### **`wxStdDialogButtonSizer::SetAffirmativeButton`**

#### **`void SetAffirmativeButton(wxButton* button)`**

Sets the affirmative button for the sizer. This allows you to use identifiers other than the standard identifiers outlined above.

### **`wxStdDialogButtonSizer::SetCancelButton`**

#### **`void SetCancelButton(wxButton* button)`**

Sets the cancel button for the sizer. This allows you to use identifiers other than the standard identifiers outlined above.

### **`wxStdDialogButtonSizer::SetNegativeButton`**

#### **`void SetNegativeButton(wxButton* button)`**

Sets the negative button for the sizer. This allows you to use identifiers other than the standard identifiers outlined above.

## **`wxStopWatch`**

The `wxStopWatch` class allow you to measure time intervals. For example, you may use it to measure the time elapsed by some function:

```
wxStopWatch sw;
CallLongRunningFunction();
wxLogMessage("The long running function took %ldms to execute",
             sw.Time());
sw.Pause();
... stopwatch is stopped now ...
sw.Resume();
CallLongRunningFunction();
wxLogMessage("And calling it twice took $ldms in all", sw.Time());
```

**Include files**

<wx/stopwatch.h>

**See also**

*::wxStartTimer* (p. 1979), *::wxGetElapsedTime* (p. 1977), *wxTimer* (p. 1680)

**wxStopWatch::wxStopWatch****wxStopWatch()**

Constructor. This starts the stop watch.

**wxStopWatch::Pause****void Pause()**

Pauses the stop watch. Call *wxStopWatch::Resume* (p. 1544) to resume time measuring again.

If this method is called several times, *Resume* ( ) must be called the same number of times to really resume the stop watch. You may, however, call *Start* (p. 1544) to resume it unconditionally.

**wxStopWatch::Resume****void Resume()**

Resumes the stop watch which had been paused with *wxStopWatch::Pause* (p. 1544).

**wxStopWatch::Start****void Start(long milliseconds = 0)**

(Re)starts the stop watch with a given initial value.

**wxStopWatch::Time****long Time() const**

Returns the time in milliseconds since the start (or restart) or the last call of *wxStopWatch::Pause* (p. 1544).

**wxStreamBase**

This class is the base class of most stream related classes in wxWidgets. It must not be used directly.



**Derived from**

None

**Include files**

<wx/stream.h>

**See also**

*wxStreamBuffer* (p. 1547)

**wxStreamBase::wxStreamBase**

**wxStreamBase()**

Creates a dummy stream object. It doesn't do anything.

**wxStreamBase::~~wxStreamBase**

**~wxStreamBase()**

Destructor.

**wxStreamBase::GetLength**

**wxFileOffset GetLength() const**

Returns the length of the stream in bytes. If the length cannot be determined (this is always the case for socket streams for example), returns `wxInvalidOffset`.

This function is new since wxWidgets version 2.5.4

**wxStreamBase::GetLastError**

**wxStreamError GetLastError() const**

This function returns the last error.

**wxSTREAM\_NO\_ERROR**    No error occurred.

**wxSTREAM\_EOF**            An End-Of-File occurred.

**wxSTREAM\_WRITE\_ERROR**    A generic error occurred on the last write call.

**wxSTREAM\_READ\_ERROR**    A generic error occurred on the last read call.

**wxStreamBase::GetSize**

**size\_t GetSize() const**

*GetLength* (p. 1545)

This function returns the size of the stream. For example, for a file it is the size of the file.

### **Warning**

There are streams which do not have size by definition, such as socket streams. In that cases, *GetSize* returns 0 so you should always test its return value.

### **wxStreamBase::IsOk**

**virtual bool IsOk() const**

Returns true if no error occurred on the stream.

### **See also**

*GetLastError* (p. 1545)

### **wxStreamBase::IsSeekable**

**bool IsSeekable() const**

Returns true if the streams supports seeking to arbitrary offsets.

### **wxStreamBase::OnSysRead**

**size\_t OnSysRead(void\* buffer, size\_t bufsize)**

Internal function. It is called when the stream wants to read data of the specified size. It should return the size that was actually read.

### **wxStreamBase::OnSysSeek**

**off\_t OnSysSeek(off\_t pos, wxSeekMode mode)**

Internal function. It is called when the stream needs to change the current position.

### **wxStreamBase::OnSysTell**

**off\_t OnSysTell() const**

Internal function. Is is called when the stream needs to know the real position.

### **wxStreamBase::OnSysWrite**

**size\_t OnSysWrite(const void \*buffer, size\_t bufsize)**

See *OnSysRead* (p. 1546).

## wxStreamBuffer

### Derived from

None

### Include files

<wx/stream.h>

### See also

*wxStreamBase* (p. 1544)

## wxStreamBuffer::wxStreamBuffer

### **wxStreamBuffer(wxStreamBase& stream, BufMode mode)**

Constructor, creates a new stream buffer using *stream* as a parent stream and *mode* as the IO mode. *mode* can be: *wxStreamBuffer::read*, *wxStreamBuffer::write*, *wxStreamBuffer::read\_write*.

One stream can have many stream buffers but only one is used internally to pass IO call (e.g. *wxInputStream::Read()* -> *wxStreamBuffer::Read()*), but you can call directly *wxStreamBuffer::Read* without any problems. Note that all errors and messages linked to the stream are stored in the stream, not the stream buffers:

```
streambuffer.Read(...);
streambuffer2.Read(...); /* This call erases previous error
messages set by
                        ``streambuffer'' */
```

### **wxStreamBuffer(BufMode mode)**

Constructor, creates a new empty stream buffer which won't flush any data to a stream. *mode* specifies the type of the buffer (read, write, read\_write). This stream buffer has the advantage to be stream independent and to work only on memory buffers but it is still compatible with the rest of the *wxStream* classes. You can write, read to this special stream and it will grow (if it is allowed by the user) its internal buffer. Briefly, it has all functionality of a "normal" stream.

### Warning

The "read\_write" mode doesn't currently work for standalone stream buffers.

### **wxStreamBuffer(const wxStreamBuffer&buffer)**

Constructor. It initializes the stream buffer with the data of the specified stream buffer. The new stream buffer has the same attributes, size, position and they share the same buffer. This will cause problems if the stream to which the stream buffer belong is destroyed and the newly cloned stream buffer continues to be used, trying to call functions in the

(destroyed) stream. It is advised to use this feature only in very local area of the program.

**See also**

*wxStreamBuffer::SetBufferIO* (p. 1550)

**wxStreamBuffer::~~wxStreamBuffer**

**wxStreamBuffer**(~wxStreamBuffer)

Destructor. It finalizes all IO calls and frees all internal buffers if necessary.

**wxStreamBuffer::Read**

**size\_t** Read(void \**buffer*, **size\_t** *size*)

Reads a block of the specified *size* and stores the data in *buffer*. This function tries to read from the buffer first and if more data has been requested, reads more data from the associated stream and updates the buffer accordingly until all requested data is read.

**Return value**

It returns the size of the data read. If the returned size is different of the specified *size*, an error has occurred and should be tested using *GetLastError* (p. 1545).

**size\_t** Read(wxStreamBuffer \**buffer*)

Copies data to *buffer*. The function returns when *buffer* is full or when there isn't any more data in the current buffer.

**See also**

*wxStreamBuffer::Write* (p. 1548)

**wxStreamBuffer::Write**

**size\_t** Write(const void \**buffer*, **size\_t** *size*)

Writes a block of the specified *size* using data of *buffer*. The data are cached in a buffer before being sent in one block to the stream.

**size\_t** Write(wxStreamBuffer \**buffer*)

See *Read* (p. 1548).

**wxStreamBuffer::GetChar**

**char** GetChar()

Gets a single char from the stream buffer. It acts like the *Read* call.

**Problem**

You aren't directly notified if an error occurred during the IO call.

**See also**

*wxStreamBuffer::Read* (p. 1548)

**wxStreamBuffer::PutChar**

**void PutChar(char c)**

Puts a single char to the stream buffer.

**Problem**

You aren't directly notified if an error occurred during the IO call.

**See also**

*wxStreamBuffer::Read* (p. 1548)

**wxStreamBuffer::Tell**

**off\_t Tell() const**

Gets the current position in the stream. This position is calculated from the *real* position in the stream and from the internal buffer position: so it gives you the position in the *real* stream counted from the start of the stream.

**Return value**

Returns the current position in the stream if possible, `wxInvalidOffset` in the other case.

**wxStreamBuffer::Seek**

**off\_t Seek(off\_t pos, wxSeekMode mode)**

Changes the current position.

*mode* may be one of the following:

<b>wxFromStart</b>	The position is counted from the start of the stream.
<b>wxFromCurrent</b>	The position is counted from the current position of the stream.
<b>wxFromEnd</b>	The position is counted from the end of the stream.

**Return value**

Upon successful completion, it returns the new offset as measured in bytes from the beginning of the stream. Otherwise, it returns `wxInvalidOffset`.

**wxStreamBuffer::ResetBuffer**

**void ResetBuffer()**

Resets to the initial state variables concerning the buffer.

**wxStreamBuffer::SetBufferIO****void SetBufferIO(char\* buffer\_start, char\* buffer\_end)**

Specifies which pointers to use for stream buffering. You need to pass a pointer on the start of the buffer end and another on the end. The object will use this buffer to cache stream data. It may be used also as a source/destination buffer when you create an empty stream buffer (See *wxStreamBuffer::wxStreamBuffer* (p. 1547)).

**Remarks**

When you use this function, you will have to destroy the IO buffers yourself after the stream buffer is destroyed or don't use it anymore. In the case you use it with an empty buffer, the stream buffer will not resize it when it is full.

**See also**

*wxStreamBuffer* constructor (p. 1547)

*wxStreamBuffer::Fixed* (p. 1551)

*wxStreamBuffer::Flushable* (p. 1551)

**void SetBufferIO(size\_t bufsize)**

Destroys or invalidates the previous IO buffer and allocates a new one of the specified size.

**Warning**

All previous pointers aren't valid anymore.

**Remark**

The created IO buffer is growable by the object.

**See also**

*wxStreamBuffer::Fixed* (p. 1551)

*wxStreamBuffer::Flushable* (p. 1551)

**wxStreamBuffer::GetBufferStart****void \* GetBufferStart() const**

Returns a pointer on the start of the stream buffer.

**wxStreamBuffer::GetBufferEnd****void \* GetBufferEnd() const**

Returns a pointer on the end of the stream buffer.

### **wxStreamBuffer::GetBufferPos**

**void \* GetBufferPos() const**

Returns a pointer on the current position of the stream buffer.

### **wxStreamBuffer::GetIntPosition**

**off\_t GetIntPosition() const**

Returns the current position (counted in bytes) in the stream buffer.

### **wxStreamBuffer::SetIntPosition**

**void SetIntPosition(size\_t pos)**

Sets the current position (in bytes) in the stream buffer.

### **Warning**

Since it is a very low-level function, there is no check on the position: specifying an invalid position can induce unexpected results.

### **wxStreamBuffer::GetLastAccess**

**size\_t GetLastAccess() const**

Returns the amount of bytes read during the last IO call to the parent stream.

### **wxStreamBuffer::Fixed**

**void Fixed(bool fixed)**

Toggles the fixed flag. Usually this flag is toggled at the same time as *flushable*. This flag allows (when it has the false value) or forbids (when it has the true value) the stream buffer to resize dynamically the IO buffer.

### **See also**

*wxStreamBuffer::SetBufferIO* (p. 1550)

### **wxStreamBuffer::Flushable**

**void Flushable(bool flushable)**

Toggles the flushable flag. If *flushable* is disabled, no data are sent to the parent stream.

### **wxStreamBuffer::FlushBuffer**

**bool FlushBuffer()**

Flushes the IO buffer.

**wxStreamBuffer::FillBuffer****bool FillBuffer()**

Fill the IO buffer.

**wxStreamBuffer::GetDataLeft****size\_t GetDataLeft()**

Returns the amount of available data in the buffer.

**wxStreamBuffer::Stream****wxStreamBase\* Stream()**

Returns the parent stream of the stream buffer.

**wxStreamToTextRedirector**

This class can be used to (temporarily) redirect all output sent to a C++ ostream object to a *wxTextCtrl* (p. 1633) instead.

**NB:** Some compilers and/or build configurations don't support multiply inheriting *wxTextCtrl* (p. 1633) from `std::streambuf` in which case this class is not compiled in. You also must have `wxUSE_STD_Iostream` option on (i.e. set to 1) in your `setup.h` to be able to use it. Under Unix, specify `--enable-std_iostreams` switch when running configure for this.

Example of usage: `using namespace std;`

```
wxTextCtrl *text = new wxTextCtrl(...);

{
    wxStreamToTextRedirector redirect(text);

    // this goes to the text control
    cout << "Hello, text!" << endl;
}

// this goes somewhere else, presumably to stdout
cout << "Hello, console!" << endl;
```

**Derived from**

No base class

**Include files**



<wx/textctrl.h>

### See also

*wxTextCtrl* (p. 1633)

## **wxStreamToTextRedirector::wxStreamToTextRedirector**

**wxStreamToTextRedirector**(*wxTextCtrl* \**text*, *ostream* \**ostr* = *NULL*)

The constructor starts redirecting output sent to *ostr* or *cout* for the default parameter value to the text control *text*.

### Parameters

*text*

The text control to append output too, must be non-NULL

*ostr*

The C++ stream to redirect, *cout* is used if it is NULL

## **wxStreamToTextRedirector::~~wxStreamToTextRedirector**

**~wxStreamToTextRedirector**()

When a *wxStreamToTextRedirector* object is destroyed, the redirection is ended and any output sent to the C++ ostream which had been specified at the time of the object construction will go to its original destination.

## **wxString**

*wxString* is a class representing a character string. Please see the *wxString overview* (p. 2047) for more information about it.

As explained there, *wxString* implements most of the methods of the *std::string* class. These standard functions are not documented in this manual, please see the STL documentation (<http://www.cppreference.com/cppstl.html>). The behaviour of all these functions is identical to the behaviour described there.

You may notice that *wxString* sometimes has many functions which do the same thing like, for example, *Length()* (p. 1569), *Len()* (p. 1569) and *length()* which all return the string length. In all cases of such duplication the *std::string*-compatible method (*length()* in this case, always the lowercase version) should be used as it will ensure smoother transition to *std::string* when *wxWidgets* starts using it instead of *wxString*.

### Derived from

None

**Include files**

<wx/string.h>

**Predefined objects**

Objects:

**wxEmptyString****See also**

*wxString* overview (p. 2047), *Unicode* overview (p. 2056)

**Constructors and assignment operators**

A string may be constructed either from a C string, (some number of copies of) a single character or a wide (UNICODE) string. For all constructors (except the default which creates an empty string) there is also a corresponding assignment operator.

*wxString* (p. 1560)  
*operator =* (p. 1577)  
*~wxString* (p. 1561)

**String length**

These functions return the string length and check whether the string is empty or empty it.

*Len* (p. 1569)  
*IsEmpty* (p. 1568)  
*operator!* (p. 1576)  
*Empty* (p. 1564)  
*Clear* (p. 1563)

**Character access**

Many functions in this section take a character index in the string. As with C strings and/or arrays, the indices start from 0, so the first character of a string is `string[0]`. Attempt to access a character beyond the end of the string (which may be even 0 if the string is empty) will provoke an assert failure in *debug build* (p. 2072), but no checks are done in release builds.

This section also contains both implicit and explicit conversions to C style strings. Although implicit conversion is quite convenient, it is advised to use explicit *c\_str()* (p. 1563) method for the sake of clarity. Also see *overview* (p. 2049) for the cases where it is necessary to use it.

*GetChar* (p. 1567)  
*GetWritableChar* (p. 1567)  
*SetChar* (p. 1572)

*Last* (p. 1569)  
*operator []* (p. 1577)  
*c\_str* (p. 1563)  
*mb\_str* (p. 1570)  
*wc\_str* (p. 1576)  
*fn\_str* (p. 1565)  
*operator const char\** (p. 1578)

## Concatenation

Anything may be concatenated (appended to) with a string. However, you can't append something to a C string (including literal constants), so to do this it should be converted to a `wxString` first.

*operator <<* (p. 1578)  
*operator +=* (p. 1577)  
*operator +* (p. 1577)  
*Append* (p. 1562)  
*Prepend* (p. 1570)

## Comparison

The default comparison function *Cmp* (p. 1564) is case-sensitive and so is the default version of *IsSameAs* (p. 1568). For case insensitive comparisons you should use *CmpNoCase* (p. 1564) or give a second parameter to *IsSameAs*. This last function is may be more convenient if only equality of the strings matters because it returns a boolean `true` value if the strings are the same and not 0 (which is usually false in C) as *Cmp( )* does.

*Matches* (p. 1570) is a poor man's regular expression matcher: it only understands '\*' and '?' metacharacters in the sense of DOS command line interpreter.

*StartsWith* (p. 1572) is helpful when parsing a line of text which should start with some predefined prefix and is more efficient than doing direct string comparison as you would also have to precalculate the length of the prefix then.

*Cmp* (p. 1564)  
*CmpNoCase* (p. 1564)  
*IsSameAs* (p. 1568)  
*Matches* (p. 1570)  
*StartsWith* (p. 1572)  
*EndsWith* (p. 1572)

## Substring extraction

These functions allow to extract substring from this string. All of them don't modify the original string and return a new string containing the extracted substring.

*Mid* (p. 1570)  
*operator()* (p. 1578)  
*Left* (p. 1569)

*Right* (p. 1572)  
*BeforeFirst* (p. 1562)  
*BeforeLast* (p. 1563)  
*AfterFirst* (p. 1562)  
*AfterLast* (p. 1562)  
*StartsWith* (p. 1572)  
*EndsWith* (p. 1572)

### Case conversion

The *MakeXXX()* variants modify the string in place, while the other functions return a new string which contains the original text converted to the upper or lower case and leave the original string unchanged.

*MakeUpper* (p. 1570)  
*Upper* (p. 1575)  
*MakeLower* (p. 1569)  
*Lower* (p. 1569)

### Searching and replacing

These functions replace the standard *strchr()* and *strstr()* functions.

*Find* (p. 1565)  
*Replace* (p. 1571)

### Conversion to numbers

The string provides functions for conversion to signed and unsigned integer and floating point numbers. All three functions take a pointer to the variable to put the numeric value in and return `true` if the **entire** string could be converted to a number.

*ToLong* (p. 1574)  
*ToLongLong* (p. 1574)  
*ToULong* (p. 1574)  
*ToULongLong* (p. 1575)  
*ToDouble* (p. 1573)

### Writing values into the string

Both formatted versions (*Printf* (p. 1570)) and stream-like insertion operators exist (for basic types only). Additionally, the *Format* (p. 1565) function allows to use simply append formatted value to a string:

```
// the following 2 snippets are equivalent

wxString s = "...";
s += wxString::Format("%d", n);

wxString s;
s.Printf("...%d", n);
```

*Format* (p. 1565)  
*FormatV* (p. 1566)  
*Printf* (p. 1570)  
*PrintfV* (p. 1571)  
*operator <<* (p. 1578)

## Memory management

These are "advanced" functions and they will be needed quite rarely. *Alloc* (p. 1561) and *Shrink* (p. 1572) are only interesting for optimization purposes. *GetWriteBuf* (p. 1567) may be very useful when working with some external API which requires the caller to provide a writable buffer, but extreme care should be taken when using it: before performing any other operation on the string *UngetWriteBuf* (p. 1575) **must** be called!

*Alloc* (p. 1561)  
*Shrink* (p. 1572)  
*GetWriteBuf* (p. 1567)  
*UngetWriteBuf* (p. 1575)

## Miscellaneous

Other string functions.

*Trim* (p. 1575)  
*Truncate* (p. 1575)  
*Pad* (p. 1570)

## wxWidgets 1.xx compatibility functions

These functions are deprecated, please consider using new wxWidgets 2.0 functions instead of them (or, even better, std::string compatible variants).

*CompareTo* (p. 1564)  
*Contains* (p. 1564)  
*First* (p. 1565)  
*Freq* (p. 1566)  
*Index* (p. 1567)  
*IsAscii* (p. 1567)  
*IsNull* (p. 1568)  
*IsNumber* (p. 1568)  
*IsWord* (p. 1568)  
*Last* (p. 1569)  
*Length* (p. 1569)  
*LowerCase* (p. 1569)  
*Remove* (p. 1571)  
*Strip* (p. 1572)  
*SubString* (p. 1573)  
*UpperCase* (p. 1576)

## std::string compatibility functions

The supported functions are only listed here, please see any STL reference for their documentation.

```
// take nLen chars starting at nPos
wxString(const wxString& str, size_t nPos, size_t nLen);
// take all characters from pStart to pEnd (poor man's iterators)
wxString(const void *pStart, const void *pEnd);

// lib.string.capacity
// return the length of the string
size_t size() const;
// return the length of the string
size_t length() const;
// return the maximum size of the string
size_t max_size() const;
// resize the string, filling the space with c if c != 0
void resize(size_t nSize, char ch = '\0');
// delete the contents of the string
void clear();
// returns true if the string is empty
bool empty() const;

// lib.string.access
// return the character at position n
char at(size_t n) const;
// returns the writable character at position n
char& at(size_t n);

// lib.string.modifiers
// append a string
wxString& append(const wxString& str);
// append elements str[pos], ..., str[pos+n]
wxString& append(const wxString& str, size_t pos, size_t n);
// append first n (or all if n == npos) characters of sz
wxString& append(const char *sz, size_t n = npos);

// append n copies of ch
wxString& append(size_t n, char ch);

// same as `this_string = str'
wxString& assign(const wxString& str);
// same as ` = str[pos..pos + n]
wxString& assign(const wxString& str, size_t pos, size_t n);
// same as ` = first n (or all if n == npos) characters of sz'
wxString& assign(const char *sz, size_t n = npos);
// same as ` = n copies of ch'
wxString& assign(size_t n, char ch);

// insert another string
wxString& insert(size_t nPos, const wxString& str);
// insert n chars of str starting at nStart (in str)
wxString& insert(size_t nPos, const wxString& str, size_t nStart,
size_t n);
```

```
// insert first n (or all if n == npos) characters of sz
wxString& insert(size_t nPos, const char *sz, size_t n = npos);
// insert n copies of ch
wxString& insert(size_t nPos, size_t n, char ch);

// delete characters from nStart to nStart + nLen
wxString& erase(size_t nStart = 0, size_t nLen = npos);

// replaces the substring of length nLen starting at nStart
wxString& replace(size_t nStart, size_t nLen, const char* sz);
// replaces the substring with nCount copies of ch
wxString& replace(size_t nStart, size_t nLen, size_t nCount, char
ch);
// replaces a substring with another substring
wxString& replace(size_t nStart, size_t nLen,
                  const wxString& str, size_t nStart2, size_t
nLen2);
// replaces the substring with first nCount chars of sz
wxString& replace(size_t nStart, size_t nLen,
                  const char* sz, size_t nCount);

// swap two strings
void swap(wxString& str);

// All find() functions take the nStart argument which specifies
the
// position to start the search on, the default value is 0. All
functions
// return npos if there were no match.

// find a substring
size_t find(const wxString& str, size_t nStart = 0) const;

// find first n characters of sz
size_t find(const char* sz, size_t nStart = 0, size_t n = npos) const;

// find the first occurrence of character ch after nStart
size_t find(char ch, size_t nStart = 0) const;

// rfind() family is exactly like find() but works right to left

// as find, but from the end
size_t rfind(const wxString& str, size_t nStart = npos) const;

// as find, but from the end
size_t rfind(const char* sz, size_t nStart = npos,
             size_t n = npos) const;
// as find, but from the end
size_t rfind(char ch, size_t nStart = npos) const;

// find first/last occurrence of any character in the set

//
size_t find_first_of(const wxString& str, size_t nStart = 0) const;
//
size_t find_first_of(const char* sz, size_t nStart = 0) const;
// same as find(char, size_t)
```

```
size_t find_first_of(char c, size_t nStart = 0) const;
//
size_t find_last_of (const wxString& str, size_t nStart = npos)
const;
//
size_t find_last_of (const char* s, size_t nStart = npos) const;
// same as rfind(char, size_t)
size_t find_last_of (char c, size_t nStart = npos) const;

// find first/last occurrence of any character not in the set

//
size_t find_first_not_of(const wxString& str, size_t nStart = 0)
const;
//
size_t find_first_not_of(const char* s, size_t nStart = 0) const;
//
size_t find_first_not_of(char ch, size_t nStart = 0) const;
//
size_t find_last_not_of(const wxString& str, size_t nStart=npos)
const;
//
size_t find_last_not_of(const char* s, size_t nStart = npos) const;
//
size_t find_last_not_of(char ch, size_t nStart = npos) const;

// All compare functions return a negative, zero or positive value
// if the [sub]string is less, equal or greater than the compare()
argument.

// just like strcmp()
int compare(const wxString& str) const;
// comparison with a substring
int compare(size_t nStart, size_t nLen, const wxString& str) const;
// comparison of 2 substrings
int compare(size_t nStart, size_t nLen,
            const wxString& str, size_t nStart2, size_t nLen2)
const;
// just like strcmp()
int compare(const char* sz) const;
// substring comparison with first nCount characters of sz
int compare(size_t nStart, size_t nLen,
            const char* sz, size_t nCount = npos) const;

// substring extraction
wxString substr(size_t nStart = 0, size_t nLen = npos) const;
```

## **wxString::wxString**

### **wxString()**

Default constructor. Initializes the string to "" (empty string).

### **wxString(const wxString& x)**



Copy constructor.

**wxString(wxChar *ch*, size\_t *n* = 1)**

Constructs a string of *n* copies of character *ch*.

**wxString(const wxChar\* *psz*, size\_t *nLength* = wxSTRING\_MAXLEN)**

Takes first *nLength* characters from the C string *psz*. The default value of `wxSTRING_MAXLEN` means to take all the string.

Note that this constructor may be used even if *psz* points to a buffer with binary data (i.e. containing `NUL` characters) as long as you provide the correct value for *nLength*. However, the default form of it works only with strings without intermediate `NUL`s because it uses `strlen()` to calculate the effective length and it would not give correct results otherwise.

**wxString(const unsigned char\* *psz*, size\_t *nLength* = wxSTRING\_MAXLEN)**

For compilers using unsigned char: takes first *nLength* characters from the C string *psz*. The default value of `wxSTRING_MAXLEN` means take all the string. For ANSI builds only (note the use of `char` instead of `wxChar`).

### Constructors with conversion

The following constructors allow you to construct `wxString` from a wide string in ANSI build or from a C string in Unicode build.

**wxString(const wchar\_t\* *psz*, wxMBConv& *conv*, size\_t *nLength* = wxSTRING\_MAXLEN)**

Initializes the string from first *nLength* characters of wide string. The default value of `wxSTRING_MAXLEN` means take all the string. In ANSI build, *conv*'s `WC2MB` (p. 1040) method is called to convert *psz* to wide string. It is ignored in Unicode build.

**wxString(const char\* *psz*, wxMBConv& *conv*, size\_t *nLength* = wxSTRING\_MAXLEN)**

Initializes the string from first *nLength* characters of C string. The default value of `wxSTRING_MAXLEN` means take all the string. In Unicode build, *conv*'s `MB2WC` (p. 1039) method is called to convert *psz* to wide string. It is ignored in ANSI build.

### See also

*wxMBConv* classes (p. 2059), *mb\_str* (p. 1570), *wc\_str* (p. 1576)

### wxString::~~wxString

**~wxString()**

String destructor. Note that this is not virtual, so `wxString` must not be inherited from.

### wxString::Alloc

**void Alloc(size\_t *nLen*)**

Preallocate enough space for `wxString` to store *nLen* characters. This function may be used to increase speed when the string is constructed by repeated concatenation as in

```
// delete all vowels from the string
wxString DeleteAllVowels(const wxString& original)
{
    wxString result;

    size_t len = original.length();

    result.Alloc(len);

    for ( size_t n = 0; n < len; n++ )
    {
        if ( strchr("aeuio", tolower(original[n])) == NULL )
            result += original[n];
    }

    return result;
}
```

because it will avoid the need to reallocate string memory many times (in case of long strings). Note that it does not set the maximal length of a string - it will still expand if more than *nLen* characters are stored in it. Also, it does not truncate the existing string (use *Truncate()* (p. 1575) for this) even if its current length is greater than *nLen*

### **wxString::Append**

**wxString& Append(const wxChar\* psz)**

Concatenates *psz* to this string, returning a reference to it.

**wxString& Append(wxChar ch, int count = 1)**

Concatenates character *ch* to this string, *count* times, returning a reference to it.

### **wxString::AfterFirst**

**wxString AfterFirst(wxChar ch) const**

Gets all the characters after the first occurrence of *ch*. Returns the empty string if *ch* is not found.

### **wxString::AfterLast**

**wxString AfterLast(wxChar ch) const**

Gets all the characters after the last occurrence of *ch*. Returns the whole string if *ch* is not found.

### **wxString::BeforeFirst**

**wxString BeforeFirst(wxChar ch) const**

Gets all characters before the first occurrence of *ch*. Returns the whole string if *ch* is not found.

**wxString::BeforeLast****wxString BeforeLast(wxChar ch) const**

Gets all characters before the last occurrence of *ch*. Returns the empty string if *ch* is not found.

**wxString::c\_str****const wxChar \* c\_str() const**

Returns a pointer to the string data (`const char*` in ANSI build, `const wchar_t*` in Unicode build).

Note that the returned value will not be convertible to `char*` or `wchar_t*` in wxWidgets 3, consider using *char\_str* (p. 1563) or *wchar\_string* (p. 1576) if you need to pass string value to a function expecting non-const pointer.

**See also**

*mb\_str* (p. 1570), *wc\_str* (p. 1576), *fn\_str* (p. 1565), *char\_str* (p. 1563), *wchar\_string* (p. 1576)

**wxString::char\_str****wxWritableCharBuffer char\_str(wxMBConv& conv = wxConvLibc) const**

Returns an object with string data that is implicitly convertible to `char*` pointer. Note that any change to the returned buffer is lost and so this function is only usable for passing strings to legacy libraries that don't have const-correct API. Use *wxStringBuffer* (p. 1579) if you want to modify the string.

This function is new since wxWidgets version 2.8.4

**See also**

*mb\_str* (p. 1570), *wc\_str* (p. 1576), *fn\_str* (p. 1565), *c\_str* (p. 1563), *wchar\_str* (p. 1576)

**wxString::Clear****void Clear()**

Empties the string and frees memory occupied by it.

See also: *Empty* (p. 1564)

**wxString::Cmp****int Cmp(const wxString& s) const****int Cmp(const wxChar\* psz) const**

Case-sensitive comparison.

Returns a positive value if the string is greater than the argument, zero if it is equal to it or a negative value if it is less than the argument (same semantics as the standard *strcmp()* function).

See also *CmpNoCase* (p. 1564), *IsSameAs* (p. 1568).

**wxString::CmpNoCase****int CmpNoCase(const wxString& s) const****int CmpNoCase(const wxChar\* psz) const**

Case-insensitive comparison.

Returns a positive value if the string is greater than the argument, zero if it is equal to it or a negative value if it is less than the argument (same semantics as the standard *strcmp()* function).

See also *Cmp* (p. 1564), *IsSameAs* (p. 1568).

**wxString::CompareTo**

```
enum wxString::caseCompare {exact, ignoreCase};
```

**int CompareTo(const wxChar\* psz, caseCompare cmp = exact) const**

Case-sensitive comparison. Returns 0 if equal, 1 if greater or -1 if less.

This is a wxWidgets 1.xx compatibility function; use *Cmp* (p. 1564) instead.

**wxString::Contains****bool Contains(const wxString& str) const**

Returns *true* if target appears anywhere in *wxString*; else *false*.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

**wxString::Empty****void Empty()**

Makes the string empty, but doesn't free memory occupied by the string.

See also: *Clear()* (p. 1563).

### **wxString::Find**

**int Find(wxChar ch, bool fromEnd = false) const**

Searches for the given character. Returns the starting index, or `wxNOT_FOUND` if not found.

**int Find(const wxChar\* sz) const**

Searches for the given string. Returns the starting index, or `wxNOT_FOUND` if not found.

### **wxString::First**

**int First(wxChar c)**

**int First(const wxChar\* psz) const**

**int First(const wxString& str) const**

Same as *Find* (p. 1565).

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

### **wxString::fn\_str**

**const wchar\_t\* fn\_str() const**

**const char\* fn\_str() const**

**const wxCharBuffer fn\_str() const**

Returns string representation suitable for passing to OS' functions for file handling. In ANSI build, this is same as *c\_str* (p. 1563). In Unicode build, returned value can be either wide character string or C string in charset matching the `wxConvFileName` object, depending on the OS.

### **See also**

*wxMBConv* (p. 1038), *wc\_str* (p. 1576), *mb\_str* (p. 1576)

### **wxString::Format**

**static wxString Format(const wxChar \*format, ...)**

This static function returns the string containing the result of calling *Printf* (p. 1570) with the passed parameters on it.

### **See also**

*FormatV* (p. 1566), *Printf* (p. 1570)

**wxString::FormatV****static wxString FormatV(const wxChar \*format, va\_list argptr)**

This static function returns the string containing the result of calling *PrintfV* (p. 1571) with the passed parameters on it.

**See also**

*Format* (p. 1565), *PrintfV* (p. 1571)

**wxString::Freq****int Freq(wxChar ch) const**

Returns the number of occurrences of *ch* in the string.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

**wxString::From8BitData****static wxString From8BitData(const char\* buf, size\_t len)****static wxString From8BitData(const char\* buf)**

Converts given buffer of binary data from 8-bit string to wxString. In Unicode build, the string is interpreted as being in ISO-8859-1 encoding. The version without *len* parameter takes NUL-terminated data.

This is a convenience method useful when storing binary data in wxString. It should be used *only* for that purpose and only in conjunction with *To8BitData* (p. 1573). Use *mb\_str()* for conversion of character data to known encoding.

This function is new since wxWidgets version 2.8.4

**See also**

*To8BitData* (p. 1573)

**wxString::FromAscii****static wxString FromAscii(const char\* s)****static wxString FromAscii(const char c)**

Converts the string or character from an ASCII, 7-bit form to the native wxString representation. Most useful when using a Unicode build of wxWidgets (note the use of *char* instead of *wxChar*). Use *wxString constructors* (p. 1560) if you need to convert from another charset.

**wxString::FromUTF8**

**static wxString FromUTF8(const char\* s)**

**static wxString FromUTF8(const char\* s, size\_t len)**

Converts C string encoded in UTF-8 to wxString.

Note that this method assumes that *s* is a valid UTF-8 sequence and doesn't do any validation in release builds, it's validity is only checked in debug builds.

**wxString::GetChar**

**wxChar GetChar(size\_t n) const**

Returns the character at position *n* (read-only).

**wxString::GetData**

**const wxChar\* GetData() const**

wxWidgets compatibility conversion. Returns a constant pointer to the data in the string.

**wxString::GetWritableChar**

**wxChar& GetWritableChar(size\_t n)**

Returns a reference to the character at position *n*.

**wxString::GetWriteBuf**

**wxChar\* GetWriteBuf(size\_t len)**

Returns a writable buffer of at least *len* bytes. It returns a pointer to a new memory block, and the existing data will not be copied.

Call *wxString::UngetWriteBuf* (p. 1575) as soon as possible to put the string back into a reasonable state.

**wxString::Index**

**size\_t Index(wxChar ch) const**

**size\_t Index(const wxChar\* sz) const**

Same as *wxString::Find* (p. 1565).

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

**wxString::IsAscii**

**bool IsAscii() const**

Returns `true` if the string contains only ASCII characters.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

### **wxString::IsEmpty**

**bool IsEmpty() const**

Returns `true` if the string is empty.

### **wxString::IsNull**

**bool IsNull() const**

Returns `true` if the string is empty (same as *IsEmpty* (p. 1568)).

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

### **wxString::IsNumber**

**bool IsNumber() const**

Returns `true` if the string is an integer (with possible sign).

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

### **wxString::IsSameAs**

**bool IsSameAs(const wxChar\* psz, bool caseSensitive = true) const**

Test for string equality, case-sensitive (default) or not.

`caseSensitive` is `true` by default (case matters).

Returns `true` if strings are equal, `false` otherwise.

See also *Cmp* (p. 1564), *CmpNoCase* (p. 1564)

**bool IsSameAs(wxChar c, bool caseSensitive = true) const**

Test whether the string is equal to the single character `c`. The test is case-sensitive if `caseSensitive` is `true` (default) or not if it is `false`.

Returns `true` if the string is equal to the character, `false` otherwise.

See also *Cmp* (p. 1564), *CmpNoCase* (p. 1564)

### **wxString::IsWord**

**bool IsWord() const**

Returns `true` if the string is a word.



This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

### **wxString::Last**

**wxChar Last() const**

Returns the last character.

**wxChar& Last()**

Returns a reference to the last character (writable).

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

### **wxString::Left**

**wxString Left(size\_t count) const**

Returns the first *count* characters of the string.

### **wxString::Len**

**size\_t Len() const**

Returns the length of the string.

### **wxString::Length**

**size\_t Length() const**

Returns the length of the string (same as Len).

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

### **wxString::Lower**

**wxString Lower() const**

Returns this string converted to the lower case.

### **wxString::LowerCase**

**void LowerCase()**

Same as MakeLower.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

### **wxString::MakeLower**

**wxString& MakeLower()**

Converts all characters to lower case and returns the result.

### **wxString::MakeUpper**

**wxString& MakeUpper()**

Converts all characters to upper case and returns the result.

### **wxString::Matches**

**bool Matches(const wxChar\* szMask) const**

Returns `true` if the string contents matches a mask containing '\*' and '?'.

### **wxString::mb\_str**

**const char\* mb\_str(wxMBConv& conv) const**

**const wxCharBuffer mb\_str(wxMBConv& conv) const**

Returns multibyte (C string) representation of the string. In Unicode build, converts using *conv*'s `cWC2MB` (p. 1041) method and returns `wxCharBuffer`. In ANSI build, this function is same as `c_str` (p. 1563). The macro `wxWX2MBbuf` is defined as the correct return type (without `const`).

### **See also**

*wxMBConv* (p. 1038), *c\_str* (p. 1563), *wc\_str* (p. 1576), *fn\_str* (p. 1565), *char\_str* (p. 1563)

### **wxString::Mid**

**wxString Mid(size\_t first, size\_t count = wxSTRING\_MAXLEN) const**

Returns a substring starting at *first*, with length *count*, or the rest of the string if *count* is the default value.

### **wxString::Pad**

**wxString& Pad(size\_t count, wxChar pad = ' ', bool fromRight = true)**

Adds *count* copies of *pad* to the beginning, or to the end of the string (the default).

Removes spaces from the left or from the right (default).

### **wxString::Prepend**

**wxString& Prepend(const wxString& str)**

Prepends *str* to this string, returning a reference to this string.

### **wxString::Printf**

**int Printf(const wxChar\* pszFormat, ...)**

Similar to the standard function *sprintf()*. Returns the number of characters written, or an integer less than zero on error.

Note that if `wxUSE_PRINTF_POS_PARAMS` is set to 1, then this function supports Unix98-style positional parameters:

```
wxString str;

str.Printf(wxT("%d %d %d"), 1, 2, 3);
// str now contains "1 2 3"

str.Printf(wxT("%2$d %3$d %1$d"), 1, 2, 3);
// str now contains "2 3 1"
```

**NB:** This function will use a safe version of *vsprintf()* (usually called *vsnprintf()*) whenever available to always allocate the buffer of correct size. Unfortunately, this function is not available on all platforms and the dangerous *vsprintf()* will be used then which may lead to buffer overflows.

**wxString::PrintfV****int PrintfV(const wxChar\* pszFormat, va\_list argPtr)**

Similar to *vprintf*. Returns the number of characters written, or an integer less than zero on error.

**wxString::Remove****wxString& Remove(size\_t pos)**

Same as *Truncate*. Removes the portion from *pos* to the end of the string.

**wxString& Remove(size\_t pos, size\_t len)**

Removes *len* characters from the string, starting at *pos*.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

**wxString::RemoveLast****wxString& RemoveLast()**

Removes the last character.

**wxString::Replace****size\_t Replace(const wxChar\* szOld, const wxChar\* szNew, bool replaceAll = true)**

Replace first (or all) occurrences of substring with another one.

*replaceAll*: global replace (default), or only the first occurrence.

Returns the number of replacements made.

### **wxString::Right**

**wxString Right(size\_t count) const**

Returns the last *count* characters.

### **wxString::SetChar**

**void SetChar(size\_t n, wxChar ch)**

Sets the character at position *n*.

### **wxString::Shrink**

**void Shrink()**

Minimizes the string's memory. This can be useful after a call to *Alloc()* (p. 1561) if too much memory were preallocated.

### **wxString::StartsWith**

**bool StartsWith(const wxChar \*prefix, wxString \*rest = NULL) const**

This function can be used to test if the string starts with the specified *prefix*. If it does, the function will return `true` and put the rest of the string (i.e. after the prefix) into *rest* string if it is not `NULL`. Otherwise, the function returns `false` and doesn't modify *rest*.

### **wxString::EndsWith**

**bool EndsWith(const wxChar \*suffix, wxString \*rest = NULL) const**

This function can be used to test if the string ends with the specified *suffix*. If it does, the function will return `true` and put the beginning of the string before the suffix into *rest* string if it is not `NULL`. Otherwise, the function returns `false` and doesn't modify the *rest*.

### **wxString::Strip**

```
enum wxString::stripType {leading = 0x1, trailing = 0x2, both = 0x3};
```

**wxString Strip(stripType s = trailing) const**

Strip characters at the front and/or end. The same as *Trim* except that it doesn't change this string.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

**wxString::SubString****wxString SubString(size\_t from, size\_t to) const**

Returns the part of the string between the indices *from* and *to* inclusive.

This is a wxWidgets 1.xx compatibility function, use *Mid* (p. 1570) instead (but note that parameters have different meaning).

**wxString::To8BitData****const char\* To8BitData() const**

Converts the string to an 8-bit string (ANSI builds only).

**const wxCharBuffer To8BitData() const**

Converts the string to an 8-bit string in ISO-8859-1 encoding in the form of a wxCharBuffer (Unicode builds only).

This is a convenience method useful when storing binary data in wxString. It should be used *only* for this purpose. It is only valid to call this method on strings created using *From8BitData* (p. 1566).

This function is new since wxWidgets version 2.8.4

**See also**

*From8BitData* (p. 1566)

**wxString::ToAscii****const char\* ToAscii() const****const wxCharBuffer ToAscii() const**

Converts the string to an ASCII, 7-bit string in the form of a wxCharBuffer (Unicode builds only) or a C string (ANSI builds).

Note that this conversion only works if the string contains only ASCII characters. The *mb\_str* (p. 1570) method provides more powerful means of converting wxString to C string.

**wxString::ToDouble****bool ToDouble(double \*val) const**

Attempts to convert the string to a floating point number. Returns `true` on success (the number is stored in the location pointed to by *val*) or `false` if the string does not represent such number.

**See also**

*wxString::ToLong* (p. 1574),  
*wxString::ToULong* (p. 1574)

### **wxString::ToLong**

**bool ToLong(long \*val, int base = 10) const**

Attempts to convert the string to a signed integer in base *base*. Returns `true` on success in which case the number is stored in the location pointed to by *val* or `false` if the string does not represent a valid number in the given base.

The value of *base* must be comprised between 2 and 36, inclusive, or be a special value 0 which means that the usual rules of C numbers are applied: if the number starts with `0x` it is considered to be in base16, if it starts with `0` - in base 8 and in base 10 otherwise. Note that you may not want to specify the base 0 if you are parsing the numbers which may have leading zeroes as they can yield unexpected (to the user not familiar with C) results.

#### **See also**

*wxString::ToDouble* (p. 1573),  
*wxString::ToULong* (p. 1574)

### **wxString::ToLongLong**

**bool ToLongLong(wxLongLong\_t \*val, int base = 10) const**

This is exactly the same as *ToLong* (p. 1574) but works with 64 bit integer numbers.

Notice that currently it doesn't work (always returns `false`) if parsing of 64 bit numbers is not supported by the underlying C run-time library. Compilers with C99 support and Microsoft Visual C++ version 7 and higher do support this.

#### **See also**

*wxString::ToLong* (p. 1574),  
*wxString::ToULongLong* (p. 1575)

### **wxString::ToULong**

**bool ToULong(unsigned long \*val, int base = 10) const**

Attempts to convert the string to an unsigned integer in base *base*. Returns `true` on success in which case the number is stored in the location pointed to by *val* or `false` if the string does not represent a valid number in the given base. Please notice that this function behaves in the same way as the standard `strtoul()` and so it simply converts negative numbers to unsigned representation instead of rejecting them (e.g. -1 is returned as `ULONG_MAX`).

See *wxString::ToLong* (p. 1574) for the more detailed description of the *base* parameter.

#### **See also**

*wxString::ToDouble* (p. 1573),  
*wxString::ToLong* (p. 1574)

### **wxString::ToULongLong**

**bool ToULongLong(wxULongLong\_t \*val, int base = 10) const**

This is exactly the same as *ToULong* (p. 1574) but works with 64 bit integer numbers.

Please see *ToLongLong* (p. 1574) for additional remarks.

### **wxString::ToUTF8**

**const wxCharBuffer ToUF8() const**

Same as *utf8\_str* (p. 1576).

This function is new since wxWidgets version 2.8.4

### **wxString::Trim**

**wxString& Trim(bool fromRight = true)**

Removes white-space (space, tabs, form feed, newline and carriage return) from the left or from the right end of the string (right is default).

### **wxString::Truncate**

**wxString& Truncate(size\_t len)**

Truncate the string to the given length.

### **wxString::UngetWriteBuf**

**void UngetWriteBuf()**

**void UngetWriteBuf(size\_t len)**

Puts the string back into a reasonable state (in which it can be used normally), after *wxString::GetWriteBuf* (p. 1567) was called.

The version of the function without the *len* parameter will calculate the new string length itself assuming that the string is terminated by the first `NUL` character in it while the second one will use the specified length and thus is the only version which should be used with the strings with embedded `NUL`s (it is also slightly more efficient as `strlen()` doesn't have to be called).

### **wxString::Upper**

**wxString Upper() const**

Returns this string converted to upper case.

### **wxString::UpperCase**

#### **void UpperCase()**

The same as MakeUpper.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

### **wxString::utf8\_str**

#### **const wxCharBuffer utf8\_str() const**

Converts the strings contents to UTF-8 and returns it as a temporary wxCharBuffer object.

This function is new since wxWidgets version 2.8.4

### **wxString::wc\_str**

#### **const wchar\_t\* wc\_str(wxMBConv& conv) const**

#### **const wxWCharBuffer wc\_str(wxMBConv& conv) const**

Returns wide character representation of the string. In ANSI build, converts using *conv*'s *cMB2WC* (p. 1040) method and returns wxWCharBuffer. In Unicode build, this function is same as *c\_str* (p. 1563). The macro *wxWX2WCbuf* is defined as the correct return type (without *const*).

#### **See also**

*wxMBConv* (p. 1038), *c\_str* (p. 1563), *mb\_str* (p. 1576), *fn\_str* (p. 1565), *wchar\_str* (p. 1576)

### **wxString::wchar\_str**

#### **wxWritableWCharBuffer wchar\_str() const**

Returns an object with string data that is implicitly convertible to *char\** pointer. Note that any change to the returned buffer is lost and so this function is only usable for passing strings to legacy libraries that don't have *const*-correct API. Use *wxStringBuffer* (p. 1579) if you want to modify the string.

This function is new since wxWidgets version 2.8.4

#### **See also**

*mb\_str* (p. 1570), *wc\_str* (p. 1576), *fn\_str* (p. 1565), *c\_str* (p. 1563), *char\_str* (p. 1563)

### **wxString::operator!**

#### **bool operator!() const**



Empty string is `false`, so `!string` will only return `true` if the string is empty. This allows the tests for NULLness of a `const wxChar *` pointer and emptiness of the string to look the same in the code and makes it easier to port old code to `wxString`.

See also *IsEmpty()* (p. 1568).

#### **wxString::operator =**

**wxString& operator =(const wxString& str)**

**wxString& operator =(const wxChar\* psz)**

**wxString& operator =(wxChar c)**

Assignment: the effect of each operation is the same as for the corresponding constructor (see *wxString constructors* (p. 1560)).

#### **wxString::operator +**

Concatenation: all these operators return a new string equal to the concatenation of the operands.

**wxString operator +(const wxString& x, const wxString& y)**

**wxString operator +(const wxString& x, const wxChar\* y)**

**wxString operator +(const wxString& x, wxChar y)**

**wxString operator +(const wxChar\* x, const wxString& y)**

#### **wxString::operator +=**

**void operator +=(const wxString& str)**

**void operator +=(const wxChar\* psz)**

**void operator +=(wxChar c)**

Concatenation in place: the argument is appended to the string.

#### **wxString::operator []**

**wxChar& operator [] (size\_t i)**

**wxChar operator [] (size\_t i) const**

**wxChar& operator [] (int i)**

**wxChar operator [] (int i) const**

Element extraction.

**wxString::operator ()****wxString operator ()(size\_t start, size\_t len)**

Same as Mid (substring extraction).

**wxString::operator <<****wxString& operator <<(const wxString& str)****wxString& operator <<(const wxChar\* psz)****wxString& operator <<(wxChar ch)**

Same as +=.

**wxString& operator <<(int i)****wxString& operator <<(float f)****wxString& operator <<(double d)**

These functions work as C++ stream insertion operators: they insert the given value into the string. Precision or format cannot be set using them, you can use *Printf* (p. 1570) for this.

**wxString::operator >>****friend istream& operator >>(istream& is, wxString& str)**

Extraction from a stream.

**wxString::operator const wxChar\*****operator const wxChar\*() const**

Implicit conversion to a C string.

**Comparison operators****bool operator ==(const wxString& x, const wxString& y)****bool operator ==(const wxString& x, const wxChar\* t)****bool operator !=(const wxString& x, const wxString& y)****bool operator !=(const wxString& x, const wxChar\* t)****bool operator >(const wxString& x, const wxString& y)****bool operator >(const wxString& x, const wxChar\* t)****bool operator >=(const wxString& x, const wxString& y)**

**bool operator >=(const wxString& x, const wxChar\* t)**

**bool operator <(const wxString& x, const wxString& y)**

**bool operator <(const wxString& x, const wxChar\* t)**

**bool operator <=(const wxString& x, const wxString& y)**

**bool operator <=(const wxString& x, const wxChar\* t)**

#### Remarks

These comparisons are case-sensitive.

## wxStringBuffer

This tiny class allows to conveniently access the *wxString* (p. 1553) internal buffer as a writable pointer without any risk of forgetting to restore the string to the usable state later.

For example, assuming you have a low-level OS function called `GetMeaningOfLifeAsString(char *)` returning the value in the provided buffer (which must be writable, of course) you might call it like this:

```
wxString theAnswer;  
GetMeaningOfLifeAsString(wxStringBuffer(theAnswer, 1024));  
if ( theAnswer != "42" )  
{  
    wxLogError("Something is very wrong!");  
}
```

Note that the exact usage of this depends on whether or not `wxUSE_STL` is enabled. If `wxUSE_STL` is enabled, `wxStringBuffer` creates a separate empty character buffer, and if `wxUSE_STL` is disabled, it uses `GetWriteBuf()` from `wxString`, keeping the same buffer `wxString` uses intact. In other words, relying on `wxStringBuffer` containing the old `wxString` data is probably not a good idea if you want to build your program in both with and without `wxUSE_STL`.

#### Derived from

None

#### Include files

<wx/string.h>

## wxStringBuffer::wxStringBuffer

**wxStringBuffer(const wxString& str, size\_t len)**

Constructs a writable string buffer object associated with the given string and containing enough space for at least *len* characters. Basically, this is equivalent to calling

*GetWriteBuf* (p. 1567) and saving the result.

### **wxStringBuffer::~~wxStringBuffer**

#### **~wxStringBuffer()**

Restores the string passed to the constructor to the usable state by calling *UngetWriteBuf* (p. 1575) on it.

### **wxStringBuffer::operator wxChar \***

#### **wxChar \* operator wxChar \*()**

Returns the writable pointer to a buffer of the size at least equal to the length specified in the constructor.

## **wxStringBufferLength**

This tiny class allows to conveniently access the *wxString* (p. 1553) internal buffer as a writable pointer without any risk of forgetting to restore the string to the usable state later, and allows the user to set the internal length of the string.

For example, assuming you have a low-level OS function called `int GetMeaningOfLifeAsString(char *)` copying the value in the provided buffer (which must be writable, of course), and returning the actual length of the string, you might call it like this:

```
wxString theAnswer;
wxStringBuffer theAnswerBuffer(theAnswer, 1024);
int nLength = GetMeaningOfLifeAsString(theAnswerBuffer);
theAnswerBuffer.SetLength(nLength);
if ( theAnswer != "42" )
{
    wxLogError("Something is very wrong!");
}
```

Note that the exact usage of this depends on whether or not `wxUSE_STL` is enabled. If `wxUSE_STL` is enabled, `wxStringBuffer` creates a separate empty character buffer, and if `wxUSE_STL` is disabled, it uses `GetWriteBuf()` from `wxString`, keeping the same buffer `wxString` uses intact. In other words, relying on `wxStringBuffer` containing the old `wxString` data is probably not a good idea if you want to build your program in both with and without `wxUSE_STL`.

Note that `SetLength` must be called before `wxStringBufferLength` destructs.

### **Derived from**

None

### **Include files**

<wx/string.h>

**wxStringBufferLength::wxStringBufferLength****wxStringBufferLength(const wxString& str, size\_t len)**

Constructs a writable string buffer object associated with the given string and containing enough space for at least *len* characters. Basically, this is equivalent to calling *GetWriteBuf* (p. 1567) and saving the result.

**wxStringBufferLength::~~wxStringBufferLength****~wxStringBufferLength()**

Restores the string passed to the constructor to the usable state by calling *UngetWriteBuf* (p. 1575) on it.

**wxStringBufferLength::SetLength****void SetLength(size\_t nLength)**

Sets the internal length of the string referred to by *wxStringBufferLength* to *nLength* characters.

Must be called before *wxStringBufferLength* destructs.

**wxStringBufferLength::operator wxChar \*****wxChar \* operator wxChar \*()**

Returns the writable pointer to a buffer of the size at least equal to the length specified in the constructor.

**wxStringClientData**

Predefined client data class for holding a string.

**Derived from**

*wxClientData* (p. 193)

**Include files**

<clntdata.h>

**Data structures****wxStringClientData::wxStringClientData**

**wxStringClientData()**

Empty constructor.

**wxStringClientData(const wxString& data)**

Create client data with string.

**wxStringClientData::GetData****const wxString& GetData() const**

Get string client data.

**wxStringClientData::SetData****void SetData(const wxString& data)**

Set string client data.

## **wxStringInputStream**

This class implements an input stream which reads data from a string. It supports seeking.

**Derived from**

*wxInputStream* (p. 941)

**Include files**

<wx/sstream.h>

**wxStringInputStream::wxStringInputStream****wxStringInputStream(const wxString& s)**

Creates a new read-only stream using the specified string. Note that the string is copied by the stream so if the original string is modified after using this constructor, changes to it are not reflected when reading from stream.

## **wxStringOutputStream**

This class implements an output stream which writes data either to a user-provided or internally allocated string. Note that currently this stream does not support seeking but can tell its current position.

**Derived from**

*wxOutputStream* (p. 1156)

**Include files**

<wx/sstream.h>

**wxStringOutputStream::wxStringOutputStream**

**wxStringOutputStream(wxString \*str = NULL)**

If the provided pointer is non-NULL, data will be written to it. Otherwise, an internal string is used for the data written to this stream, use *GetString()* (p. 1583) to get access to it.

If *str* is used, data written to the stream is appended to the current contents of it, i.e. the string is not cleared here. However if it is not empty, the positions returned by *TellO* (p. 1157) will be offset by the initial string length, i.e. initial stream position will be the initial length of the string and not 0.

**wxStringOutputStream::GetString**

**const wxString& GetString() const**

Returns the string containing all the data written to the stream so far.

**wxStringTokenizer**

wxStringTokenizer helps you to break a string up into a number of tokens. It replaces the standard C function *strtok()* and also extends it in a number of ways.

To use this class, you should create a wxStringTokenizer object, give it the string to tokenize and also the delimiters which separate tokens in the string (by default, white space characters will be used).

Then *GetNextToken* (p. 1585) may be called repeatedly until it *HasMoreTokens* (p. 1585) returns false.

For example:

```
wxStringTokenizer tkz(wxT("first:second:third:fourth"), wxT(":"));
while ( tkz.HasMoreTokens() )
{
    wxString token = tkz.GetNextToken();

    // process token here
}
```

By default, wxStringTokenizer will behave in the same way as *strtok()* if the delimiters string only contains white space characters but, unlike the standard function, it will return empty tokens if this is not the case. This is helpful for parsing strictly formatted data where the number of fields is fixed but some of them may be empty (i.e. TAB or comma delimited text files).

The behaviour is governed by the last *constructor* (p. 1584)/*SetString* (p. 1585) parameter *mode* which may be one of the following:

<code>wxTOKEN_DEFAULT</code>	Default behaviour (as described above): same as <code>wxTOKEN_STRTOK</code> if the delimiter string contains only whitespaces, same as <code>wxTOKEN_RET_EMPTY</code> otherwise
<code>wxTOKEN_RET_EMPTY</code>	In this mode, the empty tokens in the middle of the string will be returned, i.e. "a : b : " will be tokenized in three tokens 'a', " and 'b'. Notice that all trailing delimiters are ignored in this mode, not just the last one, i.e. a string "a : b : " would still result in the same set of tokens.
<code>wxTOKEN_RET_EMPTY_ALL</code>	In this mode, empty trailing tokens (including the one after the last delimiter character) will be returned as well. The string "a : b : " will be tokenized in four tokens: the already mentioned ones and another empty one as the last one and a string "a : b : " will have five tokens.
<code>wxTOKEN_RET_DELIMS</code>	In this mode, the delimiter character after the end of the current token (there may be none if this is the last token) is returned appended to the token. Otherwise, it is the same mode as <code>wxTOKEN_RET_EMPTY</code> . Notice that there is no mode like this one but behaving like <code>wxTOKEN_RET_EMPTY_ALL</code> instead of <code>wxTOKEN_RET_EMPTY</code> , use <code>wxTOKEN_RET_EMPTY_ALL</code> and <i>GetLastDelimiter()</i> (p. 1585) to emulate it.
<code>wxTOKEN_STRTOK</code>	In this mode the class behaves exactly like the standard <code>strtok()</code> function: the empty tokens are never returned.

### Derived from

*wxObject* (p. 1148)

### See also

*wxStringTokenize* (p. 1931)

### Include files

<wx/tokenzr.h>

## **wxStringTokenizer::wxStringTokenizer**

### **wxStringTokenizer()**

Default constructor. You must call *SetString* (p. 1585) before calling any other methods.

**wxStringTokenizer(const wxString& str, const wxString& delims = " \t\r\n",**



**wxStringTokenizerMode** *mode* = *wxTOKEN\_DEFAULT*)

Constructor. Pass the string to tokenize, a string containing delimiters and the mode specifying how the string should be tokenized.

### **wxStringTokenizer::CountTokens**

**int CountTokens() const**

Returns the number of tokens remaining in the input string. The number of tokens returned by this function is decremented each time *GetNextToken* (p. 1585) is called and when it reaches 0 *HasMoreTokens* (p. 1585) returns *false*.

### **wxStringTokenizer::HasMoreTokens**

**bool HasMoreTokens() const**

Returns *true* if the tokenizer has further tokens, *false* if none are left.

### **wxStringTokenizer::GetLastDelimiter**

**wxChar GetLastDelimiter()**

Returns the delimiter which ended scan for the last token returned by *GetNextToken()* (p. 1585) or *NUL* if there had been no calls to this function yet or if it returned the trailing empty token in *wxTOKEN\_RET\_EMPTY\_ALL* mode.

This function is new since wxWidgets version 2.7.0

### **wxStringTokenizer::GetNextToken**

**wxString GetNextToken() const**

Returns the next token or empty string if the end of string was reached.

### **wxStringTokenizer::GetPosition**

**size\_t GetPosition() const**

Returns the current position (i.e. one index after the last returned token or 0 if *GetNextToken()* has never been called) in the original string.

### **wxStringTokenizer::GetString**

**wxString GetString() const**

Returns the part of the starting string without all token already extracted.

### **wxStringTokenizer::SetString**

```
void SetString(const wxString& to_tokenize, const wxString& delims = " \t\r\n",  
wxStringTokenizerMode mode = wxTOKEN_DEFAULT)
```

Initializes the tokenizer.

Pass the string to tokenize, a string containing delimiters, and the mode specifying how the string should be tokenized.

## wxSymbolPickerDialog

wxSymbolPickerDialog presents the user with a choice of fonts and a grid of available characters. This modal dialog provides the application with a selected symbol and optional font selection.

Although this dialog is contained in the rich text library, the dialog is generic and can be used in other contexts.

To use the dialog, pass a default symbol specified as a string, an initial font name, and a current font name. The difference between the initial font and current font is that the initial font determines what the font control will be set to when the dialog shows - an empty string will show the selection *normal text*. The current font, on the other hand, is used by the dialog to determine what font to display the characters in, even when no initial font is selected. This allows the user (and application) to distinguish between inserting a symbol in the current font, and inserting it with a specified font.

When the dialog is dismissed, the application can get the selected symbol with `GetSymbol` and test whether a font was specified with `UseNormalFont`, fetching the specified font with `GetFontName`.

Here's a realistic example, inserting the supplied symbol into a rich text control in either the current font or specified font.

```
wxRichTextCtrl* ctrl = (wxRichTextCtrl*)  
FindWindow(ID_RICHTEXT_CTRL);  
  
wxTextAttrEx attr;  
attr.SetFlags(wxTEXT_ATTR_FONT);  
ctrl->GetStyle(ctrl->GetInsertionPoint(), attr);  
  
wxString currentFontName;  
if (attr.HasFont() && attr.GetFont().Ok())  
    currentFontName = attr.GetFont().GetFaceName();  
  
// Don't set the initial font in the dialog (so the user is choosing  
// 'normal text', i.e. the current font) but do tell the dialog  
// what 'normal text' is.  
  
wxSymbolPickerDialog dlg(wxT(""), wxEmptyString,  
currentFontName, this);  
  
if (dlg.ShowModal() == wxID_OK)  
{  
    if (dlg.HasSelection())  
    {
```

```
        long insertionPoint = ctrl->GetInsertionPoint();

        ctrl->WriteText(dlg.GetSymbol());

        if (!dlg.UseNormalFont())
        {
            wxFont font(attr.GetFont());
            font.SetFaceName(dlg.GetFontName());
            attr.SetFont(font);
            ctrl->SetStyle(insertionPoint, insertionPoint+1,
attr);
        }
    }
}
```

**Derived from**

*wxDialog* (p. 496)

**Include files**

<wx/richtext/richtextsymboldlg.h>

**Data structures****wxSymbolPickerDialog::wxSymbolPickerDialog**

**wxSymbolPickerDialog(const wxString& *symbol*, const wxString& *initialFont*, const wxString& *normalTextFont*, wxWindow\* *parent*, wxWindowID *id* = wxID\_ANY, const wxString& *title* = \_("Symbols"), const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size* = wxDefaultSize, long *style* = wxDEFAULT\_DIALOG\_STYLE|wxRESIZE\_BORDER|wxCLOSE\_BOX)**

**wxSymbolPickerDialog()**

Constructors.

**Parameters**

*symbol*

The initial symbol to show. Specify a single character in a string, or an empty string.

*initialFont*

The initial font to be displayed in the font list. If empty, the item *normal text* will be selected.

*normalTextFont*

The font the dialog will use to display the symbols if the initial font is empty.

*parent*

The dialog's parent.

*id*

The dialog's identifier.

*title*

The dialog's caption.

*pos*

The dialog's position.

*size*

The dialog's size.

*style*

The dialog's window style.

### **wxSymbolPickerDialog::Create**

```
bool Create(const wxString& symbol, const wxString& initialFont, const wxString&
normalTextFont, wxWindow* parent, wxWindowID id = wxID_ANY, const wxString&
title = _("Symbols"), const wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize, long style =
wxDEFAULT_DIALOG_STYLE|wxRESIZE_BORDER|wxCLOSE_BOX)
```

Creation: see *the constructor* (p. 1587) for details about the parameters.

### **wxSymbolPickerDialog::GetFontName**

```
wxString GetFontName() const
```

Returns the font name (the font reflected in the font list).

### **wxSymbolPickerDialog::GetFromUnicode**

```
bool GetFromUnicode() const
```

Returns `true` if the dialog is showing the full range of Unicode characters.

### **wxSymbolPickerDialog::GetNormalTextFontName**

```
wxString GetNormalTextFontName() const
```

Gets the font name used for displaying symbols in the absence of a selected font.

### **wxSymbolPickerDialog::GetSymbol**

**wxString GetSymbol() const**

Gets the current or initial symbol as a string.

**wxSymbolPickerDialog::GetSymbolChar****int GetSymbolChar() const**

Gets the selected symbol character as an integer.

**wxSymbolPickerDialog::HasSelection****bool HasSelection() const**

Returns `true` if a symbol is selected.

**wxSymbolPickerDialog::SetFontName****void SetFontName(const wxString& value)**

Sets the initial/selected font name.

**wxSymbolPickerDialog::SetFromUnicode****void SetFromUnicode(bool value)**

Sets the internal flag indicating that the full Unicode range should be displayed.

**wxSymbolPickerDialog::SetNormalTextFontName****void SetNormalTextFontName(const wxString& value)**

Sets the name of the font to be used in the absence of a selected font.

**wxSymbolPickerDialog::SetSymbol****void SetSymbol(const wxString& value)**

Sets the symbol as a one or zero character string.

**wxSymbolPickerDialog::SetUnicodeMode****void SetUnicodeMode(bool unicodeMode)**

Sets Unicode display mode.

**wxSymbolPickerDialog::UseNormalFont****bool UseNormalFont() const**

Returns true if the has specified normal text - that is, there is no selected font.

## **wxSysColourChangedEvent**

This class is used for system colour change events, which are generated when the user changes the colour settings using the control panel. This is only appropriate under Windows.

### **Derived from**

*wxEvent* (p. 572)

*wxObject* (p. 1148)

### **Include files**

<wx/event.h>

### **Event table macros**

To process a system colour changed event, use this event handler macro to direct input to a member function that takes a `wxSysColourChanged` argument.

**EVT\_SYS\_COLOUR\_CHANGED(func)** Process a `wxEVT_SYS_COLOUR_CHANGED` event.

### **Remarks**

The default event handler for this event propagates the event to child windows, since Windows only sends the events to top-level windows. If intercepting this event for a top-level window, remember to call the base class handler, or to pass the event on to the window's children explicitly.

### **See also**

*Event handling overview* (p. 2077)

## **wxSysColourChangedEvent::wxSysColourChangedEvent**

**wxSysColourChangedEvent()**

Constructor.

## **wxSystemOptions**

`wxSystemOptions` stores option/value pairs that `wxWidgets` itself or applications can use to alter behaviour at run-time. It can be used to optimize behaviour that doesn't deserve a distinct API, but is still important to be able to configure.

These options are currently recognised by `wxWidgets`.

**Windows**

Option	Value
no-maskblt	1 to never use WIN32's MaskBlt function, 0 to allow it to be used where possible. Default: 0. In some circumstances the MaskBlt function can be slower than using the fallback code, especially if using DC cacheing. By default, MaskBlt will be used where it is implemented by the operating system and driver.
msw.remap	If 1 (the default), wxToolBar bitmap colours will be remapped to the current theme's values. Set this to 0 to disable this functionality, for example if you're using more than 16 colours in your tool bitmaps.
msw.window.no-clip-children	If 1, windows will not automatically get the WS_CLIPCHILDREN style. This restores the way windows are refreshed back to the method used in versions of wxWidgets earlier than 2.5.4, and for some complex window hierarchies it can reduce apparent refresh delays. You may still specify wxCLIP_CHILDREN for individual windows.
msw.notebook.themed-background	If set to 0, globally disables themed backgrounds on notebook pages. Note that this won't disable the theme on the actual notebook background (noticeable only if there are no pages).
msw.staticbox.optimized-paint	If set to 0, switches off optimized wxStaticBox painting. Setting this to 0 causes more flicker, but allows applications to paint graphics on the parent of a static box (the optimized refresh causes any such drawing to disappear).
msw.display.directdraw	If set to 1, use DirectDraw-based implementation of <i>wxDisplay</i> (p. 519). By default the standard Win32 functions are used.
msw.font.no-proof-quality	If set to 1, use default fonts quality instead of proof quality when creating fonts. With proof quality the fonts have slightly better appearance but not all fonts are available in this quality, e.g. the Terminal font in small sizes is not and this option may be used if wider fonts selection is more important than higher quality.

**GTK+**

Option	Value
gtk.window.force-background-colour	If 1, the backgrounds of windows with the <code>wxBG_STYLE_COLOUR</code> background style are cleared forcibly instead of relying on the underlying GTK+ window colour. This works around a display problem when running applications under KDE with the <code>gtk-qt</code> theme installed (0.6 and below).

## Mac

Option	Value
mac.window-plain-transition	If 1, uses a plainer transition when showing a window. You can also use the symbol <code>wxMAC_WINDOW_PLAIN_TRANSITION</code> .
window-default-variant	The default variant used by windows (cast to integer from the <code>wxWindowVariant</code> enum). Also known as <code>wxWINDOW_DEFAULT_VARIANT</code> .
mac.listctrl.always_use_generic	Tells <code>wxListCtrl</code> to use the generic control even when it is capable of using the native control instead. Also known as <code>wxMAC_ALWAYS_USE_GENERIC_LISTCTRL</code> .

## MGL

Option	Value
mgl.aa-threshold	Set this integer option to point size below which fonts are not antialiased. Default: 10.
mgl.screen-refresh	Screen refresh rate in Hz. A reasonable default is used if not specified.

## Motif

Option	Value
motif.largebuttons	If 1, uses a bigger default size for <code>wxButtons</code> .

The compile-time option to include or exclude this functionality is `wxUSE_SYSTEM_OPTIONS`.

## Derived from

*wxObject* (p. 1148)

## Include files

<wx/sysopt.h>



**wxSystemOptions::wxSystemOptions****wxSystemOptions()**

Default constructor. You don't need to create an instance of `wxSystemOptions` since all of its functions are static.

**wxSystemOptions::GetOption****wxString GetOption(const wxString& name) const**

Gets an option. The function is case-insensitive to *name*.

Returns empty string if the option hasn't been set.

**See also**

*wxSystemOptions::SetOption* (p. 1594), *wxSystemOptions::GetOptionInt* (p. 1593),  
*wxSystemOptions::HasOption* (p. 1593)

**wxSystemOptions::GetOptionInt****int GetOptionInt(const wxString& name) const**

Gets an option as an integer. The function is case-insensitive to *name*.

If the option hasn't been set, this function returns 0.

**See also**

*wxSystemOptions::SetOption* (p. 1594), *wxSystemOptions::GetOption* (p. 1593),  
*wxSystemOptions::HasOption* (p. 1593)

**wxSystemOptions::HasOption****bool HasOption(const wxString& name) const**

Returns `true` if the given option is present. The function is case-insensitive to *name*.

**See also**

*wxSystemOptions::SetOption* (p. 1594), *wxSystemOptions::GetOption* (p. 1593),  
*wxSystemOptions::GetOptionInt* (p. 1593)

**wxSystemOptions::IsFalse****bool IsFalse(const wxString& name) const**

Returns `true` if the option with the given *name* had been set to 0 value. This is mostly useful for boolean options for which you can't use `GetOptionInt(name) == 0` as this

would also be true if the option hadn't been set at all.

### **wxSystemOptions::SetOption**

**void SetOption(const wxString& name, const wxString& value)**

**void SetOption(const wxString& name, int value)**

Sets an option. The function is case-insensitive to *name*.

#### **See also**

*wxSystemOptions::GetOption* (p. 1593), *wxSystemOptions::GetOptionInt* (p. 1593),  
*wxSystemOptions::HasOption* (p. 1593)

## **wxSystemSettings**

*wxSystemSettings* allows the application to ask for details about the system. This can include settings such as standard colours, fonts, and user interface element sizes.

#### **Derived from**

*wxObject* (p. 1148)

#### **Include files**

<wx/settings.h>

#### **See also**

*wxFont* (p. 655), *wxColour* (p. 214)

### **wxSystemSettings::wxSystemSettings**

**wxSystemSettings()**

Default constructor. You don't need to create an instance of *wxSystemSettings* since all of its functions are static.

### **wxSystemSettings::GetColour**

**static wxColour GetColour(wxSystemColour index)**

Returns a system colour.

*index* can be one of:

<b>wxSYS_COLOUR_SCROLLBAR</b>	The scrollbar grey area.
<b>wxSYS_COLOUR_BACKGROUND</b>	The desktop colour.

<b>wxSYS_COLOUR_ACTIVECAPTION</b>	Active window caption.
<b>wxSYS_COLOUR_INACTIVECAPTION</b>	Inactive window caption.
<b>wxSYS_COLOUR_MENU</b>	Menu background.
<b>wxSYS_COLOUR_WINDOW</b>	Window background.
<b>wxSYS_COLOUR_WINDOWFRAME</b>	Window frame.
<b>wxSYS_COLOUR_MENUTEXT</b>	Menu text.
<b>wxSYS_COLOUR_WINDOWTEXT</b>	Text in windows.
<b>wxSYS_COLOUR_CAPTIONTEXT</b>	Text in caption, size box and scrollbar arrow box.
<b>wxSYS_COLOUR_ACTIVEBORDER</b>	Active window border.
<b>wxSYS_COLOUR_INACTIVEBORDER</b>	Inactive window border.
<b>wxSYS_COLOUR_APPWORKSPACE</b>	Background colour MDI applications.
<b>wxSYS_COLOUR_HIGHLIGHT</b>	Item(s) selected in a control.
<b>wxSYS_COLOUR_HIGHLIGHTTEXT</b>	Text of item(s) selected in a control.
<b>wxSYS_COLOUR_BTNFACE</b>	Face shading on push buttons.
<b>wxSYS_COLOUR_BTNSHADOW</b>	Edge shading on push buttons.
<b>wxSYS_COLOUR_GRAYTEXT</b>	Greyed (disabled) text.
<b>wxSYS_COLOUR_BTNTEXT</b>	Text on push buttons.
<b>wxSYS_COLOUR_INACTIVECAPTIONTEXT</b>	Colour of text in active captions.
<b>wxSYS_COLOUR_BTNHIGHLIGHT</b>	Highlight colour for buttons (same as wxSYS_COLOUR_3DHILIGHT).
<b>wxSYS_COLOUR_3DDKSHADOW</b>	Dark shadow for three-dimensional display elements.
<b>wxSYS_COLOUR_3DLIGHT</b>	Light colour for three-dimensional display elements.
<b>wxSYS_COLOUR_INFOTEXT</b>	Text colour for tooltip controls.
<b>wxSYS_COLOUR_INFOBK</b>	Background colour for tooltip controls.
<b>wxSYS_COLOUR_DESKTOP</b>	Same as wxSYS_COLOUR_BACKGROUND.
<b>wxSYS_COLOUR_3DFACE</b>	Same as wxSYS_COLOUR_BTNFACE.
<b>wxSYS_COLOUR_3DSHADOW</b>	Same as wxSYS_COLOUR_BTNSHADOW.

<b>wxSYS_COLOUR_3DHIGHLIGHT</b>	Same as wxSYS_COLOUR_BTNHIGHLIGHT.
<b>wxSYS_COLOUR_3DHILIGHT</b>	Same as wxSYS_COLOUR_BTNHIGHLIGHT.
<b>wxSYS_COLOUR_BTNHILIGHT</b>	Same as wxSYS_COLOUR_BTNHIGHLIGHT.

**wxPython note:** This static method is implemented in Python as a standalone function named `wxSystemSettings_GetColour`

### **wxSystemSettings::GetFont**

**static wxFont GetFont(wxSystemFont *index*)**

Returns a system font.

*index* can be one of:

<b>wxSYS_OEM_FIXED_FONT</b>	Original equipment manufacturer dependent fixed-pitch font.
<b>wxSYS_ANSI_FIXED_FONT</b>	Windows fixed-pitch font.
<b>wxSYS_ANSI_VAR_FONT</b>	Windows variable-pitch (proportional) font.
<b>wxSYS_SYSTEM_FONT</b>	System font.
<b>wxSYS_DEVICE_DEFAULT_FONT</b>	Device-dependent font (Windows NT only).
<b>wxSYS_DEFAULT_GUI_FONT</b>	Default font for user interface objects such as menus and dialog boxes. Note that with modern GUIs nothing guarantees that the same font is used for all GUI elements, so some controls might use a different font by default.

**wxPython note:** This static method is implemented in Python as a standalone function named `wxSystemSettings_GetFont`

### **wxSystemSettings::GetMetric**

**static int GetMetric(wxSystemMetric *index*, wxWindow\* *win* = NULL)**

Returns the value of a system metric, or -1 if the metric is not supported on the current system. The value of *win* determines if the metric returned is a global value or a *wxWindow* (p. 1795) based value, in which case it might determine the widget, the display the window is on, or something similar. The window given should be as close to the metric as possible (e.g a *wxTopLevelWindow* in case of the `wxSYS_CAPTION_Y` metric).

*index* can be one of:

<b>wxSYS_MOUSE_BUTTONS</b>	Number of buttons on mouse, or zero if no mouse was installed.
<b>wxSYS_BORDER_X</b>	Width of single border.

<b>wxSYS_BORDER_Y</b>	Height of single border.
<b>wxSYS_CURSOR_X</b>	Width of cursor.
<b>wxSYS_CURSOR_Y</b>	Height of cursor.
<b>wxSYS_DCLICK_X</b>	Width in pixels of rectangle within which two successive mouse clicks must fall to generate a double-click.
<b>wxSYS_DCLICK_Y</b>	Height in pixels of rectangle within which two successive mouse clicks must fall to generate a double-click.
<b>wxSYS_DRAG_X</b>	Width in pixels of a rectangle centered on a drag point to allow for limited movement of the mouse pointer before a drag operation begins.
<b>wxSYS_DRAG_Y</b>	Height in pixels of a rectangle centered on a drag point to allow for limited movement of the mouse pointer before a drag operation begins.
<b>wxSYS_EDGE_X</b>	Width of a 3D border, in pixels.
<b>wxSYS_EDGE_Y</b>	Height of a 3D border, in pixels.
<b>wxSYS_HSCROLL_ARROW_X</b>	Width of arrow bitmap on horizontal scrollbar.
<b>wxSYS_HSCROLL_ARROW_Y</b>	Height of arrow bitmap on horizontal scrollbar.
<b>wxSYS_HTHUMB_X</b>	Width of horizontal scrollbar thumb.
<b>wxSYS_ICON_X</b>	The default width of an icon.
<b>wxSYS_ICON_Y</b>	The default height of an icon.
<b>wxSYS_ICONSPACING_X</b>	Width of a grid cell for items in large icon view, in pixels. Each item fits into a rectangle of this size when arranged.
<b>wxSYS_ICONSPACING_Y</b>	Height of a grid cell for items in large icon view, in pixels. Each item fits into a rectangle of this size when arranged.
<b>wxSYS_WINDOWMIN_X</b>	Minimum width of a window.
<b>wxSYS_WINDOWMIN_Y</b>	Minimum height of a window.
<b>wxSYS_SCREEN_X</b>	Width of the screen in pixels.
<b>wxSYS_SCREEN_Y</b>	Height of the screen in pixels.
<b>wxSYS_FRAME_SIZE_X</b>	Width of the window frame for a wxTHICK_FRAME window.

<b>wxSYS_FRAME_SIZE_Y</b>	Height of the window frame for a <b>wxTHICK_FRAME</b> window.
<b>wxSYS_SMALLICON_X</b>	Recommended width of a small icon (in window captions, and small icon view).
<b>wxSYS_SMALLICON_Y</b>	Recommended height of a small icon (in window captions, and small icon view).
<b>wxSYS_HSCROLL_Y</b>	Height of horizontal scrollbar in pixels.
<b>wxSYS_VSCROLL_X</b>	Width of vertical scrollbar in pixels.
<b>wxSYS_VSCROLL_ARROW_X</b>	Width of arrow bitmap on a vertical scrollbar.
<b>wxSYS_VSCROLL_ARROW_Y</b>	Height of arrow bitmap on a vertical scrollbar.
<b>wxSYS_VTHUMB_Y</b>	Height of vertical scrollbar thumb.
<b>wxSYS_CAPTION_Y</b>	Height of normal caption area.
<b>wxSYS_MENU_Y</b>	Height of single-line menu bar.
<b>wxSYS_NETWORK_PRESENT</b>	1 if there is a network present, 0 otherwise.
<b>wxSYS_PENWINDOWS_PRESENT</b>	1 if PenWindows is installed, 0 otherwise.
<b>wxSYS_SHOW_SOUNDS</b>	Non-zero if the user requires an application to present information visually in situations where it would otherwise present the information only in audible form; zero otherwise.
<b>wxSYS_SWAP_BUTTONS</b>	Non-zero if the meanings of the left and right mouse buttons are swapped; zero otherwise.

*win* is a pointer to the window for which the metric is requested. Specifying the *win* parameter is encouraged, because some metrics on some ports are not supported without one, or they might be capable of reporting better values if given one. If a window does not make sense for a metric, one should still be given, as for example it might determine which displays cursor width is requested with **wxSYS\_CURSOR\_X**.

**wxPython note:** This static method is implemented in Python as a standalone function named `wxSystemSettings_GetMetric`

### **wxSystemSettings::GetScreenType**

**static wxSystemScreenType GetScreenType()**

Returns the screen type. The return value is one of:

<b>wxSYS_SCREEN_NONE</b>	Undefined screen type
<b>wxSYS_SCREEN_TINY</b>	Tiny screen, less than 320x240

<b>wxSYS_SCREEN_PDA</b>	PDA screen, 320x240 or more but less than 640x480
<b>wxSYS_SCREEN_SMALL</b>	Small screen, 640x480 or more but less than 800x600
<b>wxSYS_SCREEN_DESKTOP</b>	Desktop screen, 800x600 or more

## wxTarClassFactory

Class factory for the tar archive format. See the base class for details.

### Derived from

*wxArchiveClassFactory* (p. 58)

### Include files

<wx/tarstrm.h>

### See also

*Archive formats such as zip* (p. 2223)

*Generic archive programming* (p. 2227)

*wxTarEntry* (p. 1599)

*wxTarInputStream* (p. 1603)

*wxTarOutputStream* (p. 1604)

## wxTarEntry

Holds the meta-data for an entry in a tar.

### Derived from

*wxArchiveEntry* (p. 62)

### Include files

<wx/tarstrm.h>

### Data structures

Constants for *GetTypeFlag* (p. 1602):

```
// TypeFlag values
enum {
    wxTAR_REGTYPE    = '0',    // regular file
    wxTAR_LNKTYPE    = '1',    // hard link
    wxTAR_SYMTYPE    = '2',    // symbolic link
    wxTAR_CHRTYPE    = '3',    // character special
    wxTAR_BLKTYPE    = '4',    // block special
    wxTAR_DIRTYPE    = '5',    // directory
    wxTAR_FIFOTYPE   = '6',    // named pipe
}
```

```
        wxTAR_CONTYPE = '7'           // contiguous file
    };
```

### See also

*Archive formats such as zip* (p. 2223)

*wxTarInputStream* (p. 1603)

*wxTarOutputStream* (p. 1604)

### Field availability

The tar format stores all the meta-data for an entry ahead of its data, therefore *GetNextEntry()* (p. 1604) always returns a fully populated *wxTarEntry* object, both when reading from seekable and non-seekable streams.

### wxTarEntry::wxTarEntry

**wxTarEntry(const wxString& name = wxEmptyString, const wxDateTime& dt = wxDateTime::Now(), wxFileOffset size = wxInvalidOffset)**

Constructor. The tar archive format stores the entry's size ahead of the entry's data. Therefore when creating an archive on a non-seekable stream it is necessary to supply the correct size when each entry is created.

**wxTarEntry(const wxTarEntry& entry)**

Copy constructor.

### wxTarEntry::Get/SetAccessTime

**wxDateTime GetAccessTime() const**

**void SetAccessTime(const wxDateTime& dt)**

The entry's access time stamp. See also *wxArchiveEntry::Get/SetDateTime* (p. 62).

### wxTarEntry::Get/SetCreateTime

**wxDateTime GetCreateTime() const**

**void SetCreateTime(const wxDateTime& dt)**

The entry's creation time stamp. See also *wxArchiveEntry::Get/SetDateTime* (p. 62).

### wxTarEntry::Get/SetDevMajor and Get/SetDevMinor

**int GetDevMajor() const**

**int GetDevMinor() const**



```
void SetDevMajor(int dev)
```

```
void SetDevMinor(int dev)
```

OS specific IDs defining a device, these are only meaningful when *TypeFlag* (p. 1602) is set to *wxTAR\_CHRTYPE* or *wxTAR\_BLKTYPE*.

### **wxTarEntry::Get/SetGroupId and Get/SetUserId**

```
int GetGroupId() const
```

```
int GetUserId() const
```

```
void SetGroupId(int id)
```

```
void SetUserId(int id)
```

The user ID and group ID that has *permissions* (p. 1602) over this entry. These values aren't usually useful unless the file will only be restored to the same system it originated from. *Get/SetGroupName* and *Get/SetUserName* (p. 1601) can be used instead.

### **wxTarEntry::Get/SetGroupName and Get/SetUserName**

```
wxString GetGroupName() const
```

```
wxString GetUserName() const
```

```
void SetGroupName(const wxString& group)
```

```
void SetUserName(const wxString& user)
```

The names of the user and group that has *permissions* (p. 1602) over this entry. These are not present in very old tars.

### **wxTarEntry::GetInternalName**

```
wxString GetInternalName() const
```

Returns the entry's filename in the internal format used within the archive. The name can include directory components, i.e. it can be a full path.

The names of directory entries are returned without any trailing path separator. This gives a canonical name that can be used in comparisons.

```
wxString GetInternalName(const wxString& name, wxPathFormat format =  
wxPATH_NATIVE, bool* plsDir = NULL)
```

A static member that translates a filename into the internal format used within the archive. If the third parameter is provided, the bool pointed to is set to indicate whether the name looks like a directory name (i.e. has a trailing path separator).

### **wxTarEntry::Get/SetLinkName**

**wxString GetLinkName() const**

**void SetLinkName(const wxString& link)**

The filename of a previous entry in the tar that this entry is a link to. Only meaningful when *TypeFlag* (p. 1602) is set to *wxTAR\_LNKTYPE* or *wxTAR\_SYMTYPE*.

**wxTarEntry::Get/SetMode**

**int GetMode() const**

**void SetMode(int mode)**

UNIX permission bits for this entry. Giving read, write and execute permissions to the file's *User and Group* (p. 1601) and to others. Symbols are defined for them in `<wx/file.h>`.

```
#define wxS_IRUSR 00400
#define wxS_IWUSR 00200
#define wxS_IXUSR 00100

#define wxS_IRGRP 00040
#define wxS_IWGRP 00020
#define wxS_IXGRP 00010

#define wxS_IROTH 00004
#define wxS_IWOTH 00002
#define wxS_IXOTH 00001
```

**wxTarEntry::Get/SetSize**

**void SetSize(wxFileOffset size)**

**wxFileOffset GetSize() const**

The size of the entry's data in bytes.

The tar archive format stores the entry's size ahead of the entry's data. Therefore when creating an archive on a non-seekable stream it is necessary to supply the correct size when each entry is created. For seekable streams this is not necessary as *wxTarOutputStream* (p. 1604) will attempt to seek back and fix the entry's header when the entry is closed, though it is still more efficient if the size is given beforehand.

**wxTarEntry::Get/SetTypeFlag**

**int GetTypeFlag() const**

**void SetTypeFlag(int type)**

Returns the type of the entry. It should be one of the following:

```
// TypeFlag values
enum {
```

```
    wxTAR_REGTYPE    = '0',        // regular file
    wxTAR_LNKTYPE    = '1',        // hard link
    wxTAR_SYMTYPE    = '2',        // symbolic link
    wxTAR_CHRTYPE    = '3',        // character special
    wxTAR_BLKTYPE    = '4',        // block special
    wxTAR_DIRTYPE    = '5',        // directory
    wxTAR_FIFOTYPE    = '6',        // named pipe
    wxTAR_CONTYPE    = '7',        // contiguous file
};
```

When creating archives use just these values. When reading archives any other values should be treated as `wxTAR_REGTYPE`.

### **wxTarEntry::operator=**

**wxTarEntry& operator operator=(const wxTarEntry& entry)**

Assignment operator.

## **wxTarInputStream**

Input stream for reading tar files.

*GetNextEntry()* (p. 1604) returns an *wxTarEntry* (p. 1599) object containing the meta-data for the next entry in the tar (and gives away ownership). Reading from the *wxTarInputStream* then returns the entry's data. *Eof()* becomes true after an attempt has been made to read past the end of the entry's data. When there are no more entries, *GetNextEntry()* returns NULL and sets *Eof()*.

Tar entries are seekable if the parent stream is seekable. In practice this usually means they are only seekable if the tar is stored as a local file and is not compressed.

### **Derived from**

*wxArchiveInputStream* (p. 64)

### **Include files**

<wx/tarstrm.h>

**Data structures** `typedef wxTarEntry entry_type`

*Archive formats such as zip* (p. 2223)

*wxTarEntry* (p. 1599)

*wxTarOutputStream* (p. 1604)

### **wxTarInputStream::wxTarInputStream**

**wxTarInputStream(wxInputStream& stream, wxMBConv& conv = wxConvLocal)**

**wxTarInputStream(wxInputStream\* stream, wxMBConv& conv = wxConvLocal)**

Constructor. In a Unicode build the second parameter *conv* is used to translate fields from the standard tar header into Unicode. It has no effect on the stream's data. *conv* is only used for the standard tar headers, any pax extended headers are always UTF-8 encoded.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

### **wxTarInputStream::CloseEntry**

**bool CloseEntry()**

Closes the current entry. On a non-seekable stream reads to the end of the current entry first.

### **wxTarInputStream::GetNextEntry**

**wxTarEntry\* GetNextEntry()**

Closes the current entry if one is open, then reads the meta-data for the next entry and returns it in a *wxTarEntry* (p. 1599) object, giving away ownership. The stream is then open and can be read.

### **wxTarInputStream::OpenEntry**

**bool OpenEntry(wxTarEntry& entry)**

Closes the current entry if one is open, then opens the entry specified by the *entry* object. *entry* should be from the same tar file, and the tar should be on a seekable stream.

#### **See also**

*Looking up an archive entry by name* (p. 2225)

## **wxTarOutputStream**

Output stream for writing tar files.

*PutNextEntry()* (p. 1607) is used to create a new entry in the output tar, then the entry's data is written to the *wxTarOutputStream*. Another call to *PutNextEntry()* closes the current entry and begins the next.

#### **Derived from**

*wxArchiveOutputStream* (p. 69)

#### **Include files**

<wx/tarstrm.h>

## Data structures

Constants for the *format* parameter of the *constructor* (p. 1605).

```
// Archive Formats (use wxTAR_PAX, it's backward compatible)
enum wxTarFormat
{
    wxTAR_USTAR,           // POSIX.1-1990 tar format
    wxTAR_PAX              // POSIX.1-2001 tar format
};
```

## See also

*Archive formats such as zip* (p. 2223)

*wxTarEntry* (p. 1599)

*wxTarInputStream* (p. 1603)

## wxTarOutputStream::wxTarOutputStream

**wxTarOutputStream(wxOutputStream& stream, wxTarFormat format = wxTAR\_PAX, wxMBConv& conv = wxConvLocal)**

**wxTarOutputStream(wxOutputStream\* stream, wxTarFormat format = wxTAR\_PAX, wxMBConv& conv = wxConvLocal)**

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

In a Unicode build the third parameter *conv* is used to translate the headers fields into an 8-bit encoding. It has no effect on the stream's data.

When the *format* is *wxTAR\_PAX*, pax extended headers are generated when any header field will not fit the standard tar header block or if it uses any non-ascii characters.

Extended headers are stored as extra 'files' within the tar, and will be extracted as such by any other tar program that does not understand them. The *conv* parameter only affect the standard tar headers, the extended headers are always UTF-8 encoded.

When the *format* is *wxTAR\_USTAR*, no extended headers are generated, and instead a warning message is logged if any header field overflows.

## wxTarOutputStream::~wxTarOutputStream

**~wxTarOutputStream()**

The destructor calls *Close()* (p. 1605) to finish writing the tar if it has not been called already.

## wxTarOutputStream::Close

**bool Close()**

Finishes writing the tar, returning true if successful. Called by the destructor if not called explicitly.

**wxTarOutputStream::CloseEntry****bool CloseEntry()**

Close the current entry. It is called implicitly whenever another new entry is created with *CopyEntry()* (p. 1606) or *PutNextEntry()* (p. 1607), or when the tar is closed.

**wxTarOutputStream::CopyArchiveMetaData****bool CopyArchiveMetaData(wxTarInputStream& s)**

See *wxArchiveOutputStream::CopyArchiveMetaData* (p. 69). For the tar format this function does nothing.

**wxTarOutputStream::CopyEntry****bool CopyEntry(wxTarEntry\* entry, wxTarInputStream& inputStream)**

Takes ownership of *entry* and uses it to create a new entry in the tar. *entry* is then opened in *inputStream* and its contents copied to this stream.

For some other archive formats *CopyEntry()* is much more efficient than transferring the data using *Read()* and *Write()* since it will copy them without decompressing and recompressing them. For tar however it makes no difference.

For tars on seekable streams, *entry* must be from the same tar file as *stream*. For non-seekable streams, *entry* must also be the last thing read from *inputStream*.

**wxTarOutputStream::Get/SetBlockingFactor****int GetBlockingFactor() const****void SetBlockingFactor(int factor)**

The tar is zero padded to round its size up to *BlockingFactor* \* 512 bytes.

Defaults to 10 for *wxTAR\_PAX* and 20 for *wxTAR\_USTAR* (see the *constructor* (p. 1605)), as specified in the POSIX standards.

**wxTarOutputStream::PutNextDirEntry****bool PutNextDirEntry(const wxString& name, const wxDateTime& dt = wxDateTime::Now())**

Create a new directory entry (see *wxArchiveEntry::IsDir()* (p. 64)) with the given name and timestamp.

*PutNextEntry()* (p. 1607) can also be used to create directory entries, by supplying a name with a trailing path separator.

### **wxTarOutputStream::PutNextEntry**

**bool PutNextEntry(wxTarEntry\* entry)**

Takes ownership of *entry* and uses it to create a new entry in the tar.

**bool PutNextEntry(const wxString& name, const wxDateTime& dt = wxDateTime::Now(), wxFileOffset size = wxInvalidOffset)**

Create a new entry with the given name, timestamp and size.

## **wxTaskBarIcon**

This class represents a taskbar icon. A taskbar icon is an icon that appears in the 'system tray' and responds to mouse clicks, optionally with a tooltip above it to help provide information.

### **X Window System Note**

Under X Window System, the window manager must support either the System Tray Protocol by freedesktop.org

(<http://freedesktop.org/Standards/systemtray-spec>)(WMs used by modern desktop environments such as GNOME >= 2, KDE >= 3 and XFCE >= 4 all do) or the older methods used in GNOME 1.2 and KDE 1 and 2. If it doesn't, the icon will appear as a toplevel window on user's desktop.

Because not all window managers have system tray, there's no guarantee that *wxTaskBarIcon* will work correctly under X Window System and so the applications should use it only as an optional component of their user interface. The user should be required to explicitly enable the taskbar icon on Unix, it shouldn't be on by default.

### **Derived from**

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### **Include files**

<wx/taskbar.h>

### **Event handling**

To process input from a taskbar icon, use the following event handler macros to direct input to member functions that take a *wxTaskBarIconEvent* argument. Note that not all ports are required to send these events and so it's better to override *CreatePopupMenu* (p. 1608) if all that the application does is that it shows a popup menu in reaction to mouse click.

**EVT\_TASKBAR\_MOVE(func)**

Process a *wxEVT\_TASKBAR\_MOVE* event.

<b>EVT_TASKBAR_LEFT_DOWN(func)</b>	Process a wxEVT_TASKBAR_LEFT_DOWN event.
<b>EVT_TASKBAR_LEFT_UP(func)</b>	Process a wxEVT_TASKBAR_LEFT_UP event.
<b>EVT_TASKBAR_RIGHT_DOWN(func)</b>	Process a wxEVT_TASKBAR_RIGHT_DOWN event.
<b>EVT_TASKBAR_RIGHT_UP(func)</b>	Process a wxEVT_TASKBAR_RIGHT_UP event.
<b>EVT_TASKBAR_LEFT_DCLICK(func)</b>	Process a wxEVT_TASKBAR_LEFT_DCLICK event.
<b>EVT_TASKBAR_RIGHT_DCLICK(func)</b>	Process a wxEVT_TASKBAR_RIGHT_DCLICK event.
<b>EVT_TASKBAR_CLICK(func)</b>	This is a synonym for either EVT_TASKBAR_RIGHT_DOWN or UP depending on the platform, use this event macro to catch the event which should result in the menu being displayed on the current platform.

### **wxTaskBarIcon::wxTaskBarIcon**

**wxTaskBarIcon()**

Default constructor.

### **wxTaskBarIcon::~~wxTaskBarIcon**

**~wxTaskBarIcon()**

Destroys the wxTaskBarIcon object, removing the icon if not already removed.

### **wxTaskBarIcon::CreatePopupMenu**

**virtual wxMenu\* CreatePopupMenu()**

This method is called by the library when the user requests popup menu (on Windows and Unix platforms, this is when the user right-clicks the icon). Override this function in order to provide popup menu associated with the icon.

If CreatePopupMenu returns NULL (this happens by default), no menu is shown, otherwise the menu is displayed and then deleted by the library as soon as the user dismisses it. The events can be handled by a class derived from wxTaskBarIcon.

### **wxTaskBarIcon::IsIconInstalled**

**bool IsIconInstalled()**



Returns true if *SetIcon* (p. 1609) was called with no subsequent *RemoveIcon* (p. 1609).

### **wxTaskBarIcon::IsOk**

**bool IsOk()**

Returns true if the object initialized successfully.

### **wxTaskBarIcon::PopupMenu**

**bool PopupMenu(wxMenu\* menu)**

Pops up a menu at the current mouse position. The events can be handled by a class derived from *wxTaskBarIcon*.

#### **Note**

It is recommended to override *CreatePopupMenu* (p. 1608) callback instead of calling this method from event handler, because some ports (e.g. *wxCocoa*) may not implement *PopupMenu* and mouse click events at all.

### **wxTaskBarIcon::RemoveIcon**

**bool RemoveIcon()**

Removes the icon previously set with *SetIcon* (p. 1609).

### **wxTaskBarIcon::SetIcon**

**bool SetIcon(const wxIcon& icon, const wxString& tooltip)**

Sets the icon, and optional tooltip text.

## **wxTCPClient**

A *wxTCPClient* object represents the client part of a client-server conversation. It emulates a DDE-style protocol, but uses TCP/IP which is available on most platforms.

A DDE-based implementation for Windows is available using *wxDDEClient* (p. 477).

To create a client which can communicate with a suitable server, you need to derive a class from *wxTCPConnection* and another from *wxTCPClient*. The custom *wxTCPConnection* class will intercept communications in a 'conversation' with a server, and the custom *wxTCPClient* is required so that a user-overridden *wxTCPClient::OnMakeConnection* (p. 1610) member can return a *wxTCPConnection* of the required class, when a connection is made.

#### **Derived from**

*wxClientBase*

*wxObject* (p. 1148)

**Include files**

<wx/sckipc.h>

**See also**

*wxTCPServer* (p. 1614), *wxTCPConnection* (p. 1610), *Interprocess communications overview* (p. 2175)

**wxTCPClient::wxTCPClient**

**wxTCPClient()**

Constructs a client object.

**wxTCPClient::MakeConnection**

**wxConnectionBase \* MakeConnection(const wxString& host, const wxString& service, const wxString& topic)**

Tries to make a connection with a server specified by the host (a machine name under Unix), service name (must contain an integer port number under Unix), and a topic string. If the server allows a connection, a *wxTCPConnection* object will be returned. The type of *wxTCPConnection* returned can be altered by overriding the *wxTCPClient::OnMakeConnection* (p. 1610) member to return your own derived connection object.

**wxTCPClient::OnMakeConnection**

**wxConnectionBase \* OnMakeConnection()**

The type of *wxTCPConnection* (p. 1610) returned from a *wxTCPClient::MakeConnection* (p. 1610) call can be altered by deriving the **OnMakeConnection** member to return your own derived connection object. By default, a *wxTCPConnection* object is returned.

The advantage of deriving your own connection class is that it will enable you to intercept messages initiated by the server, such as *wxTCPConnection::OnAdvise* (p. 1612). You may also want to store application-specific data in instances of the new class.

**wxTCPClient::ValidHost**

**bool ValidHost(const wxString& host)**

Returns true if this is a valid host name, false otherwise.

**wxTCPConnection**

A *wxTCPClient* object represents the connection between a client and a server. It

emulates a DDE-style protocol, but uses TCP/IP which is available on most platforms.

A DDE-based implementation for Windows is available using *wxDDEConnection* (p. 478).

A *wxTCPConnection* object can be created by making a connection using a *wxTCPClient* (p. 1609) object, or by the acceptance of a connection by a *wxTCPServer* (p. 1614) object. The bulk of a conversation is controlled by calling members in a **wxTCPConnection** object or by overriding its members.

An application should normally derive a new connection class from *wxTCPConnection*, in order to override the communication event handlers to do something interesting.

### Derived from

*wxConnectionBase*  
*wxObject* (p. 1148)

### Include files

<wx/sckipc.h>

### Types

*wxIPCFormat* is defined as follows:

```
enum wxIPCFormat
{
    wxIPC_INVALID =          0,
    wxIPC_TEXT =             1, /* CF_TEXT */
    wxIPC_BITMAP =           2, /* CF_BITMAP */
    wxIPC_METAFILE =         3, /* CF_METAFILEPICT */
    wxIPC_SYLK =              4,
    wxIPC_DIF =               5,
    wxIPC_TIFF =              6,
    wxIPC_OEMTEXT =           7, /* CF_OEMTEXT */
    wxIPC_DIB =               8, /* CF_DIB */
    wxIPC_PALETTE =           9,
    wxIPC_PENDATA =           10,
    wxIPC_RIFF =              11,
    wxIPC_WAVE =              12,
    wxIPC_UNICODETEXT =       13,
    wxIPC_ENHMETAFILE =       14,
    wxIPC_FILENAME =          15, /* CF_HDROP */
    wxIPC_LOCALE =            16,
    wxIPC_PRIVATE =           20
};
```

### See also

*wxTCPClient* (p. 1609), *wxTCPServer* (p. 1614), *Interprocess communications overview* (p. 2175)

### wxTCPConnection::wxTCPConnection

**wxTCPConnection()****wxTCPConnection(char\* buffer, int size)**

Constructs a connection object. If no user-defined connection object is to be derived from `wxTCPConnection`, then the constructor should not be called directly, since the default connection object will be provided on requesting (or accepting) a connection. However, if the user defines his or her own derived connection object, the `wxTCPServer::OnAcceptConnection` (p. 1615) and/or `wxTCPClient::OnMakeConnection` (p. 1610) members should be replaced by functions which construct the new connection object. If the arguments of the `wxTCPConnection` constructor are void, then a default buffer is associated with the connection. Otherwise, the programmer must provide a buffer and size of the buffer for the connection object to use in transactions.

**wxTCPConnection::Advise**

**bool Advise(const wxString& item, char\* data, int size = -1, wxIPCFormat format = wxCF\_TEXT)**

Called by the server application to advise the client of a change in the data associated with the given item. Causes the client connection's `wxTCPConnection::OnAdvise` (p. 1612) member to be called. Returns true if successful.

**wxTCPConnection::Execute**

**bool Execute(char\* data, int size = -1, wxIPCFormat format = wxCF\_TEXT)**

Called by the client application to execute a command on the server. Can also be used to transfer arbitrary data to the server (similar to `wxTCPConnection::Poke` (p. 1613) in that respect). Causes the server connection's `wxTCPConnection::OnExecute` (p. 1613) member to be called. Returns true if successful.

**wxTCPConnection::Disconnect**

**bool Disconnect()**

Called by the client or server application to disconnect from the other program; it causes the `wxTCPConnection::OnDisconnect` (p. 1613) 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.

**wxTCPConnection::OnAdvise**

**virtual bool OnAdvise(const wxString& topic, const wxString& item, char\* data, int size, wxIPCFormat format)**

Message sent to the client application when the server notifies it of a change in the data associated with the given item.

**wxTCPConnection::OnDisconnect****virtual bool OnDisconnect()**

Message sent to the client or server application when the other application notifies it to delete the connection. Default behaviour is to delete the connection object.

**wxTCPConnection::OnExecute****virtual bool OnExecute(const wxString& topic, char\* data, int size, wxIPCFormat format)**

Message sent to the server application when the client notifies it to execute the given data. Note that there is no item associated with this message.

**wxTCPConnection::OnPoke****virtual bool OnPoke(const wxString& topic, const wxString& item, char\* data, int size, wxIPCFormat format)**

Message sent to the server application when the client notifies it to accept the given data.

**wxTCPConnection::OnRequest****virtual char\* OnRequest(const wxString& topic, const wxString& item, int \*size, wxIPCFormat format)**

Message sent to the server application when the client calls *wxTCPConnection::Request* (p. 1614). The server should respond by returning a character string from **OnRequest**, or NULL to indicate no data.

**wxTCPConnection::OnStartAdvise****virtual bool OnStartAdvise(const wxString& topic, const wxString& item)**

Message sent to the server application by the client, when the client wishes to start an 'advise loop' for the given topic and item. The server can refuse to participate by returning false.

**wxTCPConnection::OnStopAdvise****virtual bool OnStopAdvise(const wxString& topic, const wxString& item)**

Message sent to the server application by the client, when the client wishes to stop an 'advise loop' for the given topic and item. The server can refuse to stop the advise loop by returning false, although this doesn't have much meaning in practice.

**wxTCPConnection::Poke****bool Poke(const wxString& item, char\* data, int size = -1, wxIPCFormat format =**

*wxCf\_Text()*

Called by the client application to poke data into the server. Can be used to transfer arbitrary data to the server. Causes the server connection's *wxTCPConnection::OnPoke* (p. 1613) member to be called. Returns true if successful.

### **wxTCPConnection::Request**

**char\* Request(const wxString& item, int \*size, wxIPCFormat format = wxIPC\_TEXT)**

Called by the client application to request data from the server. Causes the server connection's *wxTCPConnection::OnRequest* (p. 1613) member to be called. Returns a character string (actually a pointer to the connection's buffer) if successful, NULL otherwise.

### **wxTCPConnection::StartAdvise**

**bool StartAdvise(const wxString& item)**

Called by the client application to ask if an advise loop can be started with the server. Causes the server connection's *wxTCPConnection::OnStartAdvise* (p. 1613) member to be called. Returns true if the server okays it, false otherwise.

### **wxTCPConnection::StopAdvise**

**bool StopAdvise(const wxString& item)**

Called by the client application to ask if an advise loop can be stopped. Causes the server connection's *wxTCPConnection::OnStopAdvise* (p. 1613) member to be called. Returns true if the server okays it, false otherwise.

## **wxTCPServer**

A *wxTCPServer* object represents the server part of a client-server conversation. It emulates a DDE-style protocol, but uses TCP/IP which is available on most platforms.

A DDE-based implementation for Windows is available using *wxDDEServer* (p. 482).

### **Derived from**

*wxServerBase*  
*wxObject* (p. 1148)

### **Include files**

<wx/sckipc.h>

### **See also**

*wxTCPClient* (p. 1609), *wxTCPConnection* (p. 1610), *IPC overview* (p. 2175)

## **wxTCPServer::wxTCPServer**

### **wxTCPServer()**

Constructs a server object.

## **wxTCPServer::Create**

### **bool Create(const wxString& service)**

Registers the server using the given service name. Under Unix, the string must contain an integer id which is used as an Internet port number. false is returned if the call failed (for example, the port number is already in use).

## **wxTCPServer::OnAcceptConnection**

### **virtual wxConnectionBase \* OnAcceptConnection(const wxString& topic)**

When a client calls **MakeConnection**, the server receives the message and this member is called. The application should derive a member to intercept this message and return a connection object of either the standard wxTCPConnection type, or of a user-derived type. If the topic is "STDIO", the application may wish to refuse the connection. Under Unix, when a server is created the OnAcceptConnection message is always sent for standard input and output.

## **wxTempFile**

wxTempFile provides a relatively safe way to replace the contents of the existing file. The name is explained by the fact that it may be also used as just a temporary file if you don't replace the old file contents.

Usually, when a program replaces the contents of some file it first opens it for writing, thus losing all of the old data and then starts recreating it. This approach is not very safe because during the regeneration of the file bad things may happen: the program may find that there is an internal error preventing it from completing file generation, the user may interrupt it (especially if file generation takes long time) and, finally, any other external interrupts (power supply failure or a disk error) will leave you without either the original file or the new one.

wxTempFile addresses this problem by creating a temporary file which is meant to replace the original file - but only after it is fully written. So, if the user interrupts the program during the file generation, the old file won't be lost. Also, if the program discovers itself that it doesn't want to replace the old file there is no problem - in fact, wxTempFile will **not** replace the old file by default, you should explicitly call *Commit* (p. 1617) to do it. Calling *Discard* (p. 1617) explicitly discards any modifications: it closes and deletes the temporary file and leaves the original file unchanged. If you don't call neither of *Commit*() and *Discard*(), the destructor will call *Discard*() automatically.

To summarize: if you want to replace another file, create an instance of `wxTempFile` passing the name of the file to be replaced to the constructor (you may also use default constructor and pass the file name to `Open` (p. 1616)). Then you can *write* (p. 1617) to `wxTempFile` using *wxFile* (p. 591)-like functions and later call `Commit()` to replace the old file (and close this one) or call `Discard()` to cancel the modifications.

**Derived from**

No base class

**Include files**

<wx/file.h>

**See also:**

*wxFile* (p. 591)

*wxTempFileOutputStream* (p. 1618)

**wxTempFile::wxTempFile**

**wxTempFile()**

Default constructor - *Open* (p. 1616) must be used to open the file.

**wxTempFile::wxTempFile**

**wxTempFile(const wxString& strName)**

Associates `wxTempFile` with the file to be replaced and opens it. You should use *IsOpened* (p. 1616) to verify if the constructor succeeded.

**wxTempFile::Open**

**bool Open(const wxString& strName)**

Open the temporary file, returns `true` on success, `false` if an error occurred.

*strName* is the name of file to be replaced. The temporary file is always created in the directory where *strName* is. In particular, if *strName* doesn't include the path, it is created in the current directory and the program should have write access to it for the function to succeed.

**wxTempFile::IsOpened**

**bool IsOpened() const**

Returns `true` if the file was successfully opened.

**wxTempFile::Length**



**wxFileOffset Length() const**

Returns the length of the file.

**wxTempFile::Seek**

**wxFileOffset Seek(wxFileOffset ofs, wxSeekMode mode = wxFromStart)**

Seeks to the specified position.

**wxTempFile::Tell****wxFileOffset Tell() const**

Returns the current position or `wxInvalidOffset` if file is not opened or if another error occurred.

**wxTempFile::Write**

**bool Write(const void \*p, size\_t n)**

Write to the file, return `true` on success, `false` on failure.

**wxTempFile::Write**

**bool Write(const wxString& str, wxMBConv& conv = wxConvLibc)**

Write to the file, return `true` on success, `false` on failure.

The second argument is only meaningful in Unicode build of `wxWidgets` when `conv` is used to convert `str` to multibyte representation.

**wxTempFile::Commit**

**bool Commit()**

Validate changes: deletes the old file of name `m_strName` and renames the new file to the old name. Returns `true` if both actions succeeded. If `false` is returned it may unfortunately mean two quite different things: either that either the old file couldn't be deleted or that the new file couldn't be renamed to the old name.

**wxTempFile::Discard**

**void Discard()**

Discard changes: the old file contents is not changed, temporary file is deleted.

**wxTempFile::~wxTempFile**

**~wxTempFile()**

Destructor calls *Discard()* (p. 1617) if temporary file is still opened.

## **wxTempFileOutputStream**

*wxTempFileOutputStream* is an output stream based on *wxTempFile* (p. 1615). It provides a relatively safe way to replace the contents of the existing file.

### **Derived from**

*wxOutputStream* (p. 1156)

### **Include files**

<wx/wfstream.h>

### **See also**

*wxTempFile* (p. 1615)

## **wxTempFileOutputStream::wxTempFileOutputStream**

**wxTempFileOutputStream(const wxString& fileName)**

Associates *wxTempFileOutputStream* with the file to be replaced and opens it. You should use *IsOk* (p. 1546) to verify if the constructor succeeded.

Call *Commit()* (p. 1618) or *Close()* (p. 1156) to replace the old file and close this one. Calling *Discard()* (p. 1618) (or allowing the destructor to do it) will discard the changes.

## **wxTempFileOutputStream::Commit**

**bool Commit()**

Validate changes: deletes the old file of the given name and renames the new file to the old name. Returns *true* if both actions succeeded. If *false* is returned it may unfortunately mean two quite different things: either that either the old file couldn't be deleted or that the new file couldn't be renamed to the old name.

## **wxTempFileOutputStream::Discard**

**void Discard()**

Discard changes: the old file contents are not changed, the temporary file is deleted.

## **wxTextAttr**

*wxTextAttr* represents the character and paragraph attributes, or style, for a range of text in a *wxTextCtrl* (p. 1633).

When setting up a `wxTextAttr` object, pass a bitlist mask to `SetFlags` to indicate which style elements should be changed. As a convenience, when you call a setter such as `SetFont`, the relevant bit will be set.

### Derived from

No base class

### Include files

<wx/textctrl.h>

### Typedefs

`wxTextPos` is the type containing the index of a position in a text control. `wxTextCoord` contains the index of a column or a row in the control.

Note that although both of these types should probably have been unsigned, due to backwards compatibility reasons, are defined as `long` currently. Their use (instead of plain `long`) is still encouraged as it makes the code more readable.

### Constants

The following values can be passed to `SetAlignment` to determine paragraph alignment.

```
enum wxTextAttrAlignment
{
    wxTEXT_ALIGNMENT_DEFAULT,
    wxTEXT_ALIGNMENT_LEFT,
    wxTEXT_ALIGNMENT_CENTRE,
    wxTEXT_ALIGNMENT_CENTER = wxTEXT_ALIGNMENT_CENTRE,
    wxTEXT_ALIGNMENT_RIGHT,
    wxTEXT_ALIGNMENT_JUSTIFIED
};
```

These values are passed in a bitlist to `SetFlags` to determine what attributes will be considered when setting the attributes for a text control.

```
#define wxTEXT_ATTR_TEXT_COLOUR          0x0001
#define wxTEXT_ATTR_BACKGROUND_COLOUR    0x0002
#define wxTEXT_ATTR_FONT_FACE            0x0004
#define wxTEXT_ATTR_FONT_SIZE            0x0008
#define wxTEXT_ATTR_FONT_WEIGHT          0x0010
#define wxTEXT_ATTR_FONT_ITALIC          0x0020
#define wxTEXT_ATTR_FONT_UNDERLINE       0x0040
#define wxTEXT_ATTR_FONT \
    wxTEXT_ATTR_FONT_FACE | wxTEXT_ATTR_FONT_SIZE | \
wxTEXT_ATTR_FONT_WEIGHT | \
    wxTEXT_ATTR_FONT_ITALIC | wxTEXT_ATTR_FONT_UNDERLINE
#define wxTEXT_ATTR_ALIGNMENT            0x0080
#define wxTEXT_ATTR_LEFT_INDENT          0x0100
#define wxTEXT_ATTR_RIGHT_INDENT         0x0200
#define wxTEXT_ATTR_TABS                  0x0400
```

**wxTextAttr::wxTextAttr****wxTextAttr()**

**wxTextAttr**(const wxColour& colText, const wxColour& colBack = wxNullColour, const wxFont& font = wxNullFont, wxTextAttrAlignment alignment = wxTEXT\_ALIGNMENT\_DEFAULT)

The constructors initialize one or more of the text foreground colour, background colour, font, and alignment. The values not initialized in the constructor can be set later, otherwise *wxTextCtrl::SetStyle* (p. 1650) will use the default values for them.

**wxTextAttr::GetAlignment****wxTextAttrAlignment GetAlignment() const**

Returns the paragraph alignment.

**wxTextAttr::GetBackgroundColour****const wxColour& GetBackgroundColour() const**

Return the background colour specified by this attribute.

**wxTextAttr::GetFont****const wxFont& GetFont() const**

Return the text font specified by this attribute.

**wxTextAttr::GetLeftIndent****int GetLeftIndent() const**

Returns the left indent in tenths of a millimetre.

**wxTextAttr::GetLeftSubIndent****int GetLeftSubIndent() const**

Returns the left sub indent for all lines but the first line in a paragraph in tenths of a millimetre.

**wxTextAttr::GetRightIndent****int GetRightIndent() const**

Returns the right indent in tenths of a millimetre.

**wxTextAttr::GetTabs****const wxArrayInt& GetTabs() const**

Returns the array of integers representing the tab stops. Each array element specifies the tab stop in tenths of a millimetre.

**wxTextAttr::GetTextColour****const wxColour& GetTextColour() const**

Return the text colour specified by this attribute.

**wxTextAttr::HasAlignment****bool HasAlignment() const**

Returns `true` if this style specifies the text alignment.

**wxTextAttr::HasBackgroundColour****bool HasBackgroundColour() const**

Returns `true` if this style specifies the background colour to use.

**wxTextAttr::HasFont****bool HasFont() const**

Returns `true` if this style specifies the font to use.

**wxTextAttr::HasLeftIndent****bool HasLeftIndent() const**

Returns `true` if this style specifies the left indent.

**wxTextAttr::HasRightIndent****bool HasRightIndent() const**

Returns `true` if this style specifies the right indent.

**wxTextAttr::HasTabs****bool HasTabs() const**

Returns `true` if this style specifies any tabstops.

**wxTextAttr::HasTextColour****bool HasTextColour() const**

Returns `true` if this style specifies the foreground colour to use.

**wxTextAttr::GetFlags****long GetFlags()**

Returns a bitlist indicating which attributes will be set.

**wxTextAttr::IsDefault****bool IsDefault() const**

Returns `true` if this style specifies any non-default attributes.

**wxTextAttr::Merge****void Merge(const wxTextAttr& overlay)**

Copies all defined/valid properties from *overlay* to current object.

**static wxTextAttr Merge(const wxTextAttr& base, const wxTextAttr& overlay)**

Creates a new `wxTextAttr` which is a merge of *base* and *overlay*. Properties defined in *overlay* take precedence over those in *base*. Properties undefined/invalid in both are undefined in the result.

**wxTextAttr::SetAlignment****void SetAlignment(wxTextAttrAlignment alignment)**

Sets the paragraph alignment.

**wxTextAttr::SetBackgroundColour****void SetBackgroundColour(const wxColour& colour)**

Sets the background colour.

**wxTextAttr::SetFlags****void SetFlags(long flags)**

Pass a bitlist indicating which attributes will be set.

**wxTextAttr::SetFont**

**void SetFont(const wxFont& font)**

Sets the text font.

**wxTextAttr::SetLeftIndent**

**void SetLeftIndent(int indent, int subIndent = 0)**

Sets the left indent in tenths of a millimetre. subIndent sets the indent for all lines but the first line in a paragraph relative to the first line.

**wxTextAttr::SetRightIndent**

**void SetRightIndent(int indent)**

Sets the right indent in tenths of a millimetre.

**wxTextAttr::SetTabs**

**void SetTabs(const wxArrayInt& tabs)**

Sets the array of integers representing the tab stops. Each array element specifies the tab stop in tenths of a millimetre.

**wxTextAttr::SetTextColour**

**void SetTextColour(const wxColour& colour)**

Sets the text colour.

## **wxTextAttrEx**

wxTextAttrEx is an extended version of wxTextAttr with more paragraph attributes. Currently it is only used with *wxRichTextCtrl* (p. 1316).

It is intended that eventually, the members of wxTextAttrEx will be folded into wxTextAttr, and wxTextAttr will be the official cross-platform API for text controls that support attributes. However, for now, wxTextAttrEx is provided as a means of enabling extra functionality in wxRichTextCtrl, while retaining some compatibility with the wxTextAttr API.

The most efficient method of accessing wxRichTextCtrl functionality is a third attribute class, *wxRichTextAttr* (p. 1281), which optimizes its storage to allow it to be used for implementing objects in a buffer, as well as access to that buffer.

This section only documents the additional members; see *wxTextAttr* (p. 1618) for the remaining functions.

### **Derived from**

*wxTextAttr* (p. 1618)

**Include files**

<wx/richtext/richtextbuffer.h>

**Constants**

The following values can be passed to `SetAlignment` to determine paragraph alignment.

```
enum wxTextAttrAlignment
{
    wxTEXT_ALIGNMENT_DEFAULT,
    wxTEXT_ALIGNMENT_LEFT,
    wxTEXT_ALIGNMENT_CENTRE,
    wxTEXT_ALIGNMENT_CENTER = wxTEXT_ALIGNMENT_CENTRE,
    wxTEXT_ALIGNMENT_RIGHT,
    wxTEXT_ALIGNMENT_JUSTIFIED
};
```

These values are passed in a bitlist to `SetFlags` to determine what attributes will be considered when setting the attributes for a text control.

```
// Standard wxTextAttr constants

#define wxTEXT_ATTR_TEXT_COLOUR          0x0001
#define wxTEXT_ATTR_BACKGROUND_COLOUR    0x0002
#define wxTEXT_ATTR_FONT_FACE            0x0004
#define wxTEXT_ATTR_FONT_SIZE            0x0008
#define wxTEXT_ATTR_FONT_WEIGHT          0x0010
#define wxTEXT_ATTR_FONT_ITALIC          0x0020
#define wxTEXT_ATTR_FONT_UNDERLINE       0x0040
#define wxTEXT_ATTR_FONT \
    wxTEXT_ATTR_FONT_FACE | wxTEXT_ATTR_FONT_SIZE | \
wxTEXT_ATTR_FONT_WEIGHT | \
| wxTEXT_ATTR_FONT_ITALIC | wxTEXT_ATTR_FONT_UNDERLINE
#define wxTEXT_ATTR_ALIGNMENT            0x0080
#define wxTEXT_ATTR_LEFT_INDENT           0x0100
#define wxTEXT_ATTR_RIGHT_INDENT          0x0200
#define wxTEXT_ATTR_TABS                  0x0400

// Extra formatting flags not in wxTextAttr

#define wxTEXT_ATTR_PARA_SPACING_AFTER    0x00000800
#define wxTEXT_ATTR_PARA_SPACING_BEFORE  0x00001000
#define wxTEXT_ATTR_LINE_SPACING          0x00002000
#define wxTEXT_ATTR_CHARACTER_STYLE_NAME  0x00004000
#define wxTEXT_ATTR_PARAGRAPH_STYLE_NAME  0x00008000
#define wxTEXT_ATTR_LIST_STYLE_NAME       0x00010000
#define wxTEXT_ATTR_BULLET_STYLE         0x00020000
#define wxTEXT_ATTR_BULLET_NUMBER        0x00040000
#define wxTEXT_ATTR_BULLET_TEXT          0x00080000
#define wxTEXT_ATTR_BULLET_NAME          0x00100000
#define wxTEXT_ATTR_URL                   0x00200000
#define wxTEXT_ATTR_PAGE_BREAK            0x00400000
#define wxTEXT_ATTR_EFFECTS               0x00800000
```



```
#define wxTEXT_ATTR_OUTLINE_LEVEL          0x01000000
```

The following styles can be passed to `wxTextAttrEx::SetBulletStyle`:

```
#define wxTEXT_ATTR_BULLET_STYLE_NONE          0x00000000
#define wxTEXT_ATTR_BULLET_STYLE_ARABIC        0x00000001
#define wxTEXT_ATTR_BULLET_STYLE_LETTERS_UPPER 0x00000002
#define wxTEXT_ATTR_BULLET_STYLE_LETTERS_LOWER 0x00000004
#define wxTEXT_ATTR_BULLET_STYLE_ROMAN_UPPER   0x00000008
#define wxTEXT_ATTR_BULLET_STYLE_ROMAN_LOWER   0x00000010
#define wxTEXT_ATTR_BULLET_STYLE_SYMBOL        0x00000020
#define wxTEXT_ATTR_BULLET_STYLE_BITMAP        0x00000040
#define wxTEXT_ATTR_BULLET_STYLE_PARENTHESSES 0x00000080
#define wxTEXT_ATTR_BULLET_STYLE_PERIOD        0x00000100
#define wxTEXT_ATTR_BULLET_STYLE_STANDARD      0x00000200
#define wxTEXT_ATTR_BULLET_STYLE_RIGHT_PARENTHESIS 0x00000400
#define wxTEXT_ATTR_BULLET_STYLE_OUTLINE      0x00000800
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_LEFT   0x00000000
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_RIGHT  0x00000100
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_CENTRE 0x00002000
```

Of these, `wxTEXT_ATTR_BULLET_STYLE_BITMAP` is unimplemented.

The following constants can be passed to `wxTextAttrEx::SetLineSpacing`:

```
#define wxTEXT_ATTR_LINE_SPACING_NORMAL        10
#define wxTEXT_ATTR_LINE_SPACING_HALF          15
#define wxTEXT_ATTR_LINE_SPACING_TWICE         20
```

The following styles can be passed to `wxTextAttrEx::SetTextEffects`:

```
#define wxTEXT_ATTR_EFFECT_NONE          0x00000000
#define wxTEXT_ATTR_EFFECT_CAPITALS      0x00000001
#define wxTEXT_ATTR_EFFECT_SMALL_CAPITALS 0x00000002
#define wxTEXT_ATTR_EFFECT_STRIKETHROUGH 0x00000004
#define wxTEXT_ATTR_EFFECT_DOUBLE_STRIKETHROUGH 0x00000008
#define wxTEXT_ATTR_EFFECT_SHADOW        0x00000010
#define wxTEXT_ATTR_EFFECT_EMBOSS        0x00000020
#define wxTEXT_ATTR_EFFECT_OUTLINE       0x00000040
#define wxTEXT_ATTR_EFFECT_ENGRAVE       0x00000080
#define wxTEXT_ATTR_EFFECT_SUPERSCRIPT   0x00000100
#define wxTEXT_ATTR_EFFECT_SUBSCRIPT     0x00000200
```

Of these, only `wxTEXT_ATTR_EFFECT_CAPITALS` and `wxTEXT_ATTR_EFFECT_STRIKETHROUGH` are implemented.

**See also**

*wxTextAttr* (p. 1618), *wxRichTextAttr* (p. 1281), *wxRichTextCtrl* (p. 1316)

## **wxTextAttrEx::wxTextAttrEx**

**wxTextAttrEx()**

**wxTextAttrEx(const wxTextAttrEx& attr)**

Constructors.

## **wxTextAttrEx::GetBulletFont**

**const wxString& GetBulletFont() const**

Returns a string containing the name of the font associated with the bullet symbol. Only valid for attributes with `wxTEXT_ATTR_BULLET_SYMBOL`.

## **wxTextAttrEx::GetBulletName**

**const wxString& GetBulletName() const**

Returns the standard bullet name, applicable if the bullet style is `wxTEXT_ATTR_BULLET_STYLE_STANDARD`. Currently the following standard bullet names are supported:

- `standard/circle`
- `standard/square`
- `standard/diamond`
- `standard/triangle`

If you wish your application to support further bullet graphics, you can derive a class from `wxRichTextRenderer` or `wxRichTextStdRenderer`, override `DrawStandardBullet` and `EnumerateStandardBulletNames`, and set an instance of the class using *wxRichTextBuffer::SetRenderer* (p. 1314).

## **wxTextAttrEx::GetBulletNumber**

**int GetBulletNumber() const**

Returns the bullet number.

## **wxTextAttrEx::GetBulletStyle**

**int GetBulletStyle() const**

Returns the bullet style. See *wxTextAttrEx::SetBulletStyle* (p. 1631) for a list of available

styles.

**wxTextAttrEx::GetBulletText****const wxString& GetBulletText() const**

Returns the bullet text, which could be a symbol, or (for example) cached outline text.

**wxTextAttrEx::GetCharacterStyleName****const wxString& GetCharacterStyleName() const**

Returns the name of the character style.

**wxTextAttrEx::GetLineSpacing****int GetLineSpacing() const**

Returns the line spacing value, one of wxTEXT\_ATTR\_LINE\_SPACING\_NORMAL, wxTEXT\_ATTR\_LINE\_SPACING\_HALF, and wxTEXT\_ATTR\_LINE\_SPACING\_TWICE.

**wxTextAttrEx::GetListStyleName****const wxString& GetListStyleName() const**

Returns the name of the list style.

**wxTextAttrEx::GetOutlineLevel****bool GetOutlineLevel() const**

Returns the outline level.

**wxTextAttrEx::GetParagraphSpacingAfter****int GetParagraphSpacingAfter() const**

Returns the space in tenths of a millimeter after the paragraph.

**wxTextAttrEx::GetParagraphSpacingBefore****int GetParagraphSpacingBefore() const**

Returns the space in tenths of a millimeter before the paragraph.

**wxTextAttrEx::GetParagraphStyleName****const wxString& GetParagraphStyleName() const**

Returns the name of the paragraph style.

**wxTextAttrEx::GetTextEffectFlags****int GetTextEffectFlags() const**

Returns the text effect bits of interest. See *wxTextAttr::SetFlags* (p. 1622) for further information.

**wxTextAttrEx::GetTextEffects****int GetTextEffects() const**

Returns the text effects, a bit list of styles. See *wxTextAttrEx::SetTextEffects* (p. 1632) for details.

**wxTextAttrEx::GetURL****const wxString& GetURL() const**

Returns the URL for the content. Content with `wxTEXT_ATTR_URL` style causes `wxRichTextCtrl` to show a hand cursor over it, and `wxRichTextCtrl` generates a `wxTextUrlEvent` when the content is clicked.

**wxTextAttrEx::HasBulletName****bool HasBulletName() const**

Returns `true` if the attribute object specifies a standard bullet name.

**wxTextAttrEx::HasBulletNumber****bool HasBulletNumber() const**

Returns `true` if the attribute object specifies a bullet number.

**wxTextAttrEx::HasBulletStyle****bool HasBulletStyle() const**

Returns `true` if the attribute object specifies a bullet style.

**wxTextAttrEx::HasBulletText****bool HasBulletText() const**

Returns `true` if the attribute object specifies bullet text (usually containing a symbol).

**wxTextAttrEx::HasCharacterStyleName****bool HasCharacterStyleName() const**

Returns `true` if the attribute object specifies a character style name.

**wxTextAttrEx::HasLineSpacing****bool HasLineSpacing() const**

Returns `true` if the attribute object specifies line spacing.

**wxTextAttrEx::HasListStyleName****bool HasListStyleName() const**

Returns `true` if the attribute object specifies a list style name.

**wxTextAttrEx::HasOutlineLevel****bool HasOutlineLevel() const**

Returns `true` if the attribute object specifies an outline level.

**wxTextAttrEx::HasPageBreak****bool HasPageBreak() const**

Returns `true` if the attribute object specifies a page break before this paragraph.

**wxTextAttrEx::HasParagraphSpacingAfter****bool HasParagraphSpacingAfter() const**

Returns `true` if the attribute object specifies spacing after a paragraph.

**wxTextAttrEx::HasParagraphSpacingBefore****bool HasParagraphSpacingBefore() const**

Returns `true` if the attribute object specifies spacing before a paragraph.

**wxTextAttrEx::HasParagraphStyleName****bool HasParagraphStyleName() const**

Returns `true` if the attribute object specifies a paragraph style name.

**wxTextAttrEx::HasTextEffects****bool HasTextEffects() const**

Returns `true` if the attribute object specifies text effects.

**wxTextAttrEx::HasURL****bool HasURL() const**

Returns `true` if the attribute object specifies a URL.

**wxTextAttrEx::Init****void Init()**

Initialises this object.

**wxTextAttrEx::IsCharacterStyle****bool IsCharacterStyle() const**

Returns `true` if the object represents a character style, that is, the flags specify a font or a text background or foreground colour.

**wxTextAttrEx::IsDefault****bool IsDefault() const**

Returns `false` if we have any attributes set, `true` otherwise.

**wxTextAttrEx::IsParagraphStyle****bool IsParagraphStyle() const**

Returns `true` if the object represents a paragraph style, that is, the flags specify alignment, indentation, tabs, paragraph spacing, or bullet style.

**wxTextAttrEx::SetBulletFont****void SetBulletFont(const wxString& font)**

Sets the name of the font associated with the bullet symbol. Only valid for attributes with `wxTEXT_ATTR_BULLET_SYMBOL`.

**wxTextAttrEx::SetBulletNumber****void SetBulletNumber(int n)**

Sets the bullet number.

**wxTextAttrEx::SetBulletName****void SetBulletName(const wxString& name)**

Sets the standard bullet name, applicable if the bullet style is

`wxTEXT_ATTR_BULLET_STYLE_STANDARD`. See `wxTextAttrEx::GetBulletName` (p. 1626) for a list of supported names, and how to expand the range of supported types.

### **wxTextAttrEx::SetBulletStyle**

**void SetBulletStyle(int style)**

Sets the bullet style. The following styles can be passed:

```
#define wxTEXT_ATTR_BULLET_STYLE_NONE          0x00000000
#define wxTEXT_ATTR_BULLET_STYLE_ARABIC        0x00000001
#define wxTEXT_ATTR_BULLET_STYLE_LETTERS_UPPER 0x00000002
#define wxTEXT_ATTR_BULLET_STYLE_LETTERS_LOWER 0x00000004
#define wxTEXT_ATTR_BULLET_STYLE_ROMAN_UPPER   0x00000008
#define wxTEXT_ATTR_BULLET_STYLE_ROMAN_LOWER   0x00000010
#define wxTEXT_ATTR_BULLET_STYLE_SYMBOL        0x00000020
#define wxTEXT_ATTR_BULLET_STYLE_BITMAP        0x00000040
#define wxTEXT_ATTR_BULLET_STYLE_PARENTHESIS   0x00000080
#define wxTEXT_ATTR_BULLET_STYLE_PERIOD        0x00000100
#define wxTEXT_ATTR_BULLET_STYLE_STANDARD      0x00000200
#define wxTEXT_ATTR_BULLET_STYLE_RIGHT_PARENTHESIS 0x00000400
#define wxTEXT_ATTR_BULLET_STYLE_OUTLINE       0x00000800
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_LEFT    0x00000000
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_RIGHT   0x00001000
#define wxTEXT_ATTR_BULLET_STYLE_ALIGN_CENTRE  0x00002000
```

Currently `wxTEXT_ATTR_BULLET_STYLE_BITMAP` is not supported.

### **wxTextAttrEx::SetBulletText**

**void SetBulletText(const wxString& text)**

Sets the bullet text, which could be a symbol, or (for example) cached outline text.

### **wxTextAttrEx::SetCharacterStyleName**

**void SetCharacterStyleName(const wxString& name)**

Sets the character style name.

### **wxTextAttrEx::SetLineSpacing**

**void SetLineSpacing(int spacing)**

Sets the line spacing. *spacing* is a multiple, where 10 means single-spacing, 15 means 1.5 spacing, and 20 means double spacing. The following constants are defined for convenience:

```
#define wxTEXT_ATTR_LINE_SPACING_NORMAL 10
#define wxTEXT_ATTR_LINE_SPACING_HALF 15
#define wxTEXT_ATTR_LINE_SPACING_TWICE 20
```

**wxTextAttrEx::SetListStyleName****void SetListStyleName(const wxString& name)**

Sets the list style name.

**wxTextAttrEx::SetOutlineLevel****void SetOutlineLevel(int level)**

Specifies the outline level. Zero represents normal text. At present, the outline level is not used, but may be used in future for determining list levels and for applications that need to store document structure information.

**wxTextAttrEx::SetPageBreak****void SetPageBreak(bool pageBreak = true)**

Specifies a page break before this paragraph.

**wxTextAttrEx::SetParagraphSpacingAfter****void SetParagraphSpacingAfter(int spacing)**

Sets the spacing after a paragraph, in tenths of a millimetre.

**wxTextAttrEx::SetParagraphSpacingBefore****void SetParagraphSpacingBefore(int spacing)**

Sets the spacing before a paragraph, in tenths of a millimetre.

**wxTextAttrEx::SetParagraphStyleName****void SetParagraphStyleName(const wxString& name)**

Sets the name of the paragraph style.

**wxTextAttrEx::SetTextEffectFlags****void SetTextEffectFlags(int flags)**

Sets the text effect bits of interest. You should also pass wxTEXT\_ATTR\_EFFECTS to *wxTextAttr::SetFlags* (p. 1622).

**wxTextAttrEx::SetTextEffects**



**void SetTextEffects(int effects)**

Sets the text effects, a bit list of styles.

The following styles can be passed:

#define wxTEXT_ATTR_EFFECT_NONE	0x00000000
#define wxTEXT_ATTR_EFFECT_CAPITALS	0x00000001
#define wxTEXT_ATTR_EFFECT_SMALL_CAPITALS	0x00000002
#define wxTEXT_ATTR_EFFECT_STRIKETHROUGH	0x00000004
#define wxTEXT_ATTR_EFFECT_DOUBLE_STRIKETHROUGH	0x00000008
#define wxTEXT_ATTR_EFFECT_SHADOW	0x00000010
#define wxTEXT_ATTR_EFFECT_EMBOSS	0x00000020
#define wxTEXT_ATTR_EFFECT_OUTLINE	0x00000040
#define wxTEXT_ATTR_EFFECT_ENGRAVE	0x00000080
#define wxTEXT_ATTR_EFFECT_SUPERSCRIPT	0x00000100
#define wxTEXT_ATTR_EFFECT_SUBSCRIPT	0x00000200

Of these, only `wxTEXT_ATTR_EFFECT_CAPITALS` and `wxTEXT_ATTR_EFFECT_STRIKETHROUGH` are implemented. `wxTEXT_ATTR_EFFECT_CAPITALS` capitalises text when displayed (leaving the case of the actual buffer text unchanged), and `wxTEXT_ATTR_EFFECT_STRIKETHROUGH` draws a line through text.

To set effects, you should also pass `wxTEXT_ATTR_EFFECTS` to `wxTextAttr::SetFlags` (p. 1622), and call `wxTextAttrEx::SetTextEffectFlags` (p. 1632) with the styles (taken from the above set) that you are interested in setting.

**wxTextAttrEx::SetURL****void SetURL(const wxString& url)**

Sets the URL for the content. Sets the `wxTEXT_ATTR_URL` style; content with this style causes `wxRichTextCtrl` to show a hand cursor over it, and `wxRichTextCtrl` generates a `wxTextUrlEvent` when the content is clicked.

**wxTextAttrEx::operator=****void operator operator=(const wxTextAttr& attr)**

Assignment from a `wxTextAttr` object.

**void operator operator=(const wxTextAttrEx& attr)**

Assignment from a `wxTextAttrEx` object.

**wxTextCtrl**

A text control allows text to be displayed and edited. It may be single line or multi-line.

**Derived from**

streambuf  
*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/textctrl.h>

**Window styles**

<b>wxTE_PROCESS_ENTER</b>	The control will generate the event <code>wxEVT_COMMAND_TEXT_ENTER</code> (otherwise pressing Enter key is either processed internally by the control or used for navigation between dialog controls).
<b>wxTE_PROCESS_TAB</b>	The control will receive <code>wxEVT_CHAR</code> events for TAB pressed - normally, TAB is used for passing to the next control in a dialog instead. For the control created with this style, you can still use Ctrl-Enter to pass to the next control from the keyboard.
<b>wxTE_MULTILINE</b>	The text control allows multiple lines.
<b>wxTE_PASSWORD</b>	The text will be echoed as asterisks.
<b>wxTE_READONLY</b>	The text will not be user-editable.
<b>wxTE_RICH</b>	Use rich text control under Win32, this allows to have more than 64KB of text in the control even under Win9x. This style is ignored under other platforms.
<b>wxTE_RICH2</b>	Use rich text control version 2.0 or 3.0 under Win32, this style is ignored under other platforms
<b>wxTE_AUTO_URL</b>	Highlight the URLs and generate the <code>wxTextUrlEvents</code> when mouse events occur over them. This style is only supported for <code>wxTE_RICH</code> Win32 and multi-line <code>wxGTK2</code> text controls.
<b>wxTE_NOHIDESEL</b>	By default, the Windows text control doesn't show the selection when it doesn't have focus - use this style to force it to always show it. It doesn't do anything under other platforms.
<b>wxHSCROLL</b>	A horizontal scrollbar will be created and used, so that text won't be wrapped. No effect under <code>wxGTK1</code> .
<b>wxTE_LEFT</b>	The text in the control will be left-justified (default).
<b>wxTE_CENTRE</b>	The text in the control will be centered (currently <code>wxMSW</code> and <code>wxGTK2</code> only).

<b>wxTE_RIGHT</b>	The text in the control will be right-justified (currently wxMSW and wxGTK2 only).
<b>wxTE_DONTWRAP</b>	Same as wxHSCROLL style: don't wrap at all, show horizontal scrollbar instead.
<b>wxTE_CHARWRAP</b>	Wrap the lines too long to be shown entirely at any position (wxUniv and wxGTK2 only).
<b>wxTE_WORDWRAP</b>	Wrap the lines too long to be shown entirely at word boundaries (wxUniv and wxGTK2 only).
<b>wxTE_BESTWRAP</b>	Wrap the lines at word boundaries or at any other character if there are words longer than the window width (this is the default).
<b>wxTE_CAPITALIZE</b>	On PocketPC and Smartphone, causes the first letter to be capitalized.

See also *window styles overview* (p. 2089) and *wxTextCtrl::wxTextCtrl* (p. 1638).

Note that alignment styles (**wxTE\_LEFT**, **wxTE\_CENTRE** and **wxTE\_RIGHT**) can be changed dynamically after control creation on wxMSW and wxGTK. **wxTE\_READONLY**, **wxTE\_PASSWORD** and wrapping styles can be dynamically changed under wxGTK but not wxMSW. The other styles can be only set during control creation.

#### **wxTextCtrl text format**

The multiline text controls always store the text as a sequence of lines separated by `\n` characters, i.e. in the Unix text format even on non-Unix platforms. This allows the user code to ignore the differences between the platforms but at a price: the indices in the control such as those returned by *GetInsertionPoint* (p. 1641) or *GetSelection* (p. 1643) can **not** be used as indices into the string returned by *GetValue* (p. 1644) as they're going to be slightly off for platforms using `\r\n` as separator (as Windows does), for example.

Instead, if you need to obtain a substring between the 2 indices obtained from the control with the help of the functions mentioned above, you should use *GetRange* (p. 1643). And the indices themselves can only be passed to other methods, for example *SetInsertionPoint* (p. 1649) or *SetSelection* (p. 1650).

To summarize: never use the indices returned by (multiline) *wxTextCtrl* as indices into the string it contains, but only as arguments to be passed back to the other *wxTextCtrl* methods.

#### **wxTextCtrl styles**

Multi-line text controls support the styles, i.e. provide a possibility to set colours and font for individual characters in it (note that under Windows **wxTE\_RICH** style is required for style support). To use the styles you can either call *SetDefaultStyle* (p. 1648) before inserting the text or call *SetStyle* (p. 1650) later to change the style of the text already in the control (the first solution is much more efficient).

In either case, if the style doesn't specify some of the attributes (for example you only want

to set the text colour but without changing the font nor the text background), the values of the default style will be used for them. If there is no default style, the attributes of the text control itself are used.

So the following code correctly describes what it does: the second call to *SetDefaultStyle* (p. 1648) doesn't change the text foreground colour (which stays red) while the last one doesn't change the background colour (which stays grey):

```
text->SetDefaultStyle(wxTextAttr(*wxRED));
text->AppendText("Red text\n");
text->SetDefaultStyle(wxTextAttr(wxNullColour,
*wxLIGHT_GREY));
text->AppendText("Red on grey text\n");
text->SetDefaultStyle(wxTextAttr(*wxBLUE));
text->AppendText("Blue on grey text\n");
```

### **wxTextCtrl and C++ streams**

This class multiply-inherits from **streambuf** where compilers allow, allowing code such as the following:

```
wxTextCtrl *control = new wxTextCtrl(...);

ostream stream(control)

stream << 123.456 << " some text\n";
stream.flush();
```

If your compiler does not support derivation from **streambuf** and gives a compile error, define the symbol **NO\_TEXT\_WINDOW\_STREAM** in the `wxTextCtrl` header file.

Note that independently of this setting you can always use `wxTextCtrl` itself in a stream-like manner:

```
wxTextCtrl *control = new wxTextCtrl(...);

*control << 123.456 << " some text\n";
```

always works. However the possibility to create an `ostream` associated with `wxTextCtrl` may be useful if you need to redirect the output of a function taking an `ostream` as parameter to a text control.

Another commonly requested need is to redirect **std::cout** to the text control. This could be done in the following way:

```
#include <iostream>

wxTextCtrl *control = new wxTextCtrl(...);

std::streambuf *sbOld = std::cout.rdbuf();
std::cout.rdbuf(control);

// use cout as usual, the output appears in the text control
...
```

```
std::cout.rdbuf(sbOld);
```

But `wxWidgets` provides a convenient class to make it even simpler so instead you may just do

```
#include <iostream>

wxTextCtrl *control = new wxTextCtrl(...);

wxStreamToTextRedirector redirect(control);

// all output to cout goes into the text control until the exit from
current
// scope
```

See *wxStreamToTextRedirector* (p. 1552) for more details.

### Constants

The values below are the possible return codes of the *HitTest* (p. 1644) method:

```
// the point asked is ...
enum wxTextCtrlHitTestResult
{
    wxTE_HT_UNKNOWN = -2,    // this means HitTest() is simply not
    implemented
    wxTE_HT_BEFORE,         // either to the left or upper
    wxTE_HT_ON_TEXT,        // directly on
    wxTE_HT_BELOW,          // below [the last line]
    wxTE_HT_BEYOND          // after [the end of line]
};
// ... the character returned
```

### Event handling

The following commands are processed by default event handlers in `wxTextCtrl`: `wxID_CUT`, `wxID_COPY`, `wxID_PASTE`, `wxID_UNDO`, `wxID_REDO`. The associated UI update events are also processed automatically, when the control has the focus.

To process input from a text control, use these event handler macros to direct input to member functions that take a *wxCommandEvent* (p. 250) argument.

<b>EVT_TEXT(id, func)</b>	Respond to a <code>wxEVT_COMMAND_TEXT_UPDATED</code> event, generated when the text changes. Notice that this event will be sent when the text controls contents changes - whether this is due to user input or comes from the program itself (for example, if <code>SetValue()</code> is called); see <code>ChangeValue()</code> for a function which does not send this event.
---------------------------	--

<b>EVT_TEXT_ENTER(id, func)</b>	Respond to a
---------------------------------	--------------

wxEVT\_COMMAND\_TEXT\_ENTER event, generated when enter is pressed in a text control (which must have wxTE\_PROCESS\_ENTER style for this event to be generated).

**EVT\_TEXT\_URL(id, func)**

A mouse event occurred over an URL in the text control (wxMSW and wxGTK2 only)

**EVT\_TEXT\_MAXLEN(id, func)**

User tried to enter more text into the control than the limit set by *SetMaxLength* (p. 1649).

**wxTextCtrl::wxTextCtrl****wxTextCtrl()**

Default constructor.

**wxTextCtrl(wxWindow\* parent, wxWindowID id, const wxString& value = "", const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0, const wxValidator& validator = wxDefaultValidator, const wxString& name = wxTextCtrlNameStr)**

Constructor, creating and showing a text control.

**Parameters***parent*

Parent window. Should not be NULL.

*id*

Control identifier. A value of -1 denotes a default value.

*value*

Default text value.

*pos*

Text control position.

*size*

Text control size.

*style*

Window style. See *wxTextCtrl* (p. 1633).

*validator*

Window validator.

*name*

Window name.

### Remarks

The horizontal scrollbar (**wxHSCROLL** style flag) will only be created for multi-line text controls. Without a horizontal scrollbar, text lines that don't fit in the control's size will be wrapped (but no newline character is inserted). Single line controls don't have a horizontal scrollbar, the text is automatically scrolled so that the *insertion point* (p. 1641) is always visible.

### See also

*wxTextCtrl::Create* (p. 1640), *wxValidator* (p. 1767)

### **wxTextCtrl::~~wxTextCtrl**

**~wxTextCtrl()**

Destructor, destroying the text control.

### **wxTextCtrl::AppendText**

**void AppendText(const wxString& text)**

Appends the text to the end of the text control.

### Parameters

*text*

Text to write to the text control.

### Remarks

After the text is appended, the insertion point will be at the end of the text control. If this behaviour is not desired, the programmer should use *GetInsertionPoint* (p. 1641) and *SetInsertionPoint* (p. 1649).

### See also

*wxTextCtrl::WriteText* (p. 1652)

### **wxTextCtrl::CanCopy**

**virtual bool CanCopy()**

Returns `true` if the selection can be copied to the clipboard.

### **wxTextCtrl::CanCut**

**virtual bool CanCut()**

Returns `true` if the selection can be cut to the clipboard.

### **wxTextCtrl::CanPaste**

#### **virtual bool CanPaste()**

Returns `true` if the contents of the clipboard can be pasted into the text control. On some platforms (Motif, GTK) this is an approximation and returns `true` if the control is editable, `false` otherwise.

### **wxTextCtrl::CanRedo**

#### **virtual bool CanRedo()**

Returns `true` if there is a redo facility available and the last operation can be redone.

### **wxTextCtrl::CanUndo**

#### **virtual bool CanUndo()**

Returns `true` if there is an undo facility available and the last operation can be undone.

### **wxTextCtrl::Clear**

#### **virtual void Clear()**

Clears the text in the control.

Note that this function will generate a `wxEVT_COMMAND_TEXT_UPDATED` event.

### **wxTextCtrl::Copy**

#### **virtual void Copy()**

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

### **wxTextCtrl::Create**

**bool Create**(*wxWindow\* parent*, **wxWindowID** *id*, **const wxString&** *value* = `""`, **const wxPoint&** *pos* = `wxDefaultPosition`, **const wxSize&** *size* = `wxDefaultSize`, **long** *style* = `0`, **const wxValidator&** *validator* = `wxDefaultValidator`, **const wxString&** *name* = `wxTextCtrlNameStr`)

Creates the text control for two-step construction. Derived classes should call or replace this function. See *wxTextCtrl::wxTextCtrl* (p. 1638) for further details.

### **wxTextCtrl::Cut**

#### **virtual void Cut()**



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

### **wxTextCtrl::DiscardEdits**

**void DiscardEdits()**

Resets the internal 'modified' flag as if the current edits had been saved.

### **wxTextCtrl::EmulateKeyPress**

**bool EmulateKeyPress(const wxKeyEvent& event)**

This function inserts into the control the character which would have been inserted if the given key event had occurred in the text control. The *event* object should be the same as the one passed to `EVT_KEY_DOWN` handler previously by `wxWidgets`.

Please note that this function doesn't currently work correctly for all keys under any platform but MSW.

#### **Return value**

`true` if the event resulted in a change to the control, `false` otherwise.

### **wxTextCtrl::GetDefaultStyle**

**const wxTextAttr& GetDefaultStyle() const**

Returns the style currently used for the new text.

#### **See also**

*SetDefaultStyle* (p. 1648)

### **wxTextCtrl::GetInsertionPoint**

**virtual long GetInsertionPoint() const**

Returns the insertion point. This is defined as the zero based index of the character position to the right of the insertion point. For example, if the insertion point is at the end of the text control, it is equal to both *GetValue()* (p. 1644).*Length()* and *GetLastPosition()* (p. 1641).

The following code snippet safely returns the character at the insertion point or the zero character if the point is at the end of the control.

```
char GetCurrentChar(wxTextCtrl *tc) {
    if (tc->GetInsertionPoint() == tc->GetLastPosition())
        return '\0';
    return tc->GetValue[tc->GetInsertionPoint()];
}
```

### **wxTextCtrl::GetLastPosition**

**virtual wxTextPos GetLastPosition() const**

Returns the zero based index of the last position in the text control, which is equal to the number of characters in the control.

**wxTextCtrl::GetLineLength****int GetLineLength(long *lineNo*) const**

Gets the length of the specified line, not including any trailing newline character(s).

**Parameters**

*lineNo*

Line number (starting from zero).

**Return value**

The length of the line, or -1 if *lineNo* was invalid.

**wxTextCtrl::GetLineText****wxString GetLineText(long *lineNo*) const**

Returns the contents of a given line in the text control, not including any trailing newline character(s).

**Parameters**

*lineNo*

The line number, starting from zero.

**Return value**

The contents of the line.

**wxTextCtrl::GetNumberOfLines****int GetNumberOfLines() const**

Returns the number of lines in the text control buffer.

**Remarks**

Note that even empty text controls have one line (where the insertion point is), so `GetNumberOfLines()` never returns 0.

For wxGTK using GTK+ 1.2.x and earlier, the number of lines in a multi-line text control is calculated by actually counting newline characters in the buffer, i.e. this function returns the number of logical lines and doesn't depend on whether any of them are wrapped. For all the other platforms, the number of physical lines in the control is returned.

Also note that you may wish to avoid using functions that work with line numbers if you are working with controls that contain large amounts of text as this function has  $O(N)$  complexity for  $N$  being the number of lines.

### **wxTextCtrl::GetRange**

**virtual wxString GetRange(long *from*, long *to*) const**

Returns the string containing the text starting in the positions *from* and up to *to* in the control. The positions must have been returned by another wxTextCtrl method.

Please note that the positions in a multiline wxTextCtrl do **not** correspond to the indices in the string returned by *GetValue* (p. 1644) because of the different new line representations (CR or CR LF) and so this method should be used to obtain the correct results instead of extracting parts of the entire value. It may also be more efficient, especially if the control contains a lot of data.

### **wxTextCtrl::GetSelection**

**virtual void GetSelection(long\* *from*, long\* *to*) const**

Gets the current selection span. If the returned values are equal, there was no selection.

Please note that the indices returned may be used with the other wxTextCtrl methods but don't necessarily represent the correct indices into the string returned by *GetValue()* (p. 1644) for multiline controls under Windows (at least,) you should use *GetStringSelection()* (p. 1643) to get the selected text.

#### **Parameters**

*from*

The returned first position.

*to*

The returned last position.

**wxPython note:** The wxPython version of this method returns a tuple consisting of the from and to values.

**wxPerl note:** In wxPerl this method takes no parameter and returns a 2-element list ( *from*, *to* ).

### **wxTextCtrl::GetStringSelection**

**virtual wxString GetStringSelection()**

Gets the text currently selected in the control. If there is no selection, the returned string is empty.

### **wxTextCtrl::GetStyle**

**bool GetStyle(long position, wxTextAttr& style)**

Returns the style at this position in the text control. Not all platforms support this function.

**Return value**

`true` on success, `false` if an error occurred - it may also mean that the styles are not supported under this platform.

**See also**

*wxTextCtrl::SetStyle* (p. 1650), *wxTextAttr* (p. 1618)

**wxTextCtrl::GetValue**

**wxString GetValue() const**

Gets the contents of the control. Notice that for a multiline text control, the lines will be separated by (Unix-style) `\n` characters, even under Windows where they are separated by a `\r\n` sequence in the native control.

**wxTextCtrl::HitTest**

**wxTextCtrlHitTestResult HitTest(const wxPoint& pt, wxTextCoord \*col, wxTextCoord \*row) const**

This function finds the character at the specified position expressed in pixels. If the return code is not `wxTE_HT_UNKNOWN` the row and column of the character closest to this position are returned in the *col* and *row* parameters (unless the pointers are `NULL` which is allowed).

Please note that this function is currently only implemented in `wxUniv`, `wxMSW` and `wxGTK2` ports.

**See also**

*PositionToXY* (p. 1646), *XYToPosition* (p. 1652)

**wxPerl note:** In `wxPerl` this function takes only the position argument and returns a 3-element list (`result`, `col`, `row`).

**wxTextCtrl::IsEditable**

**bool IsEditable() const**

Returns `true` if the controls contents may be edited by user (note that it always can be changed by the program), i.e. if the control hasn't been put in read-only mode by a previous call to *SetEditable* (p. 1649).

**wxTextCtrl::IsEmpty**

**bool IsEmpty() const**

Returns `true` if the control is currently empty. This is the same as `GetValue().empty()` but can be much more efficient for the multiline controls containing big amounts of text.

This function is new since wxWidgets version 2.7.1

### **wxTextCtrl::IsModified**

#### **bool IsModified() const**

Returns `true` if the text has been modified by user. Note that calling `SetValue` (p. 1651) doesn't make the control modified.

#### **See also**

*MarkDirty* (p. 1646)

### **wxTextCtrl::IsMultiLine**

#### **bool IsMultiLine() const**

Returns `true` if this is a multi line edit control and `false` otherwise.

#### **See also**

*IsSingleLine* (p. 1645)

### **wxTextCtrl::IsSingleLine**

#### **bool IsSingleLine() const**

Returns `true` if this is a single line edit control and `false` otherwise.

#### **See also**

*IsMultiLine* (p. 1645)

### **wxTextCtrl::LoadFile**

#### **bool LoadFile(const wxString& filename, int fileType = wxTEXT\_TYPE\_ANY)**

Loads and displays the named file, if it exists.

#### **Parameters**

*filename*

The filename of the file to load.

*fileType*

The type of file to load. This is currently ignored in `wxTextCtrl`.

**Return value**

`true` if successful, `false` otherwise.

**wxTextCtrl::MarkDirty****void MarkDirty()**

Mark text as modified (dirty).

**See also**

*IsModified* (p. 1645)

**wxTextCtrl::OnDropFiles****void OnDropFiles(wxDropFilesEvent& event)**

This event handler function implements default drag and drop behaviour, which is to load the first dropped file into the control.

**Parameters**

*event*

The drop files event.

**Remarks**

This is not implemented on non-Windows platforms.

**See also**

*wxDropFilesEvent* (p. 557)

**wxTextCtrl::Paste****virtual void Paste()**

Pastes text from the clipboard to the text item.

**wxTextCtrl::PositionToXY****bool PositionToXY(long pos, long \*x, long \*y) const**

Converts given position to a zero-based column, line number pair.

**Parameters**

*pos*

Position.

*x*

Receives zero based column number.

*y*

Receives zero based line number.

### **Return value**

`true` on success, `false` on failure (most likely due to a too large position parameter).

### **See also**

`wxTextCtrl::XYToPosition` (p. 1652)

**wxPython note:** In Python, `PositionToXY()` returns a tuple containing the `x` and `y` values, so `(x,y) = PositionToXY()` is equivalent to the call described above.

**wxPerl note:** In wxPerl this method only takes the `pos` parameter, and returns a 2-element list ( `x`, `y` ).

### **wxTextCtrl::Redo**

#### **virtual void Redo()**

If there is a redo facility and the last operation can be redone, redoes the last operation. Does nothing if there is no redo facility.

### **wxTextCtrl::Remove**

#### **virtual void Remove(long from, long to)**

Removes the text starting at the first given position up to (but not including) the character at the last position.

#### **Parameters**

*from*

The first position.

*to*

The last position.

### **wxTextCtrl::Replace**

#### **virtual void Replace(long from, long to, const wxString& value)**

Replaces the text starting at the first position up to (but not including) the character at the last position with the given text.

**Parameters***from*

The first position.

*to*

The last position.

*value*

The value to replace the existing text with.

**wxTextCtrl::SaveFile****bool SaveFile(const wxString& filename, int fileType = wxTEXT\_TYPE\_ANY)**

Saves the contents of the control in a text file.

**Parameters***filename*

The name of the file in which to save the text.

*fileType*

The type of file to save. This is currently ignored in wxTextCtrl.

**Return value**

`true` if the operation was successful, `false` otherwise.

**wxTextCtrl::SetDefaultStyle****bool SetDefaultStyle(const wxTextAttr& style)**

Changes the default style to use for the new text which is going to be added to the control using *WriteText* (p. 1652) or *AppendText* (p. 1639).

If either of the font, foreground, or background colour is not set in *style*, the values of the previous default style are used for them. If the previous default style didn't set them neither, the global font or colours of the text control itself are used as fall back.

However if the *style* parameter is the default wxTextAttr, then the default style is just reset (instead of being combined with the new style which wouldn't change it at all).

**Parameters***style*

The style for the new text.



**Return value**

`true` on success, `false` if an error occurred - may also mean that the styles are not supported under this platform.

**See also**

*GetDefaultStyle* (p. 1641)

**wxTextCtrl::SetEditable**

**virtual void SetEditable(const bool *editable*)**

Makes the text item editable or read-only, overriding the `wxTE_READONLY` flag.

**Parameters**

*editable*

If `true`, the control is editable. If `false`, the control is read-only.

**See also**

*IsEditable* (p. 1644)

**wxTextCtrl::SetInsertionPoint**

**virtual void SetInsertionPoint(long *pos*)**

Sets the insertion point at the given position.

**Parameters**

*pos*

Position to set.

**wxTextCtrl::SetInsertionPointEnd**

**virtual void SetInsertionPointEnd()**

Sets the insertion point at the end of the text control. This is equivalent to *SetInsertionPoint* (p. 1649)(*GetLastPosition* (p. 1641)()).

**wxTextCtrl::SetMaxLength**

**virtual void SetMaxLength(unsigned long *len*)**

This function sets the maximum number of characters the user can enter into the control. In other words, it allows to limit the text value length to *len* not counting the terminating `NUL` character.

If *len* is 0, the previously set max length limit, if any, is discarded and the user may enter as

much text as the underlying native text control widget supports (typically at least 32Kb).

If the user tries to enter more characters into the text control when it already is filled up to the maximal length, `awxEVT_COMMAND_TEXT_MAXLEN` event is sent to notify the program about it (giving it the possibility to show an explanatory message, for example) and the extra input is discarded.

Note that under GTK+, this function may only be used with single line text controls.

### Compatibility

Only implemented in wxMSW/wxGTK starting with wxWidgets 2.3.2.

### **wxTextCtrl::SetModified**

**void SetModified**(bool *modified*)

Marks the control as being modified by the user or not.

### See also

*MarkDirty* (p. 1646), *DiscardEdits* (p. 1641)

### **wxTextCtrl::SetSelection**

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

Selects the text starting at the first position up to (but not including) the character at the last position. If both parameters are equal to -1 all text in the control is selected.

### Parameters

*from*

The first position.

*to*

The last position.

### **wxTextCtrl::SetStyle**

**bool SetStyle**(long *start*, long *end*, const wxTextAttr& *style*)

Changes the style of the given range. If any attribute within *style* is not set, the corresponding attribute from *GetDefaultStyle()* (p. 1641) is used.

### Parameters

*start*

The start of the range to change.

*end*

The end of the range to change.

*style*

The new style for the range.

### Return value

`true` on success, `false` if an error occurred - it may also mean that the styles are not supported under this platform.

### See also

`wxTextCtrl::GetStyle` (p. 1643), `wxTextAttr` (p. 1618)

## **wxTextCtrl::SetValue**

**virtual void SetValue(const wxString& value)**

Sets the text value and marks the control as not-modified (which means that `IsModified` (p. 1645) would return `false` immediately after the call to `SetValue`).

Note that this function will generate a `wxEVT_COMMAND_TEXT_UPDATED` event.

This function is deprecated and should not be used in new code. Please use the `ChangeValue` (p. 1651) function instead.

### Parameters

*value*

The new value to set. It may contain newline characters if the text control is multi-line.

## **wxTextCtrl::ChangeValue**

**virtual void ChangeValue(const wxString& value)**

Sets the text value and marks the control as not-modified (which means that `IsModified` (p. 1645) would return `false` immediately after the call to `SetValue`).

Note that this function will *not* generate the `wxEVT_COMMAND_TEXT_UPDATED` event. This is the only difference with `SetValue` (p. 1651). See *this topic* (p. 2081) for more information.

This function is new since wxWidgets version 2.7.1

### Parameters

*value*

The new value to set. It may contain newline characters if the text control is multi-line.

**wxTextCtrl::ShowPosition****void ShowPosition(long pos)**

Makes the line containing the given position visible.

**Parameters***pos*

The position that should be visible.

**wxTextCtrl::Undo****virtual void Undo()**

If there is an undo facility and the last operation can be undone, undoes the last operation. Does nothing if there is no undo facility.

**wxTextCtrl::WriteText****void WriteText(const wxString& text)**

Writes the text into the text control at the current insertion position.

**Parameters***text*

Text to write to the text control.

**Remarks**

Newlines in the text string are the only control characters allowed, and they will cause appropriate line breaks. See *wxTextCtrl::<<* (p. 1653) and *wxTextCtrl::AppendText* (p. 1639) for more convenient ways of writing to the window.

After the write operation, the insertion point will be at the end of the inserted text, so subsequent write operations will be appended. To append text after the user may have interacted with the control, call *wxTextCtrl::SetInsertionPointEnd* (p. 1649) before writing.

**wxTextCtrl::XYToPosition****long XYToPosition(long x, long y)**

Converts the given zero based column and line number to a position.

**Parameters***x*

The column number.

*y*

The line number.

### Return value

The position value, or -1 if *x* or *y* was invalid.

### **wxTextCtrl::operator <<**

**wxTextCtrl& operator <<(const wxString& s)**

**wxTextCtrl& operator <<(int i)**

**wxTextCtrl& operator <<(long l)**

**wxTextCtrl& operator <<(float f)**

**wxTextCtrl& operator <<(double d)**

**wxTextCtrl& operator <<(char c)**

Operator definitions for appending to a text control, for example:

```
wxTextCtrl *wnd = new wxTextCtrl(my_frame);

(*wnd) << "Welcome to text control number " << 1 << ".\n";
```

## **wxTextDataObject**

**wxTextDataObject** is a specialization of **wxDataObject** for text data. It can be used without change to paste data into the *wxClipboard* (p. 195) or a *wxDropSource* (p. 558). A user may wish to derive a new class from this class for providing text on-demand in order to minimize memory consumption when offering data in several formats, such as plain text and RTF because by default the text is stored in a string in this class, but it might as well be generated when requested. For this, *GetTextLength* (p. 1654) and *GetText* (p. 1654) will have to be overridden.

Note that if you already have the text inside a string, you will not achieve any efficiency gain by overriding these functions because copying **wxStrings** is already a very efficient operation (data is not actually copied because **wxStrings** are reference counted).

**wxPython note:** If you wish to create a derived **wxTextDataObject** class in **wxPython** you should derive the class from **wxPyTextDataObject** in order to get Python-aware capabilities for the various virtual methods.

### **Virtual functions to override**

This class may be used as is, but all of the data transfer functions may be overridden to increase efficiency.

### **Derived from**

*wxDataObjectSimple* (p. 335)

*wxDataObject* (p. 311)

#### **Include files**

<wx/dataobj.h>

#### **See also**

*Clipboard and drag and drop overview* (p. 2150), *wxDataObject* (p. 311),  
*wxDataObjectSimple* (p. 335), *wxFileDataObject* (p. 599), *wxBitmapDataObject* (p. 145)

### **wxTextDataObject::wxTextDataObject**

**wxTextDataObject(const wxString& text = wxEmptyString)**

Constructor, may be used to initialise the text (otherwise *SetText* (p. 1654) should be used later).

### **wxTextDataObject::GetTextLength**

**virtual size\_t GetTextLength() const**

Returns the data size. By default, returns the size of the text data set in the constructor or using *SetText* (p. 1654). This can be overridden to provide text size data on-demand. It is recommended to return the text length plus 1 for a trailing zero, but this is not strictly required.

### **wxTextDataObject::GetText**

**virtual wxString GetText() const**

Returns the text associated with the data object. You may wish to override this method when offering data on-demand, but this is not required by wxWidgets' internals. Use this method to get data in text form from the *wxClipboard* (p. 195).

### **wxTextDataObject::SetText**

**virtual void SetText(const wxString& strText)**

Sets the text associated with the data object. This method is called when the data object receives the data and, by default, copies the text into the member variable. If you want to process the text on the fly you may wish to override this function.

## **wxTextDropTarget**

A predefined drop target for dealing with text data.

#### **Derived from**

*wxDropTarget* (p. 561)

#### **Include files**

<wx/dnd.h>

#### **See also**

*Drag and drop overview* (p. 2150), *wxDropSource* (p. 558), *wxDropTarget* (p. 561), *wxFileDropTarget* (p. 604)

### **wxTextDropTarget::wxTextDropTarget**

**wxTextDropTarget()**

Constructor.

### **wxTextDropTarget::OnDrop**

**virtual bool OnDrop(long x, long y, const void \*data, size\_t size)**

See *wxDropTarget::OnDrop* (p. 562). This function is implemented appropriately for text, and calls *wxTextDropTarget::OnDropText* (p. 1655).

### **wxTextDropTarget::OnDropText**

**virtual bool OnDropText(wxCoord x, wxCoord y, const wxString& data)**

Override this function to receive dropped text.

#### **Parameters**

*x*

The x coordinate of the mouse.

*y*

The y coordinate of the mouse.

*data*

The data being dropped: a wxString.

#### **Return value**

Return true to accept the data, false to veto the operation.

## **wxTextEntryDialog**

This class represents a dialog that requests a one-line text string from the user. It is implemented as a generic wxWidgets dialog.

**Derived from**

*wxDialog* (p. 496)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/textdlg.h>

**See also**

*wxTextEntryDialog* overview (p. 2131)

**wxTextEntryDialog::wxTextEntryDialog**

**wxTextEntryDialog**(*wxWindow\** parent, **const wxString&** message, **const wxString&** caption = "Please enter text", **const wxString&** defaultValue = "", **long** style = wxOK | wxCANCEL | wxCENTRE, **const wxPoint&** pos = wxDefaultPosition)

Constructor. Use *wxTextEntryDialog::ShowModal* (p. 1657) to show the dialog.

**Parameters**

*parent*

Parent window.

*message*

Message to show on the dialog.

*defaultValue*

The default value, which may be the empty string.

*style*

A dialog style, specifying the buttons (wxOK, wxCANCEL) and an optional wxCENTRE style. Additionally, wxTextCtrl styles (such as **wxTE\_PASSWORD**) may be specified here.

*pos*

Dialog position.

**wxTextEntryDialog::~wxTextEntryDialog**



**~wxTextEntryDialog()**

Destructor.

**wxTextEntryDialog::GetValue****wxString GetValue() const**

Returns the text that the user has entered if the user has pressed OK, or the original value if the user has pressed Cancel.

**wxTextEntryDialog::SetValue****void SetValue(const wxString& value)**

Sets the default text value.

**wxTextEntryDialog::ShowModal****int ShowModal()**

Shows the dialog, returning wxID\_OK if the user pressed OK, and wxID\_CANCEL otherwise.

**wxTextFile**

The wxTextFile is a simple class which allows to work with text files on line by line basis. It also understands the differences in line termination characters under different platforms and will not do anything bad to files with "non native" line termination sequences - in fact, it can be also used to modify the text files and change the line termination characters from one type (say DOS) to another (say Unix).

One word of warning: the class is not at all optimized for big files and thus it will load the file entirely into memory when opened. Of course, you should not work in this way with large files (as an estimation, anything over 1 Megabyte is surely too big for this class). On the other hand, it is not a serious limitation for small files like configuration files or program sources which are well handled by wxTextFile.

The typical things you may do with wxTextFile in order are:

- Create and open it: this is done with either *Create* (p. 1659) or *Open* (p. 1662) function which opens the file (name may be specified either as the argument to these functions or in the constructor), reads its contents in memory (in the case of *Open* ( )) and closes it.
- Work with the lines in the file: this may be done either with "direct access" functions like *GetLineCount* (p. 1659) and *GetLine* (p. 1659) (*operator[]* does exactly the same but looks more like array addressing) or with "sequential access" functions which include *GetFirstLine* (p. 1660)/*GetNextLine* (p. 1661) and also *GetLastLine* (p. 1661)/*GetPrevLine* (p. 1661). For the sequential access

functions the current line number is maintained: it is returned by *GetCurrentLine* (p. 1660) and may be changed with *GoToLine* (p. 1660).

- Add/remove lines to the file: *AddLine* (p. 1659) and *InsertLine* (p. 1662) add new lines while *RemoveLine* (p. 1662) deletes the existing ones. *Clear* (p. 1662) resets the file to empty.
- Save your changes: notice that the changes you make to the file will **not** be saved automatically; calling *Close* (p. 1659) or doing nothing discards them! To save the changes you must explicitly call *Write* (p. 1662) - here, you may also change the line termination type if you wish.

### Derived from

No base class

### Include files

<wx/textfile.h>

### Data structures

The following constants identify the line termination type:

```
enum wxTextFileType
{
    wxTextFileType_None,    // incomplete (the last line of the file
                           // only)
    wxTextFileType_Unix,    // line is terminated with 'LF' = 0xA = 10
    = '\n'
    wxTextFileType_Dos,     // 'CR' 'LF'
    wxTextFileType_Mac      // 'CR' = 0xD = 13
    = '\r'
};
```

### See also

*wxFile* (p. 591)

### wxTextFile::wxTextFile

#### wxTextFile() const

Default constructor, use *Create* (p. 1659) or *Open* (p. 1662) with a file name parameter to initialize the object.

### wxTextFile::wxTextFile

#### wxTextFile(const wxString& strFile) const

Constructor does not load the file into memory, use *Open()* to do it.

**wxTextFile::~~wxTextFile****~wxTextFile() const**

Destructor does nothing.

**wxTextFile::AddLine****void AddLine(const wxString& str, wxTextFileType type = typeDefault) const**

Adds a line to the end of file.

**wxTextFile::Close****bool Close() const**

Closes the file and frees memory, **losing all changes**. Use *Write()* (p. 1662) if you want to save them.

**wxTextFile::Create****bool Create() const****bool Create(const wxString& strFile) const**

Creates the file with the given name or the name which was given in the *constructor* (p. 1658). The array of file lines is initially empty.

It will fail if the file already exists, *Open* (p. 1662) should be used in this case.

**wxTextFile::Exists****bool Exists() const**

Return true if file exists - the name of the file should have been specified in the constructor before calling *Exists()*.

**wxTextFile::IsOpened****bool IsOpened() const**

Returns true if the file is currently opened.

**wxTextFile::GetLineCount****size\_t GetLineCount() const**

Get the number of lines in the file.

**wxTextFile::GetLine**

**wxString& GetLine(size\_t n) const**

Retrieves the line number *n* from the file. The returned line may be modified but you shouldn't add line terminator at the end - this will be done by `wxTextFile`.

**wxTextFile::operator[]****wxString& operator[](size\_t n) const**

The same as `GetLine` (p. 1659).

**wxTextFile::GetCurrentLine****size\_t GetCurrentLine() const**

Returns the current line: it has meaning only when you're using `GetFirstLine()/GetNextLine()` functions, it doesn't get updated when you're using "direct access" functions like `GetLine()`. `GetFirstLine()` and `GetLastLine()` also change the value of the current line, as well as `GoToLine()`.

**wxTextFile::GoToLine****void GoToLine(size\_t n) const**

Changes the value returned by `GetCurrentLine` (p. 1660) and used by `GetFirstLine()` (p. 1660)/`GetNextLine()` (p. 1661).

**wxTextFile::Eof****bool Eof() const**

Returns true if the current line is the last one.

**wxTextFile::GetEOL****static const char\* GetEOL(wxTextFileType type = typeDefault) const**

Get the line termination string corresponding to given constant. *typeDefault* is the value defined during the compilation and corresponds to the native format of the platform, i.e. it will be `wxTextFileType_Dos` under Windows, `wxTextFileType_Unix` under Unix (including Mac OS X when compiling with the Apple Developer Tools) and `wxTextFileType_Mac` under Mac OS (including Mac OS X when compiling with CodeWarrior).

**wxTextFile::GetFirstLine****wxString& GetFirstLine() const**

This method together with `GetNextLine()` (p. 1661) allows more "iterator-like" traversal of the list of lines, i.e. you may write something like:

```
wxTextFile file;
```

```
...
for ( str = file.GetFirstLine(); !file.Eof(); str =
file.GetNextLine() )
{
    // do something with the current line in str
}
// do something with the last line in str
```

**wxTextFile::GetNextLine****wxString& GetNextLine()**

Gets the next line (see *GetFirstLine* (p. 1660) for the example).

**wxTextFile::GetPrevLine****wxString& GetPrevLine()**

Gets the previous line in the file.

**wxTextFile::GetLastLine****wxString& GetLastLine()**

Gets the last line of the file. Together with *GetPrevLine* (p. 1661) it allows to enumerate the lines in the file from the end to the beginning like this:

```
wxTextFile file;
...
for ( str = file.GetLastLine();
      file.GetCurrentLine() > 0;
      str = file.GetPrevLine() )
{
    // do something with the current line in str
}
// do something with the first line in str
```

**wxTextFile::GetLineType****wxTextFileType GetLineType(size\_t n) const**

Get the type of the line (see also *GetEOL* (p. 1660))

**wxTextFile::GuessType****wxTextFileType GuessType() const**

Guess the type of file (which is supposed to be opened). If sufficiently many lines of the file are in DOS/Unix/Mac format, the corresponding value will be returned. If the detection mechanism fails `wxTextFileType_None` is returned.

**wxTextFile::GetName****const char\* GetName() const**

Get the name of the file.

**wxTextFile::InsertLine****void InsertLine(const wxString& str, size\_t n, wxTextFileType type = typeDefault) const**

Insert a line before the line number *n*.

**wxTextFile::Open****bool Open(wxMBConv& conv = wxConvUTF8) const****bool Open(const wxString& strFile, wxMBConv& conv = wxConvUTF8) const**

Open() opens the file with the given name or the name which was given in the *constructor* (p. 1658) and also loads file in memory on success. It will fail if the file does not exist, *Create* (p. 1659) should be used in this case.

The *conv* argument is only meaningful in Unicode build of wxWidgets when it is used to convert the file to wide character representation.

**wxTextFile::RemoveLine****void RemoveLine(size\_t n) const**

Delete line number *n* from the file.

**wxTextFile::Clear****void Clear() const**

Delete all lines from the file, set current line number to 0.

**wxTextFile::Write****bool Write(wxTextFileType typeNew = wxTextFileType\_None, wxMBConv& conv = wxConvUTF8) const**

Change the file on disk. The *typeNew* parameter allows you to change the file format (default argument means "don't change type") and may be used to convert, for example, DOS files to Unix.

The *conv* argument is only meaningful in Unicode build of wxWidgets when it is used to convert all lines to multibyte representation before writing them to physical file.

Returns true if operation succeeded, false if it failed.

## wxTextInputStream

This class provides functions that read text datas using an input stream. So, you can read *text* floats, integers.

The wxTextInputStream correctly reads text files (or streams) in DOS, Macintosh and Unix formats and reports a single newline char as a line ending.

Operator >> is overloaded and you can use this class like a standard C++ iostream. Note, however, that the arguments are the fixed size types wxUInt32, wxInt32 etc and on a typical 32-bit computer, none of these match to the "long" type (wxInt32 is defined as int on 32-bit architectures) so that you cannot use long. To avoid problems (here and elsewhere), make use of wxInt32, wxUInt32 and similar types.

If you're scanning through a file using wxTextInputStream, you should check for EOF **before** reading the next item (word / number), because otherwise the last item may get lost. You should however be prepared to receive an empty item (empty string / zero number) at the end of file, especially on Windows systems. This is unavoidable because most (but not all) files end with whitespace (i.e. usually a newline).

For example:

```
wxFileInputStream input( "mytext.txt" );
wxTextInputStream text( input );
wxUInt8 i1;
float f2;
wxString line;

text >> i1;           // read a 8 bit integer.
text >> i1 >> f2;      // read a 8 bit integer followed by float.
text >> line;         // read a text line
```

### Include files

<wx/txtstrm.h>

### wxTextInputStream::wxTextInputStream

**wxTextInputStream(wxInputStream& stream, const wxString& sep=" \t", wxMBCConv& conv = wxConvUTF8 )**

Constructs a text stream associated to the given input stream.

#### Parameters

*stream*

The underlying input stream.

*sep*

The initial string separator characters.

*conv*

*In Unicode build only:* The encoding converter used to convert the bytes in the underlying input stream to characters.

### **wxTextInputStream::~~wxTextInputStream**

**~wxTextInputStream()**

Destroys the wxTextInputStream object.

### **wxTextInputStream::Read8**

**wxUInt8 Read8(int base = 10)**

Reads a single unsigned byte from the stream, given in base *base*.

The value of *base* must be comprised between 2 and 36, inclusive, or be a special value 0 which means that the usual rules of C numbers are applied: if the number starts with 0x it is considered to be in base16, if it starts with 0 - in base 8 and in base 10 otherwise. Note that you may not want to specify the base 0 if you are parsing the numbers which may have leading zeroes as they can yield unexpected (to the user not familiar with C) results.

### **wxTextInputStream::Read8S**

**wxInt8 Read8S(int base = 10)**

Reads a single signed byte from the stream.

See *wxTextInputStream::Read8* (p. 1664) for the description of the *base* parameter.

### **wxTextInputStream::Read16**

**wxUInt16 Read16(int base = 10)**

Reads a unsigned 16 bit integer from the stream.

See *wxTextInputStream::Read8* (p. 1664) for the description of the *base* parameter.

### **wxTextInputStream::Read16S**

**wxInt16 Read16S(int base = 10)**

Reads a signed 16 bit integer from the stream.

See *wxTextInputStream::Read8* (p. 1664) for the description of the *base* parameter.

### **wxTextInputStream::Read32**

**wxUInt32 Read32(int base = 10)**



Reads a 32 bit unsigned integer from the stream.

See *wxTextInputStream::Read8* (p. 1664) for the description of the *base* parameter.

### **wxTextInputStream::Read32S**

**wxInt32 Read32S(int base = 10)**

Reads a 32 bit signed integer from the stream.

See *wxTextInputStream::Read8* (p. 1664) for the description of the *base* parameter.

### **wxTextInputStream::GetChar**

**wxChar GetChar()**

Reads a character, returns 0 if there are no more characters in the stream.

### **wxTextInputStream::ReadDouble**

**double ReadDouble()**

Reads a double (IEEE encoded) from the stream.

### **wxTextInputStream::ReadLine**

**wxString ReadLine()**

Reads a line from the input stream and returns it (without the end of line character).

### **wxTextInputStream::ReadString**

**wxString ReadString()**

**NB:** This method is deprecated, use *ReadLine* (p. 1665) or *ReadWord* (p. 1665) instead.

Same as *ReadLine* (p. 1665).

### **wxTextInputStream::ReadWord**

**wxString ReadWord()**

Reads a word (a sequence of characters until the next separator) from the input stream.

### **See also**

*SetStringSeparators* (p. 1665)

### **wxTextInputStream::SetStringSeparators**

**void SetStringSeparators(const wxString& sep)**

Sets the characters which are used to define the word boundaries in *ReadWord* (p. 1665).

The default separators are the space and `TAB` characters.

## wxTextOutputStream

This class provides functions that write text datas using an output stream. So, you can write *text* floats, integers.

You can also simulate the C++ `cout` class:

```
wxFileOutputStream output( stderr );
wxTextOutputStream cout( output );

cout << "This is a text line" << endl;
cout << 1234;
cout << 1.23456;
```

The `wxTextOutputStream` writes text files (or streams) on DOS, Macintosh and Unix in their native formats (concerning the line ending).

### Include files

`<wx/txtstrm.h>`

### wxTextOutputStream::wxTextOutputStream

**wxTextOutputStream(wxOutputStream& stream, wxEOL mode = wxEOL\_NATIVE, wxMBConv& conv = wxConvUTF8)**

Constructs a text stream object associated to the given output stream.

### Parameters

*stream*

The output stream.

*mode*

The end-of-line mode. One of **wxEOL\_NATIVE**, **wxEOL\_DOS**, **wxEOL\_MAC** and **wxEOL\_UNIX**.

*conv*

*In Unicode build only:* The object used to convert Unicode text into ASCII characters written to the output stream.

### wxTextOutputStream::~~wxTextOutputStream

**~wxTextOutputStream()**

Destroys the `wxTextOutputStream` object.

### **wxTextOutputStream::GetMode**

**wxEOL GetMode()**

Returns the end-of-line mode. One of **wxEOL\_DOS**, **wxEOL\_MAC** and **wxEOL\_UNIX**.

### **wxTextOutputStream::PutChar**

**void PutChar(wxChar c)**

Writes a character to the stream.

### **wxTextOutputStream::SetMode**

**void SetMode(wxEOL mode = wxEOL\_NATIVE)**

Set the end-of-line mode. One of **wxEOL\_NATIVE**, **wxEOL\_DOS**, **wxEOL\_MAC** and **wxEOL\_UNIX**.

### **wxTextOutputStream::Write8**

**void Write8(wxUInt8 i8)**

Writes the single byte *i8* to the stream.

### **wxTextOutputStream::Write16**

**void Write16(wxUInt16 i16)**

Writes the 16 bit integer *i16* to the stream.

### **wxTextOutputStream::Write32**

**void Write32(wxUInt32 i32)**

Writes the 32 bit integer *i32* to the stream.

### **wxTextOutputStream::WriteDouble**

**virtual void WriteDouble(double f)**

Writes the double *f* to the stream using the IEEE format.

### **wxTextOutputStream::WriteString**

**virtual void WriteString(const wxString& string)**

Writes *string* as a line. Depending on the end-of-line mode the end of line ("\n") characters

in the string are converted to the correct line ending terminator.

## wxTextValidator

wxTextValidator validates text controls, providing a variety of filtering behaviours.

For more information, please see *Validator overview* (p. 2092).

### Derived from

*wxValidator* (p. 1767)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/valtext.h>

### See also

*Validator overview* (p. 2092), *wxValidator* (p. 1767), *wxGenericValidator* (p. 714)

## wxTextValidator::wxTextValidator

**wxTextValidator(const wxTextValidator& validator)**

Copy constructor.

**wxTextValidator(long style = wxFILTER\_NONE, wxString\* valPtr = NULL)**

Constructor, taking a style and optional pointer to a wxString variable.

### Parameters

*style*

A bitlist of flags, which can be:

<b>wxFILTER_NONE</b>	No filtering takes place.
<b>wxFILTER_ASCII</b>	Non-ASCII characters are filtered out.
<b>wxFILTER_ALPHA</b>	Non-alpha characters are filtered out.
<b>wxFILTER_ALPHANUMERIC</b>	Non-alphanumeric characters are filtered out.
<b>wxFILTER_NUMERIC</b>	Non-numeric characters are filtered out.
<b>wxFILTER_INCLUDE_LIST</b>	Use an include list. The validator checks if the user input is on the list, complaining if not. See <i>wxTextValidator::SetIncludes</i> (p. 1670).

**wxFILTER\_EXCLUDE\_LIST** Use an exclude list. The validator checks if the user input is on the list, complaining if it is. See *wxTextValidator::SetExcludes* (p. 1670).

**wxFILTER\_INCLUDE\_CHAR\_LIST** Use an include list. The validator checks if each input character is in the list (one character per list element), complaining if not. See *wxTextValidator::SetIncludes* (p. 1670).

**wxFILTER\_EXCLUDE\_CHAR\_LIST** Use an include list. The validator checks if each input character is in the list (one character per list element), complaining if it is. See *wxTextValidator::SetExcludes* (p. 1670).

*valPtr*

A pointer to a *wxString* variable that contains the value. This variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).

### **wxTextValidator::Clone**

**virtual wxValidator\* Clone() const**

Clones the text validator using the copy constructor.

### **wxTextValidator::GetExcludes**

**wxArrayString& GetExcludes() const**

Returns a reference to the exclude list (the list of invalid values).

### **wxTextValidator::GetIncludes**

**wxArrayString& GetIncludes() const**

Returns a reference to the include list (the list of valid values).

### **wxTextValidator::GetStyle**

**long GetStyle() const**

Returns the validator style.

### **wxTextValidator::OnChar**

**void OnChar(wxKeyEvent& event)**

Receives character input from the window and filters it according to the current validator style.

**wxTextValidator::SetExcludes****void SetExcludes(const wxArrayString& *stringList*)**

Sets the exclude list (invalid values for the user input).

**wxTextValidator::SetIncludes****void SetIncludes(const wxArrayString& *stringList*)**

Sets the include list (valid values for the user input).

**wxTextValidator::SetStyle****void SetStyle(long *style*)**

Sets the validator style.

**wxTextValidator::TransferFromWindow****virtual bool TransferFromWindow()**

Transfers the value in the text control to the string.

**wxTextValidator::TransferToWindow****virtual bool TransferToWindow()**

Transfers the string value to the text control.

**wxTextValidator::Validate****virtual bool Validate(wxWindow\* *parent*)**

Validates the window contents against the include or exclude lists, depending on the validator style.

**wxThread**

A thread is basically a path of execution through a program. Threads are sometimes called *light-weight processes*, but the fundamental difference between threads and processes is that memory spaces of different processes are separated while all threads share the same address space.

While it makes it much easier to share common data between several threads, it also makes it much easier to shoot oneself in the foot, so careful use of synchronization objects such as *mutexes* (p. 1131) or *critical sections* (p. 294) is recommended. In addition, don't create global thread objects because they allocate memory in their constructor, which will cause problems for the memory checking system.

**Derived from**

None.

**Include files**

<wx/thread.h>

**See also**

*wxMutex* (p. 1131), *wxCondition* (p. 259), *wxCriticalSection* (p. 294)

**Types of wxThreads**

There are two types of threads in wxWidgets: *detached* and *joinable*, modeled after the the POSIX thread API. This is different from the Win32 API where all threads are joinable.

By default wxThreads in wxWidgets use the detached behavior. Detached threads delete themselves once they have completed, either by themselves when they complete processing or through a call to *wxThread::Delete* (p. 1674), and thus must be created on the heap (through the new operator, for example). Conversely, joinable threads do not delete themselves when they are done processing and as such are safe to create on the stack. Joinable threads also provide the ability for one to get value it returned from *wxThread::Entry* (p. 1674) through *wxThread::Wait* (p. 1678).

You shouldn't hurry to create all the threads joinable, however, because this has a disadvantage as well: you **must** *Wait()* for a joinable thread or the system resources used by it will never be freed, and you also must delete the corresponding wxThread object yourself if you did not create it on the stack. In contrast, detached threads are of the "fire-and-forget" kind: you only have to start a detached thread and it will terminate and destroy itself.

**wxThread deletion**

Regardless of whether it has terminated or not, you should call *wxThread::Wait* (p. 1678) on a joinable thread to release its memory, as outlined in *Types of wxThreads* (p. 1671). If you created a joinable thread on the heap, remember to delete it manually with the delete operator or similar means as only detached threads handle this type of memory management.

Since detached threads delete themselves when they are finished processing, you should take care when calling a routine on one. If you are certain the thread is still running and would like to end it, you may call *wxThread::Delete* (p. 1674) to gracefully end it (which implies that the thread will be deleted after that call to *Delete()*). It should be implied that you should never attempt to delete a detached thread with the delete operator or similar means.

As mentioned, *wxThread::Wait* (p. 1678) or *wxThread::Delete* (p. 1674) attempts to gracefully terminate a joinable and detached thread, respectively. It does this by waiting until the thread in question calls *wxThread::TestDestroy* (p. 1677) or ends processing

(returns from *wxThread::Entry* (p. 1674)).

Obviously, if the thread does call *TestDestroy()* and does not end the calling thread will come to halt. This is why it is important to call *TestDestroy()* in the *Entry()* routine of your threads as often as possible.

As a last resort you can end the thread immediately through *wxThread::Kill* (p. 1676). It is strongly recommended that you do not do this, however, as it does not free the resources associated with the object (although the *wxThread* object of detached threads will still be deleted) and could leave the C runtime library in an undefined state.

### **wxWidgets calls in secondary threads**

All threads other than the "main application thread" (the one *wxApp::OnInit* (p. 52) or your main function runs in, for example) are considered "secondary threads". These include all threads created by *wxThread::Create* (p. 1673) or the corresponding constructors.

GUI calls, such as those to a *wxWindow* (p. 1795) or *wxBitmap* (p. 123) are explicitly not safe at all in secondary threads and could end your application prematurely. This is due to several reasons, including the underlying native API and the fact that *wxThread* does not run a GUI event loop similar to other APIs as MFC.

A workaround that works on some *wxWidgets* ports is calling *wxMutexGUIEnter* (p. 1919) before any GUI calls and then calling *wxMutexGUILeave* (p. 1919) afterwards. However, the recommended way is to simply process the GUI calls in the main thread through an event that is posted by either *wxPostEvent* (p. 1960) or *wxEvtHandler::AddPendingEvent* (p. 576). This does not imply that calls to these classes are thread-safe, however, as most *wxWidgets* classes are not thread-safe, including *wxString*.

### **Don't poll a wxThread**

A common problem users experience with *wxThread* is that in their main thread they will check the thread every now and then to see if it has ended through *wxThread::IsRunning* (p. 1676), only to find that their application has run into problems because the thread is using the default behavior and has already deleted itself. Naturally, they instead attempt to use joinable threads in place of the previous behavior.

However, polling a *wxThread* for when it has ended is in general a bad idea - in fact calling a routine on any running *wxThread* should be avoided if possible. Instead, find a way to notify yourself when the thread has ended. Usually you only need to notify the main thread, in which case you can post an event to it via *wxPostEvent* (p. 1960) or *wxEvtHandler::AddPendingEvent* (p. 576). In the case of secondary threads you can call a routine of another class when the thread is about to complete processing and/or set the value of a variable, possibly using *mutexes* (p. 1131) and/or other synchronization means if necessary.

### **wxThread::wxThread**

**wxThread(wxThreadKind kind = wxTHREAD\_DETACHED)**

This constructor creates a new detached (default) or joinable C++ thread object. It does



not create or start execution of the real thread -- for this you should use the *Create* (p. 1673) and *Run* (p. 1676) methods.

The possible values for *kind* parameters are:

**wxTHREAD\_DETACHED**                      Creates a detached thread.

**wxTHREAD\_JOINABLE**                      Creates a joinable thread.

### **wxThread::~~wxThread**

#### **~wxThread()**

The destructor frees the resources associated with the thread. Notice that you should never delete a detached thread -- you may only call *Delete* (p. 1674) on it or wait until it terminates (and auto destructs) itself. Because the detached threads delete themselves, they can only be allocated on the heap.

Joinable threads should be deleted explicitly. The *Delete* (p. 1674) and *Kill* (p. 1676) functions will not delete the C++ thread object. It is also safe to allocate them on stack.

### **wxThread::Create**

#### **wxThreadError Create(unsigned int stackSize = 0)**

Creates a new thread. The thread object is created in the suspended state, and you should call *Run* (p. 1676) to start running it. You may optionally specify the stack size to be allocated to it (Ignored on platforms that don't support setting it explicitly, eg. Unix system without `pthread_attr_setstacksize`). If you do not specify the stack size, the system's default value is used.

**Warning:** It is a good idea to explicitly specify a value as systems' default values vary from just a couple of KB on some systems (BSD and OS/2 systems) to one or several MB (Windows, Solaris, Linux). So, if you have a thread that requires more than just a few KB of memory, you will have mysterious problems on some platforms but not on the common ones. On the other hand, just indicating a large stack size by default will give you performance issues on those systems with small default stack since those typically use fully committed memory for the stack. On the contrary, if use a lot of threads (say several hundred), virtual address space can get tight unless you explicitly specify a smaller amount of thread stack space for each thread.

#### **Return value**

One of:

**wxTHREAD\_NO\_ERROR**                      There was no error.

**wxTHREAD\_NO\_RESOURCE**                      There were insufficient resources to create a new thread.

**wxTHREAD\_RUNNING**                      The thread is already running.

**wxThread::Delete****wxThreadError Delete()**

Calling *Delete* (p. 1674) gracefully terminates a detached thread, either when the thread calls *TestDestroy* (p. 1677) or finished processing.

(Note that while this could work on a joinable thread you simply should not call this routine on one as afterwards you may not be able to call *wxThread::Wait* (p. 1678) to free the memory of that thread).

See *wxThread deletion* (p. 1671) for a broader explanation of this routine.

**wxThread::Entry****virtual ExitCode Entry()**

This is the entry point of the thread. This function is pure virtual and must be implemented by any derived class. The thread execution will start here.

The returned value is the thread exit code which is only useful for joinable threads and is the value returned by *Wait* (p. 1678).

This function is called by wxWidgets itself and should never be called directly.

**wxThread::Exit****void Exit(ExitCode exitcode = 0)**

This is a protected function of the *wxThread* class and thus can only be called from a derived class. It also can only be called in the context of this thread, i.e. a thread can only exit from itself, not from another thread.

This function will terminate the OS thread (i.e. stop the associated path of execution) and also delete the associated C++ object for detached threads. *wxThread::OnExit* (p. 1676) will be called just before exiting.

**wxThread::GetCPUCount****static int GetCPUCount()**

Returns the number of system CPUs or -1 if the value is unknown.

**See also**

*SetConcurrency* (p. 1677)

**wxThread::GetCurrentId****static unsigned long GetCurrentId()**

Returns the platform specific thread ID of the current thread as a long. This can be used to

uniquely identify threads, even if they are not wxThreads.

### **wxThread::GetId**

**unsigned long GetId() const**

Gets the thread identifier: this is a platform dependent number that uniquely identifies the thread throughout the system during its existence (i.e. the thread identifiers may be reused).

### **wxThread::GetPriority**

**int GetPriority() const**

Gets the priority of the thread, between zero and 100.

The following priorities are defined:

<b>WXTHREAD_MIN_PRIORITY</b>	0
<b>WXTHREAD_DEFAULT_PRIORITY</b>	50
<b>WXTHREAD_MAX_PRIORITY</b>	100

### **wxThread::IsAlive**

**bool IsAlive() const**

Returns `true` if the thread is alive (i.e. started and not terminating).

Note that this function can only safely be used with joinable threads, not detached ones as the latter delete themselves and so when the real thread is no longer alive, it is not possible to call this function because the wxThread object no longer exists.

### **wxThread::IsDetached**

**bool IsDetached() const**

Returns `true` if the thread is of the detached kind, `false` if it is a joinable one.

### **wxThread::IsMain**

**static bool IsMain()**

Returns `true` if the calling thread is the main application thread.

### **wxThread::IsPaused**

**bool IsPaused() const**

Returns `true` if the thread is paused.

**wxThread::IsRunning****bool IsRunning() const**

Returns `true` if the thread is running.

This method may only be safely used for joinable threads, see the remark in *IsAlive* (p. 1675).

**wxThread::Kill****wxThreadError Kill()**

Immediately terminates the target thread. **This function is dangerous and should be used with extreme care (and not used at all whenever possible)!** The resources allocated to the thread will not be freed and the state of the C runtime library may become inconsistent. Use *Delete()* (p. 1674) for detached threads or *Wait()* (p. 1678) for joinable threads instead.

For detached threads *Kill()* will also delete the associated C++ object. However this will not happen for joinable threads and this means that you will still have to delete the *wxThread* object yourself to avoid memory leaks. In neither case *OnExit* (p. 1676) of the dying thread will be called, so no thread-specific cleanup will be performed.

This function can only be called from another thread context, i.e. a thread cannot kill itself.

It is also an error to call this function for a thread which is not running or paused (in the latter case, the thread will be resumed first) -- if you do it, a `wxTHREAD_NOT_RUNNING` error will be returned.

**wxThread::OnExit****void OnExit()**

Called when the thread exits. This function is called in the context of the thread associated with the *wxThread* object, not in the context of the main thread. This function will not be called if the thread was *killed* (p. 1676).

This function should never be called directly.

**wxThread::Pause****wxThreadError Pause()**

Suspends the thread. Under some implementations (Win32), the thread is suspended immediately, under others it will only be suspended when it calls *TestDestroy* (p. 1677) for the next time (hence, if the thread doesn't call it at all, it won't be suspended).

This function can only be called from another thread context.

**wxThread::Run**

**wxThreadError Run()**

Starts the thread execution. Should be called after *Create* (p. 1673).

This function can only be called from another thread context.

**wxThread::SetPriority**

**void SetPriority(int *priority*)**

Sets the priority of the thread, between 0 and 100. It can only be set after calling *Create()* (p. 1673) but before calling *Run()* (p. 1676).

The following priorities are already defined:

<b>WXTHREAD_MIN_PRIORITY</b>	0
<b>WXTHREAD_DEFAULT_PRIORITY</b>	50
<b>WXTHREAD_MAX_PRIORITY</b>	100

**wxThread::Sleep**

**static void Sleep(unsigned long *milliseconds*)**

Pauses the thread execution for the given amount of time.

This function should be used instead of *wxSleep* (p. 1979) by all worker threads (i.e. all except the main one).

**wxThread::Resume**

**wxThreadError Resume()**

Resumes a thread suspended by the call to *Pause* (p. 1676).

This function can only be called from another thread context.

**wxThread::SetConcurrency**

**static bool SetConcurrency(size\_t *level*)**

Sets the thread concurrency level for this process. This is, roughly, the number of threads that the system tries to schedule to run in parallel. The value of 0 for *level* may be used to set the default one.

Returns `true` on success or `false` otherwise (for example, if this function is not implemented for this platform -- currently everything except Solaris).

**wxThread::TestDestroy**

**virtual bool TestDestroy()**

This function should be called periodically by the thread to ensure that calls to *Pause* (p. 1676) and *Delete* (p. 1674) will work. If it returns `true`, the thread should exit as soon as possible.

Notice that under some platforms (POSIX), implementation of *Pause* (p. 1676) also relies on this function being called, so not calling it would prevent both stopping and suspending thread from working.

### **wxThread::This**

#### **static wxThread \* This()**

Return the thread object for the calling thread. NULL is returned if the calling thread is the main (GUI) thread, but *IsMain* (p. 1675) should be used to test whether the thread is really the main one because NULL may also be returned for the thread not created with `wxThread` class. Generally speaking, the return value for such a thread is undefined.

### **wxThread::Yield**

#### **void Yield()**

Give the rest of the thread time slice to the system allowing the other threads to run. See also *Sleep()* (p. 1677).

### **wxThread::Wait**

#### **ExitCode Wait() const**

Waits for a joinable thread to terminate and returns the value the thread returned from *wxThread::Entry* (p. 1674) or `(ExitCode)-1` on error. Notice that, unlike *Delete* (p. 1674) doesn't cancel the thread in any way so the caller waits for as long as it takes to the thread to exit.

You can only *Wait()* for joinable (not detached) threads.

This function can only be called from another thread context.

See *wxThread deletion* (p. 1671) for a broader explanation of this routine.

## **wxThreadHelper**

The `wxThreadHelper` class is a mix-in class that manages a single background thread. By deriving from `wxThreadHelper`, a class can implement the thread code in its own *wxThreadHelper::Entry* (p. 1680) method and easily share data and synchronization objects between the main thread and the worker thread. Doing this prevents the awkward passing of pointers that is needed when the original object in the main thread needs to synchronize with its worker thread in its own `wxThread` derived object.

For example, *wxFrame* (p. 682) may need to make some calculations in a background thread and then display the results of those calculations in the main window.

Ordinarily, a *wxThread* (p. 1670) derived object would be created with the calculation code implemented in *wxThread::Entry* (p. 1674). To access the inputs to the calculation, the frame object would often to pass a pointer to itself to the thread object. Similarly, the frame object would hold a pointer to the thread object. Shared data and synchronization objects could be stored in either object though the object without the data would have to access the data through a pointer.

However, with *wxThreadHelper*, the frame object and the thread object are treated as the same object. Shared data and synchronization variables are stored in the single object, eliminating a layer of indirection and the associated pointers.

**Derived from**

None.

**Include files**

<wx/thread.h>

**See also**

*wxThread* (p. 1670)

**wxThreadHelper::wxThreadHelper**

**wxThreadHelper()**

This constructor simply initializes a member variable.

**wxThreadHelper::m\_thread**

**wxThread \* m\_thread**

the actual *wxThread* (p. 1670) object.

**wxThreadHelper::~~wxThreadHelper**

**~wxThreadHelper()**

The destructor frees the resources associated with the thread.

**wxThreadHelper::Create**

**wxThreadError Create(unsigned int stackSize = 0)**

Creates a new thread. The thread object is created in the suspended state, and you should call *GetThread()->Run()* (p. 1676) to start running it. You may optionally specify the stack size to be allocated to it (Ignored on platforms that don't support setting it explicitly, eg. Unix).

**Return value**

One of:

<b>wxTHREAD_NO_ERROR</b>	There was no error.
<b>wxTHREAD_NO_RESOURCE</b>	There were insufficient resources to create a new thread.
<b>wxTHREAD_RUNNING</b>	The thread is already running.

**wxThreadHelper::Entry****virtual ExitCode Entry()**

This is the entry point of the thread. This function is pure virtual and must be implemented by any derived class. The thread execution will start here.

The returned value is the thread exit code which is only useful for joinable threads and is the value returned by *GetThread()*->*Wait()* (p. 1678).

This function is called by wxWidgets itself and should never be called directly.

**wxThreadHelper::GetThread****wxThread \* GetThread()**

This is a public function that returns the *wxThread* (p. 1670) object associated with the thread.

**wxTimer**

The *wxTimer* class allows you to execute code at specified intervals. Its precision is platform-dependent, but in general will not be better than 1ms nor worse than 1s.

There are three different ways to use this class:

1. You may derive a new class from *wxTimer* and override the *Notify* (p. 1682) member to perform the required action.
2. Or you may redirect the notifications to any *wxEvtHandler* (p. 576) derived object by using the non-default constructor or *SetOwner* (p. 1682). Then use the `EVT_TIMER` macro to connect it to the event handler which will receive *wxTimerEvent* (p. 1682) notifications.
3. Or you may use a derived class and the `EVT_TIMER` macro to connect it to an event handler defined in the derived class. If the default constructor is used, the timer object will be its own owner object, since it is derived from *wxEvtHandler*.

In any case, you must start the timer with *Start* (p. 1682) after constructing it before it actually starts sending notifications. It can be stopped later with *Stop* (p. 1682).



**Note:** A timer can only be used from the main thread.

**Derived from**

*wxEvtHandler* (p. 576) *wxObject* (p. 1148)

**Include files**

<wx/timer.h>

**See also**

*::wxStartTimer* (p. 1979), *::wxGetElapsedTime* (p. 1977), *wxStopWatch* (p. 1543)

**wxTimer::wxTimer**

**wxTimer()**

Default constructor. If you use it to construct the object and don't call *SetOwner* (p. 1682) later, you must override *Notify* (p. 1682) method to process the notifications.

**wxTimer(wxEvtHandler \*owner, int id = -1)**

Creates a timer and associates it with *owner*. Please see *SetOwner* (p. 1682) for the description of parameters.

**wxTimer::~~wxTimer**

**~wxTimer()**

Destructor. Stops the timer if it is running.

**wxTimer::GetInterval**

**int GetInterval() const**

Returns the current interval for the timer (in milliseconds).

**wxTimer::IsOneShot**

**bool IsOneShot() const**

Returns *true* if the timer is one shot, i.e. if it will stop after firing the first notification automatically.

**wxTimer::IsRunning**

**bool IsRunning() const**

Returns *true* if the timer is running, *false* if it is stopped.

**wxTimer::Notify****void Notify()**

This member should be overridden by the user if the default constructor was used and *SetOwner* (p. 1682) wasn't called.

Perform whatever action which is to be taken periodically here.

**wxTimer::SetOwner****void SetOwner(wxEventHandler \*owner, int id = -1)**

Associates the timer with the given *owner* object. When the timer is running, the owner will receive *timer events* (p. 1682) with id equal to *id* specified here.

**wxTimer::Start****bool Start(int milliseconds = -1, bool oneShot = false)**

(Re)starts the timer. If *milliseconds* parameter is -1 (value by default), 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* (p. 1682) function will be called repeatedly until the timer is stopped. If *true*, it will be called only once and the timer will stop automatically. To make your code more readable you may also use the following symbolic constants:

`wxTIMER_CONTINUOUS`    Start a normal, continuously running, timer

`wxTIMER_ONE_SHOT`     Start a one shot timer

If the timer was already running, it will be stopped by this method before restarting it.

**wxTimer::Stop****void Stop()**

Stops the timer.

**wxTimerEvent**

*wxTimerEvent* object is passed to the event handler of timer events.

For example:

```
class MyFrame : public wxFrame
{
public:
    ...
}
```

```
void OnTimer(wxTimerEvent& event);

private:
    wxTimer m_timer;
};

BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_TIMER(TIMER_ID, MyFrame::OnTimer)
END_EVENT_TABLE()

MyFrame::MyFrame()
    : m_timer(this, TIMER_ID)
{
    m_timer.Start(1000);    // 1 second interval
}

void MyFrame::OnTimer(wxTimerEvent& event)
{
    // do whatever you want to do every second here
}
```

**Derived from**

*wxEvent* (p. 572)

**Include files**

<wx/timer.h>

**See also**

*wxTimer* (p. 1680)

**wxTimerEvent::GetInterval**

**int GetInterval() const**

Returns the interval of the timer which generated this event.

**wxTimeSpan**

wxTimeSpan class represents a time interval.

**Derived from**

No base class

**Include files**

<wx/datetime.h>

**See also**

*Date classes overview* (p. 2052), *wxDateTime* (p. 348)

**Static functions**

*Milliseconds* (p. 1688)

*Millisecond* (p. 1688)

*Seconds* (p. 1689)

*Second* (p. 1689)

*Minutes* (p. 1688)

*Minute* (p. 1688)

*Hours* (p. 1687)

*Hour* (p. 1687)

*Days* (p. 1685)

*Day* (p. 1685)

*Weeks* (p. 1689)

*Week* (p. 1689)

**Constructors**

*wxTimeSpan* (p. 1689)

**Accessors**

*GetSeconds* (p. 1686)

*GetMinutes* (p. 1686)

*GetHours* (p. 1686)

*GetDays* (p. 1686)

*GetWeeks* (p. 1687)

*GetValue* (p. 1686)

**Operations**

*Add* (p. 1685)

*Subtract* (p. 1689)

*Multiply* (p. 1688)

*Negate* (p. 1688)

*Neg* (p. 1688)

*Abs* (p. 1685)

**Tests**

*IsNull* (p. 1687)

*IsPositive* (p. 1687)

*IsNegative* (p. 1687)

*IsEqualTo* (p. 1687)

*IsLongerThan* (p. 1687)

*IsShorterThan* (p. 1688)

## Formatting time spans

*Format* (p. 1685)

### **wxTimeSpan::Abs**

**wxTimeSpan Abs() const**

Returns the absolute value of the timespan: does not modify the object.

### **wxTimeSpan::Add**

**wxTimeSpan Add(const wxTimeSpan& diff) const**

**wxTimeSpan& Add(const wxTimeSpan& diff)**

**wxTimeSpan& operator+=(const wxTimeSpan& diff)**

Returns the sum of two timespans.

### **wxTimeSpan::Days**

**static wxTimespan Days(long days)**

Returns the timespan for the given number of days.

### **wxTimeSpan::Day**

**static wxTimespan Day()**

Returns the timespan for one day.

### **wxTimeSpan::Format**

**wxString Format(const wxChar \* format = wxDefaultTimeSpanFormat)**

Returns the string containing the formatted representation of the time span. The following format specifiers are allowed after %:

H	number of <b>H</b> ours
M	number of <b>M</b> inutes
S	number of <b>S</b> econds
I	number of milliseconds
D	number of <b>D</b> ays

E	number of wEeks
%	the percent character

Note that, for example, the number of hours in the description above is not well defined: it can be either the total number of hours (for example, for a time span of 50 hours this would be 50) or just the hour part of the time span, which would be 2 in this case as 50 hours is equal to 2 days and 2 hours.

`wxTimeSpan` resolves this ambiguity in the following way: if there had been, indeed, the `%D` format specified preceding the `%H`, then it is interpreted as 2. Otherwise, it is 50.

The same applies to all other format specifiers: if they follow a specifier of larger unit, only the rest part is taken, otherwise the full value is used.

### **`wxTimeSpan::GetDays`**

**`int GetDays() const`**

Returns the difference in number of days.

### **`wxTimeSpan::GetHours`**

**`int GetHours() const`**

Returns the difference in number of hours.

### **`wxTimeSpan::GetMilliseconds`**

**`wxLongLong GetMilliseconds() const`**

Returns the difference in number of milliseconds.

### **`wxTimeSpan::GetMinutes`**

**`int GetMinutes() const`**

Returns the difference in number of minutes.

### **`wxTimeSpan::GetSeconds`**

**`wxLongLong GetSeconds() const`**

Returns the difference in number of seconds.

### **`wxTimeSpan::GetValue`**

**`wxLongLong GetValue() const`**

Returns the internal representation of timespan.

**wxTimeSpan::GetWeeks****int GetWeeks() const**

Returns the difference in number of weeks.

**wxTimeSpan::Hours****static wxTimespan Hours(long hours)**

Returns the timespan for the given number of hours.

**wxTimeSpan::Hour****static wxTimespan Hour()**

Returns the timespan for one hour.

**wxTimeSpan::IsEqualTo****bool IsEqualTo(const wxTimeSpan& ts) const**

Returns `true` if two timespans are equal.

**wxTimeSpan::IsLongerThan****bool IsLongerThan(const wxTimeSpan& ts) const**

Compares two timespans: works with the absolute values, i.e. -2 hours is longer than 1 hour. Also, it will return `false` if the timespans are equal in absolute value.

**wxTimeSpan::IsNegative****bool IsNegative() const**

Returns `true` if the timespan is negative.

**wxTimeSpan::IsNull****bool IsNull() const**

Returns `true` if the timespan is empty.

**wxTimeSpan::IsPositive****bool IsPositive() const**

Returns `true` if the timespan is positive.

**wxTimeSpan::IsShorterThan****bool IsShorterThan(const wxTimeSpan& ts) const**

Compares two timespans: works with the absolute values, i.e. 1 hour is shorter than -2 hours. Also, it will return `false` if the timespans are equal in absolute value.

**wxTimeSpan::Minutes****static wxTimespan Minutes(long min)**

Returns the timespan for the given number of minutes.

**wxTimeSpan::Minute****static wxTimespan Minute()**

Returns the timespan for one minute.

**wxTimeSpan::Multiply****wxTimeSpan Multiply(int n) const****wxTimeSpan& Multiply(int n)****wxTimeSpan& operator\*=(int n)**

Multiplies timespan by a scalar.

**wxTimeSpan::Negate****wxTimeSpan Negate() const**

Returns timespan with inverted sign.

**wxTimeSpan::Neg****wxTimeSpan& Neg()****wxTimeSpan& operator-()**

Negate the value of the timespan.

**wxTimeSpan::Milliseconds****static wxTimespan Milliseconds(long ms)**

Returns the timespan for the given number of milliseconds.

**wxTimeSpan::Millisecond**



**static wxTimespan Millisecond()**

Returns the timespan for one millisecond.

**wxTimespan::Seconds**

**static wxTimespan Seconds(long sec)**

Returns the timespan for the given number of seconds.

**wxTimespan::Second**

**static wxTimespan Second()**

Returns the timespan for one second.

**wxTimespan::Subtract**

**wxTimespan Subtract(const wxTimespan&diff) const**

**wxTimespan& Subtract(const wxTimespan& diff)**

**wxTimespan& operator-=(const wxTimespan&diff)**

Returns the difference of two timespans.

**wxTimespan::Weeks**

**static wxTimespan Weeks(long weeks)**

Returns the timespan for the given number of weeks.

**wxTimespan::Week**

**static wxTimespan Week()**

Returns the timespan for one week.

**wxTimespan::wxTimespan**

**wxTimespan()**

Default constructor, constructs a zero timespan.

**wxTimespan(long hours, long min, long sec, long msec)**

Constructs timespan from separate values for each component, with the date set to 0. Hours are not restricted to 0..24 range, neither are minutes, seconds or milliseconds.

**wxTipProvider**

This is the class used together with *wxShowTip* (p. 1943) function. It must implement *GetTip* (p. 1690) function and return the current tip from it (different tip each time it is called).

You will never use this class yourself, but you need it to show startup tips with *wxShowTip*. Also, if you want to get the tips text from elsewhere than a simple text file, you will want to derive a new class from *wxTipProvider* and use it instead of the one returned by *wxCreateFileTipProvider* (p. 1935).

**Derived from**

None.

**Include files**

<wx/tipdlg.h>

**See also**

*Startup tips overview* (p. 2144), *::wxShowTip* (p. 1943)

**wxTipProvider::wxTipProvider**

**wxTipProvider**(size\_t *currentTip*)

Constructor.

*currentTip*

The starting tip index.

**wxTipProvider::GetTip**

**wxString** GetTip()

Return the text of the current tip and pass to the next one. This function is pure virtual, it should be implemented in the derived classes.

**wxTipProvider::PreprocessTip**

**virtual wxString** PreProcessTip(const wxString&*tip*)

Returns a modified tip. This function will be called immediately after read, and before being check whether it is a comment, an empty string or a string to translate. You can optionally override this in your custom user-derived class to optionally to modify the tip as soon as it is read. You can return any modification to the string. If you return *wxEmptyString*, then this tip is skipped, and the next one is read.

**wxCurrentTipProvider::GetCurrentTip**

**size\_t GetCurrentTip() const**

Returns the index of the current tip (i.e. the one which would be returned by `GetTip`).

The program usually remembers the value returned by this function after calling `wxShowTip` (p. 1943). Note that it is not the same as the value which was passed to `wxShowTip + 1` because the user might have pressed the "Next" button in the tip dialog.

**wxTipWindow**

Shows simple text in a popup tip window on creation. This is used by `wxSimpleHelpProvider` (p. 1432) to show popup help. The window automatically destroys itself when the user clicks on it or it loses the focus.

You may also use this class to emulate the tooltips when you need finer control over them than what the standard tooltips provide.

**Derived from**

`wxPopupTransientWindow`  
`wxPopupWindow`  
`wxWindow` (p. 1795)  
`wxEvtHandler` (p. 576)  
`wxObject` (p. 1148)

**Include files**

<wx/tipwin.h>

**wxTipWindow::wxTipWindow**

**wxTipWindow**(**wxWindow\*** *parent*, **const wxString&** *text*, **wxCoord** *maxLength* = 100, **wxTipWindow\*\*** *windowPtr*, **wxRect** *rectBounds* = *NULL*)

Constructor. The tip is shown immediately after the window is constructed.

**Parameters**

*parent*

The parent window, must be non-NULL

*text*

The text to show, may contain the new line characters

*maxLength*

The length of each line, in pixels. Set to a very large value to avoid wrapping lines

*windowPtr*

Simply passed to *SetTipWindowPtr* (p. 1692) below, please see its documentation for the description of this parameter

*rectBounds*

If non-NULL, passed to *SetBoundingRect* (p. 1692) below, please see its documentation for the description of this parameter

### **wxTipWindow::SetTipWindowPtr**

**void SetTipWindowPtr(wxTipWindow\*\* windowPtr)**

When the tip window closes itself (which may happen at any moment and unexpectedly to the caller) it may NULL out the pointer pointed to by *it windowPtr*. This is helpful to avoid dereferencing the tip window which had been already closed and deleted.

### **wxTipWindow::SetBoundingRect**

**void SetBoundingRect(const wxRect& rectBound)**

By default, the tip window disappears when the user clicks the mouse or presses a keyboard key or if it loses focus in any other way - for example because the user switched to another application window.

Additionally, if a non-empty *rectBound* is provided, the tip window will also automatically close if the mouse leaves this area. This is useful to dismiss the tip mouse when the mouse leaves the object it is associated with.

#### **Parameters**

*rectBound*

The bounding rectangle for the mouse in the screen coordinates

## **wxToggleButton**

*wxToggleButton* is a button that stays pressed when clicked by the user. In other words, it is similar to *wxCheckBox* (p. 180) in functionality but looks like a *wxButton* (p. 164).

You can see *wxToggleButton* in action in the sixth page of the *controls* (p. 2032) sample.

#### **Derived from**

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

#### **Include files**

<wx/tglbtn.h>

## Window styles

There are no special styles for `wxToggleButton`.

See also *window styles overview* (p. 2089).

## Event handling

**EVT\_TOGGLEBUTTON(id, func)**      Handles a toggle button click event.

## See also

*wxCheckBox* (p. 180), *wxButton* (p. 164)

## `wxToggleButton::wxToggleButton`

### `wxToggleButton()`

Default constructor.

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

Constructor, creating and showing a toggle button.

## Parameters

*parent*

Parent window. Must not be `NULL`.

*id*

Toggle button identifier. A value of -1 indicates a default value.

*label*

Text to be displayed next to the toggle button.

*pos*

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

*size*

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

*style*

Window style. See *wxToggleButton* (p. 1692).

*validator*

Window validator.

*name*

Window name.

### See also

*wxToggleButton::Create* (p. 1694), *wxValidator* (p. 1767)

## **wxToggleButton::~wxToggleButton**

**~wxToggleButton()**

Destructor, destroying the toggle button.

## **wxToggleButton::Create**

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

Creates the toggle button for two-step construction. See *wxToggleButton::wxToggleButton* (p. 1693) for details.

## **wxToggleButton::GetValue**

**bool GetValue() const**

Gets the state of the toggle button.

### Return value

Returns `true` if it is pressed, `false` otherwise.

## **wxToggleButton::SetValue**

**void SetValue(const bool state)**

Sets the toggle button to the given state. This does not cause `aeVT_TOGGLEBUTTON` event to be emitted.

### Parameters

*state*

If `true`, the button is pressed.

## **wxToolBar**

The name `wxToolBar` is defined to be a synonym for one of the following classes:

- **wxToolBar95** The native Windows 95 toolbar. Used on Windows 95, NT 4 and above.
- **wxToolBarMSW** A Windows implementation. Used on 16-bit Windows.
- **wxToolBarGTK** The GTK toolbar.

#### Derived from

`wxToolBarBase`  
`wxControl` (p. 285)  
`wxWindow` (p. 1795)  
`wxEvtHandler` (p. 576)  
`wxObject` (p. 1148)

#### Include files

<wx/toolbar.h> (to allow `wxWidgets` to select an appropriate toolbar class)  
<wx/tbarbase.h> (the base class)

#### Remarks

You may also create a toolbar that is managed by the frame, by calling `wxFrame::CreateToolBar` (p. 686). Under Pocket PC, you should *always* use this function for creating the toolbar to be managed by the frame, so that `wxWidgets` can use a combined menubar and toolbar. Where you manage your own toolbars, create a `wxToolBar` as usual.

The meaning of a "separator" is a vertical line under Windows and simple space under GTK+.

**wxToolBar95:** Note that this toolbar paints tools to reflect system-wide colours. If you use more than 16 colours in your tool bitmaps, you may wish to suppress this behaviour, otherwise system colours in your bitmaps will inadvertently be mapped to system colours. To do this, set the `msw.remap` system option before creating the toolbar:

```
wxSystemOptions::SetOption(wxT("msw.remap"), 0);
```

If you wish to use 32-bit images (which include an alpha channel for transparency) use:

```
wxSystemOptions::SetOption(wxT("msw.remap"), 2);
```

then colour remapping is switched off, and a transparent background used. But only use this option under Windows XP with true colour:

```
(wxTheApp->GetComCtl32Version() >= 600 && ::wxDisplayDepth() >= 32)
```

#### Window styles

- |                      |   |
|----------------------|---|
| <b>wxTB_FLAT</b>     | Gives the toolbar a flat look (Windows and GTK only). |
| <b>wxTB_DOCKABLE</b> | Makes the toolbar floatable and dockable (GTK only).  |

<b>wxB_HORIZONTAL</b>	Specifies horizontal layout (default).
<b>wxB_VERTICAL</b>	Specifies vertical layout.
<b>wxB_TEXT</b>	Shows the text in the toolbar buttons; by default only icons are shown.
<b>wxB_NOICONS</b>	Specifies no icons in the toolbar buttons; by default they are shown.
<b>wxB_NODIVIDER</b>	Specifies no divider (border) above the toolbar (Windows only).
<b>wxB_NOALIGN</b>	Specifies no alignment with the parent window (Windows only, not very useful).
<b>wxB_HORZ_LAYOUT</b>	Shows the text and the icons alongside, not vertically stacked (Windows and GTK 2 only). This style must be used with <b>wxB_TEXT</b> .
<b>wxB_HORZ_TEXT</b>	Combination of <b>wxB_HORZ_LAYOUT</b> and <b>wxB_TEXT</b> .
<b>wxB_NO_TOOLTIPS</b>	Don't show the short help tooltips for the tools when the mouse hovers over them.
<b>wxB_BOTTOM</b>	Align the toolbar at the bottom of parent window.
<b>wxB_RIGHT</b>	Align the toolbar at the right side of parent window.

See also *window styles overview* (p. 2089). Note that the Win32 native toolbar ignores **wxB\_NOICONS** style. Also, toggling the **wxB\_TEXT** works only if the style was initially on.

### Event handling

The toolbar class emits menu commands in the same way that a frame menubar does, so you can use one **EVT\_MENU** macro for both a menu item and a toolbar button. The event handler functions take a **wxCommandEvent** argument. For most event macros, the identifier of the tool is passed, but for **EVT\_TOOL\_ENTER** the toolbar window identifier is passed and the tool identifier is retrieved from the **wxCommandEvent**. This is because the identifier may be -1 when the mouse moves off a tool, and -1 is not allowed as an identifier in the event system.

<b>EVT_TOOL(id, func)</b>	Process a <b>wxEVT_COMMAND_TOOL_CLICKED</b> event (a synonym for <b>wxEVT_COMMAND_MENU_SELECTED</b> ). Pass the id of the tool.
<b>EVT_MENU(id, func)</b>	The same as <b>EVT_TOOL</b> .
<b>EVT_TOOL_RANGE(id1, id2, func)</b>	Process a <b>wxEVT_COMMAND_TOOL_CLICKED</b> event for a range of identifiers. Pass the ids of the



	tools.
<b>EVT_MENU_RANGE(id1, id2, func)</b>	The same as EVT_TOOL_RANGE.
<b>EVT_TOOL_RCLICKED(id, func)</b>	Process a wxEVT_COMMAND_TOOL_RCLICKED event. Pass the id of the tool.
<b>EVT_TOOL_RCLICKED_RANGE(id1, id2, func)</b>	Process a wxEVT_COMMAND_TOOL_RCLICKED event for a range of ids. Pass the ids of the tools.
<b>EVT_TOOL_ENTER(id, func)</b>	Process a wxEVT_COMMAND_TOOL_ENTER event. Pass the id of the toolbar itself. The value of wxCommandEvent::GetSelection is the tool id, or -1 if the mouse cursor has moved off a tool.

**See also**

*Toolbar overview* (p. 2138), *wxScrolledWindow* (p. 1414)

**wxToolBar::wxToolBar****wxToolBar()**

Default constructor.

**wxToolBar(wxWindow\* parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = wxTB\_HORIZONTAL | wxNO\_BORDER, const wxString& name = wxPanelNameStr)**

Constructs a toolbar.

**Parameters**

*parent*

Pointer to a parent window.

*id*

Window identifier. If -1, will automatically create an identifier.

*pos*

Window position. wxDefaultPosition is (-1, -1) which indicates that wxWidgets should generate a default position for the window. If using the wxWindow class directly, supply an actual position.

*size*

Window size. `wxDefaultSize` is `(-1, -1)` which indicates that `wxWidgets` should generate a default size for the window.

*style*

Window style. See `wxToolBar` (p. 1694) for details.

*name*

Window name.

### Remarks

After a toolbar is created, you use `wxToolBar::AddTool` (p. 1698) and perhaps `wxToolBar::AddSeparator` (p. 1698), and then you must call `wxToolBar::Realize` (p. 1707) to construct and display the toolbar tools.

You may also create a toolbar that is managed by the frame, by calling `wxFrame::CreateToolBar` (p. 686).

### **wxToolBar::~~wxToolBar**

**void ~wxToolBar()**

Toolbar destructor.

### **wxToolBar::AddControl**

**bool AddControl(wxControl\* control)**

Adds any control to the toolbar, typically e.g. a combobox.

*control*

The control to be added.

### **wxToolBar::AddSeparator**

**void AddSeparator()**

Adds a separator for spacing groups of tools.

### See also

`wxToolBar::AddTool` (p. 1698), `wxToolBar::SetToolSeparation` (p. 1710)

### **wxToolBar::AddTool**

**wxToolBarToolBase\* AddTool(int toolId, const wxString& label, const wxBitmap& bitmap1, const wxString& shortHelpString = "", wxItemKind kind = wxITEM\_NORMAL)**

**wxToolBarToolBase\* AddTool(int toolId, const wxString& label, const wxBitmap& bitmap1, const wxBitmap& bitmap2 = wxNullBitmap, wxItemKind kind =**

*wxITEM\_NORMAL*, **const wxString& shortHelpString = ""**, **const wxString& longHelpString = ""**, **wxObject\* clientData = NULL**)

**wxToolBarToolBase\* AddTool(wxToolBarToolBase\* tool)**

Adds a tool to the toolbar. The first (short and most commonly used) version has fewer parameters than the full version at the price of not being able to specify some of the more rarely used button features. The last version allows you to add an existing tool.

### Parameters

*toolId*

An integer by which the tool may be identified in subsequent operations.

*kind*

May be *wxITEM\_NORMAL* for a normal button (default), *wxITEM\_CHECK* for a checkable tool (such tool stays pressed after it had been toggled) or *wxITEM\_RADIO* for a checkable tool which makes part of a radio group of tools each of which is automatically unchecked whenever another button in the group is checked

*bitmap1*

The primary tool bitmap.

*bitmap2*

The bitmap used when the tool is disabled. If it is equal to *wxNullBitmap*, the disabled bitmap is automatically generated by greying the normal one.

*shortHelpString*

This string is used for the tools tooltip

*longHelpString*

This string is shown in the statusbar (if any) of the parent frame when the mouse pointer is inside the tool

*clientData*

An optional pointer to client data which can be retrieved later using *wxToolBar::GetToolClientData* (p. 1703).

*tool*

The tool to be added.

### Remarks

After you have added tools to a toolbar, you must call *wxToolBar::Realize* (p. 1707) in order to have the tools appear.

**See also**

*wxToolBar::AddSeparator* (p. 1698), *wxToolBar::AddCheckTool* (p. 1700),  
*wxToolBar::AddRadioTool* (p. 1700), *wxToolBar::InsertTool* (p. 1705),  
*wxToolBar::DeleteTool* (p. 1700), *wxToolBar::Realize* (p. 1707)

**wxToolBar::AddCheckTool**

**wxToolBarToolBase\* AddCheckTool**(int *toolId*, const wxString& *label*, const wxBitmap& *bitmap1*, const wxBitmap& *bitmap2*, const wxString& *shortHelpString* = "", const wxString& *longHelpString* = "", wxObject\* *clientData* = NULL)

Adds a new check (or toggle) tool to the toolbar. The parameters are the same as in *wxToolBar::AddTool* (p. 1698).

**See also**

*wxToolBar::AddTool* (p. 1698)

**wxToolBar::AddRadioTool**

**wxToolBarToolBase\* AddRadioTool**(int *toolId*, const wxString& *label*, const wxBitmap& *bitmap1*, const wxBitmap& *bitmap2*, const wxString& *shortHelpString* = "", const wxString& *longHelpString* = "", wxObject\* *clientData* = NULL)

Adds a new radio tool to the toolbar. Consecutive radio tools form a radio group such that exactly one button in the group is pressed at any moment, in other words whenever a button in the group is pressed the previously pressed button is automatically released. You should avoid having the radio groups of only one element as it would be impossible for the user to use such button.

By default, the first button in the radio group is initially pressed, the others are not.

**See also**

*wxToolBar::AddTool* (p. 1698)

**wxToolBar::ClearTools**

**void ClearTools()**

Deletes all the tools in the toolbar.

**wxToolBar::DeleteTool**

**bool DeleteTool**(int *toolId*)

Removes the specified tool from the toolbar and deletes it. If you don't want to delete the tool, but just to remove it from the toolbar (to possibly add it back later), you may use *RemoveTool* (p. 1707) instead.

Note that it is unnecessary to call *Realize* (p. 1707) for the change to take place, it will

happen immediately.

Returns true if the tool was deleted, false otherwise.

### See also

*DeleteToolByPos* (p. 1701)

## **wxToolBar::DeleteToolByPos**

**bool DeleteToolByPos(size\_t pos)**

This function behaves like *DeleteTool* (p. 1700) but it deletes the tool at the specified position and not the one with the given id.

## **wxToolBar::EnableTool**

**void EnableTool(int toolId, const bool enable)**

Enables or disables the tool.

### Parameters

*toolId*

Tool to enable or disable.

*enable*

If true, enables the tool, otherwise disables it.

**NB:** This function should only be called after *Realize* (p. 1707).

### Remarks

Some implementations will change the visible state of the tool to indicate that it is disabled.

### See also

*wxToolBar::GetToolEnabled* (p. 1703), *wxToolBar::ToggleTool* (p. 1711)

## **wxToolBar::FindById**

**wxToolBarToolBase\* FindById(int id)**

Returns a pointer to the tool identified by *id* or `NULL` if no corresponding tool is found.

## **wxToolBar::FindControl**

**wxControl\* FindControl(int id)**

Returns a pointer to the control identified by *id* or `NULL` if no corresponding control is found.

**wxToolBar::FindToolForPosition****wxToolBarToolBase\* FindToolForPosition(wxCoord x, wxCoord y) const**

Finds a tool for the given mouse position.

**Parameters**

*x*

X position.

*y*

Y position.

**Return value**

A pointer to a tool if a tool is found, or `NULL` otherwise.

**Remarks**

Currently not implemented in wxGTK (always returns `NULL` there).

**wxToolBar::GetToolsCount****int GetToolsCount() const**

Returns the number of tools in the toolbar.

**wxToolBar::GetToolSize****wxSize GetToolSize()**

Returns the size of a whole button, which is usually larger than a tool bitmap because of added 3D effects.

**See also**

*wxToolBar::SetToolBitmapSize* (p. 1708), *wxToolBar::GetToolBitmapSize* (p. 1702)

**wxToolBar::GetToolBitmapSize****wxSize GetToolBitmapSize()**

Returns the size of bitmap that the toolbar expects to have. The default bitmap size is 16 by 15 pixels.

**Remarks**

Note that this is the size of the bitmap you pass to *wxToolBar::AddTool* (p. 1698), and not the eventual size of the tool button.

**See also**

*wxToolBar::SetToolBitmapSize* (p. 1708), *wxToolBar::GetToolSize* (p. 1702)

**wxToolBar::GetMargins**

**wxSize GetMargins() const**

Returns the left/right and top/bottom margins, which are also used for inter-tools spacing.

**See also**

*wxToolBar::SetMargins* (p. 1708)

**wxToolBar::GetToolClientData**

**wxObject\* GetToolClientData(int toolId) const**

Get any client data associated with the tool.

**Parameters**

*toolId*

Id of the tool, as passed to *wxToolBar::AddTool* (p. 1698).

**Return value**

Client data, or `NULL` if there is none.

**wxToolBar::GetToolEnabled**

**bool GetToolEnabled(int toolId) const**

Called to determine whether a tool is enabled (responds to user input).

**Parameters**

*toolId*

Id of the tool in question.

**Return value**

true if the tool is enabled, false otherwise.

**See also**

*wxToolBar::EnableTool* (p. 1701)

**wxToolBar::GetToolLongHelp**

**wxString GetToolLongHelp(int toolId) const**

Returns the long help for the given tool.

**Parameters**

*toolId*

The tool in question.

**See also**

*wxToolBar::SetToolLongHelp* (p. 1709), *wxToolBar::SetToolShortHelp* (p. 1710)

**wxToolBar::GetToolPacking**

**int GetToolPacking() const**

Returns the value used for packing tools.

**See also**

*wxToolBar::SetToolPacking* (p. 1710)

**wxToolBar::GetToolPos**

**int GetToolPos(int *toolId*) const**

Returns the tool position in the toolbar, or `wxNOT_FOUND` if the tool is not found.

**wxToolBar::GetToolSeparation**

**int GetToolSeparation() const**

Returns the default separator size.

**See also**

*wxToolBar::SetToolSeparation* (p. 1710)

**wxToolBar::GetToolShortHelp**

**wxString GetToolShortHelp(int *toolId*) const**

Returns the short help for the given tool.

**Parameters**

*toolId*

The tool in question.

**See also**

*wxToolBar::GetToolLongHelp* (p. 1703), *wxToolBar::SetToolShortHelp* (p. 1710)



**wxToolBar::GetToolState****bool GetToolState(int toolId) const**

Gets the on/off state of a toggle tool.

**Parameters***toolId*

The tool in question.

**Return value**

true if the tool is toggled on, false otherwise.

**See also**

*wxToolBar::ToggleTool* (p. 1711)

**wxToolBar::InsertControl****wxToolBarToolBase \* InsertControl(size\_t pos, wxControl \*control)**

Inserts the control into the toolbar at the given position.

You must call *Realize* (p. 1707) for the change to take place.

**See also**

*AddControl* (p. 1698),  
*InsertTool* (p. 1705)

**wxToolBar::InsertSeparator****wxToolBarToolBase \* InsertSeparator(size\_t pos)**

Inserts the separator into the toolbar at the given position.

You must call *Realize* (p. 1707) for the change to take place.

**See also**

*AddSeparator* (p. 1698),  
*InsertTool* (p. 1705)

**wxToolBar::InsertTool****wxToolBarToolBase \* InsertTool(size\_t pos, int toolId, const wxBitmap& bitmap1, const wxBitmap& bitmap2 = wxNullBitmap, bool isToggle = false, wxObject\* clientData = NULL, const wxString& shortHelpString = "", const wxString& longHelpString = "")****wxToolBarToolBase \* InsertTool(size\_t pos, wxToolBarToolBase\* tool)**

Inserts the tool with the specified attributes into the toolbar at the given position.

You must call *Realize* (p. 1707) for the change to take place.

### See also

*AddTool* (p. 1698),  
*InsertControl* (p. 1705),  
*InsertSeparator* (p. 1705)

## **wxToolBar::OnLeftClick**

**bool OnLeftClick(int toolId, bool toggleDown)**

Called when the user clicks on a tool with the left mouse button.

This is the old way of detecting tool clicks; although it will still work, you should use the EVT\_MENU or EVT\_TOOL macro instead.

### Parameters

*toolId*

The identifier passed to *wxToolBar::AddTool* (p. 1698).

*toggleDown*

true if the tool is a toggle and the toggle is down, otherwise is false.

### Return value

If the tool is a toggle and this function returns false, the toggle toggle state (internal and visual) will not be changed. This provides a way of specifying that toggle operations are not permitted in some circumstances.

### See also

*wxToolBar::OnMouseEnter* (p. 1706), *wxToolBar::OnRightClick* (p. 1707)

## **wxToolBar::OnMouseEnter**

**void OnMouseEnter(int toolId)**

This is called when the mouse cursor moves into a tool or out of the toolbar.

This is the old way of detecting mouse enter events; although it will still work, you should use the EVT\_TOOL\_ENTER macro instead.

### Parameters

*toolId*

Greater than -1 if the mouse cursor has moved into the tool, or -1 if the mouse cursor has moved. The programmer can override this to provide extra information about the

tool, such as a short description on the status line.

### Remarks

With some derived toolbar classes, if the mouse moves quickly out of the toolbar, wxWidgets may not be able to detect it. Therefore this function may not always be called when expected.

### **wxToolBar::OnRightClick**

**void OnRightClick(int toolId, float x, float y)**

Called when the user clicks on a tool with the right mouse button. The programmer should override this function to detect right tool clicks.

This is the old way of detecting tool right clicks; although it will still work, you should use the EVT\_TOOL\_RCLICKED macro instead.

### Parameters

*toolId*

The identifier passed to *wxToolBar::AddTool* (p. 1698).

*x*

The x position of the mouse cursor.

*y*

The y position of the mouse cursor.

### Remarks

A typical use of this member might be to pop up a menu.

### See also

*wxToolBar::OnMouseEnter* (p. 1706), *wxToolBar::OnLeftClick* (p. 1706)

### **wxToolBar::Realize**

**bool Realize()**

This function should be called after you have added tools.

### **wxToolBar::RemoveTool**

**wxToolBarToolBase \* RemoveTool(int id)**

Removes the given tool from the toolbar but doesn't delete it. This allows to insert/add this tool back to this (or another) toolbar later.

Note that it is unnecessary to call *Realize* (p. 1707) for the change to take place, it will

happen immediately.

**See also**

*DeleteTool* (p. 1700)

**wxToolBar::SetBitmapResource**

**void SetBitmapResource(int resourceId)**

Sets the bitmap resource identifier for specifying tool bitmaps as indices into a custom bitmap. Windows CE only.

**wxToolBar::SetMargins**

**void SetMargins(const wxSize& size)**

**void SetMargins(int x, int y)**

Set the values to be used as margins for the toolbar.

**Parameters**

*size*

Margin size.

*x*

Left margin, right margin and inter-tool separation value.

*y*

Top margin, bottom margin and inter-tool separation value.

**Remarks**

This must be called before the tools are added if absolute positioning is to be used, and the default (zero-size) margins are to be overridden.

**See also**

*wxToolBar::GetMargins* (p. 1703), *wxSize* (p. 1440)

**wxToolBar::SetToolBitmapSize**

**void SetToolBitmapSize(const wxSize& size)**

Sets the default size of each tool bitmap. The default bitmap size is 16 by 15 pixels.

**Parameters**

*size*

The size of the bitmaps in the toolbar.

**Remarks**

This should be called to tell the toolbar what the tool bitmap size is. Call it before you add tools.

Note that this is the size of the bitmap you pass to *wxToolBar::AddTool* (p. 1698), and not the eventual size of the tool button.

**See also**

*wxToolBar::GetToolBitmapSize* (p. 1702), *wxToolBar::GetToolSize* (p. 1702)

**wxToolBar::SetToolClientData**

**void SetToolClientData**(int *id*, wxObject\* *clientData*)

Sets the client data associated with the tool.

**wxToolBar::SetToolDisabledBitmap**

**void SetToolDisabledBitmap**(int *id*, const wxBitmap& *bitmap*)

Sets the bitmap to be used by the tool with the given ID when the tool is in a disabled state. This can only be used on Button tools, not controls. NOTE: The native toolbar classes on the main platforms all synthesize the disabled bitmap from the normal bitmap, so this function will have no effect on those platforms.

**wxToolBar::SetToolLongHelp**

**void SetToolLongHelp**(int *toolId*, const wxString& *helpString*)

Sets the long help for the given tool.

**Parameters**

*toolId*

The tool in question.

*helpString*

A string for the long help.

**Remarks**

You might use the long help for displaying the tool purpose on the status line.

**See also**

*wxToolBar::GetToolLongHelp* (p. 1703), *wxToolBar::SetToolShortHelp* (p. 1710),

**wxToolBar::SetToolPacking****void SetToolPacking**(int *packing*)

Sets the value used for spacing tools. The default value is 1.

**Parameters***packing*

The value for packing.

**Remarks**

The packing is used for spacing in the vertical direction if the toolbar is horizontal, and for spacing in the horizontal direction if the toolbar is vertical.

**See also**

*wxToolBar::GetToolPacking* (p. 1704)

**wxToolBar::SetToolShortHelp****void SetToolShortHelp**(int *toolId*, const wxString& *helpString*)

Sets the short help for the given tool.

**Parameters***toolId*

The tool in question.

*helpString*

The string for the short help.

**Remarks**

An application might use short help for identifying the tool purpose in a tooltip.

**See also**

*wxToolBar::GetToolShortHelp* (p. 1704), *wxToolBar::SetToolLongHelp* (p. 1709)

**wxToolBar::SetToolNormalBitmap****void SetToolNormalBitmap**(int *id*, const wxBitmap& *bitmap*)

Sets the bitmap to be used by the tool with the given ID. This can only be used on Button tools, not controls.

**wxToolBar::SetToolSeparation**

**void SetToolSeparation(int *separation*)**

Sets the default separator size. The default value is 5.

**Parameters**

*separation*

The separator size.

**See also**

*wxToolBar::AddSeparator* (p. 1698)

**wxToolBar::ToggleTool**

**void ToggleTool(int *toolId*, const bool *toggle*)**

Toggles a tool on or off. This does not cause any event to get emitted.

**Parameters**

*toolId*

Tool in question.

*toggle*

If true, toggles the tool on, otherwise toggles it off.

**Remarks**

Only applies to a tool that has been specified as a toggle tool.

**See also**

*wxToolBar::GetToolState* (p. 1705)

**wxToolbook**

*wxToolbook* is a class similar to *wxNotebook* (p. 1135) but which uses a *wxToolBar* (p. 1694) to show the labels instead of the tabs.

There is no documentation for this class yet but its usage is identical to *wxNotebook* (except for the features clearly related to tabs only), so please refer to that class documentation for now. You can also use the *notebook sample* (p. 2036) to see *wxToolbook* in action.

**Derived from**

*wxBookCtrlBase*  
*wxControl* (p. 285)  
*wxControl* (p. 285)

*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/toolbook.h>

**Window styles**

**wxBK\_DEFAULT** Choose the default location for the labels depending on the current platform (currently always the top).

**See also**

*wxBookCtrl* (p. 2127), *wxNotebook* (p. 1135), *notebook sample* (p. 2036)

## **wxToolTip**

This class holds information about a tooltip associated with a window (see *wxWindow::SetToolTip* (p. 1847)).

The two static methods, *wxToolTip::Enable* (p. 1712) and *wxToolTip::SetDelay* (p. 1712) can be used to globally alter tooltips behaviour.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/tooltip.h>

### **wxToolTip::Enable**

**static void Enable**(bool *flag*)

Enable or disable tooltips globally.

May not be supported on all platforms (eg. wxCocoa).

### **wxToolTip::SetDelay**

**static void SetDelay**(long *msecs*)

Set the delay after which the tooltip appears.

May not be supported on all platforms (eg. wxCocoa).

### **wxToolTip::wxToolTip**



**wxToolTip(const wxString& tip)**

Constructor.

**wxToolTip::SetTip**

**void SetTip(const wxString& tip)**

Set the tooltip text.

**wxToolTip::GetTip**

**wxString GetTip() const**

Get the tooltip text.

**wxToolTip::GetWindow**

**wxWindow\* GetWindow() const**

Get the associated window.

## **wxTopLevelWindow**

*wxTopLevelWindow* is a common base class for *wxDialog* (p. 496) and *wxFrame* (p. 682). It is an abstract base class meaning that you never work with objects of this class directly, but all of its methods are also applicable for the two classes above.

### **Derived from**

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### **Include files**

<wx/toplevel.h>

**wxTopLevelWindow::CanSetTransparent**

**virtual bool CanSetTransparent()**

Returns `true` if the platform supports making the window translucent.

### **See also**

*wxTopLevelWindow::SetTransparent* (p. 1720)

**wxTopLevelWindow::EnableCloseButton**

**bool EnableCloseButton(bool enable = true)**

Enables or disables the Close button (most often in the right upper corner of a dialog) and the Close entry of the system menu (most often in the left upper corner of the dialog). Currently only implemented for wxMSW and wxGTK. Returns true if operation was successful. This may be wrong on X11 (including GTK+) where the window manager may not support this operation and there is no way to find out.

**wxTopLevelWindow::GetDefaultItem**

**wxWindow \* GetDefaultItem() const**

Returns a pointer to the button which is the default for this window, or `NULL`. The default button is the one activated by pressing the Enter key.

**wxTopLevelWindow::GetIcon**

**const wxIcon& GetIcon() const**

Returns the standard icon of the window. The icon will be invalid if it hadn't been previously set by *SetIcon* (p. 1717).

**See also**

*GetIcons* (p. 1714)

**wxTopLevelWindow::GetIcons**

**const wxIconBundle& GetIcons() const**

Returns all icons associated with the window, there will be none of them if neither *SetIcon* (p. 1717) nor *SetIcons* (p. 1717) had been called before.

Use *GetIcon* (p. 1714) to get the main icon of the window.

**See also**

*wxIconBundle* (p. 901)

**wxTopLevelWindow::GetTitle**

**wxString GetTitle() const**

Gets a string containing the window title.

**See also**

*wxTopLevelWindow::SetTitle* (p. 1719)

**wxTopLevelWindow::HandleSettingChange**

**virtual bool HandleSettingChange(WXWPARAM wParam, WXLPARAM lParam)**

Unique to the wxWinCE port. Responds to showing/hiding SIP (soft input panel) area and resize window accordingly. Override this if you want to avoid resizing or do additional operations.

### **wxTopLevelWindow::IsActive**

**bool IsActive() const**

Returns `true` if this window is currently active, i.e. if the user is currently working with it.

### **wxTopLevelWindow::IsAlwaysMaximized**

**virtual bool IsAlwaysMaximized() const**

Returns `true` if this window is expected to be always maximized, either due to platform policy or due to local policy regarding particular class.

### **wxTopLevelWindow::Iconize**

**void Iconize(bool iconize)**

Iconizes or restores the window.

#### **Parameters**

*iconize*

If `true`, iconizes the window; if `false`, shows and restores it.

#### **See also**

*wxTopLevelWindow::IsIconized* (p. 1715), *wxTopLevelWindow::Maximize* (p. 1716).

### **wxTopLevelWindow::IsFullScreen**

**bool IsFullScreen()**

Returns `true` if the window is in fullscreen mode.

#### **See also**

*wxTopLevelWindow::ShowFullScreen* (p. 1720)

### **wxTopLevelWindow::IsIconized**

**bool IsIconized() const**

Returns `true` if the window is iconized.

### **wxTopLevelWindow::IsMaximized**

**bool IsMaximized() const**

Returns `true` if the window is maximized.

**wxTopLevelWindow::IsUsingNativeDecorations****bool IsUsingNativeDecorations() const**

*This method is specific to wxUniversal port*

Returns `true` if this window is using native decorations, `false` if we draw them ourselves.

**See also**

*UseNativeDecorations* (p. 1721),  
*UseNativeDecorationsByDefault* (p. 1721)

**wxTopLevelWindow::Maximize****void Maximize(bool maximize)**

Maximizes or restores the window.

**Parameters**

*maximize*

If `true`, maximizes the window, otherwise it restores it.

**See also**

*wxTopLevelWindow::Iconize* (p. 1715)

**wxTopLevelWindow::RequestUserAttention****void RequestUserAttention(int flags = wxUSER\_ATTENTION\_INFO)**

Use a system-dependent way to attract users attention to the window when it is in background.

*flags* may have the value of either `wxUSER_ATTENTION_INFO` (default) or `wxUSER_ATTENTION_ERROR` which results in a more drastic action. When in doubt, use the default value.

Note that this function should normally be only used when the application is not already in foreground.

This function is currently implemented for Win32 where it flashes the window icon in the taskbar, and for wxGTK with task bars supporting it.

**wxTopLevelWindow::SetDefaultItem**

**void SetDefaultItem(wxWindow \*win)**

Changes the default item for the panel, usually *win* is a button.

**See also**

*GetDefaultItem* (p. 1714)

**wxTopLevelWindow::SetIcon**

**void SetIcon(const wxIcon& icon)**

Sets the icon for this window.

**Parameters**

*icon*

The icon to associate with this window.

**Remarks**

The window takes a 'copy' of *icon*, but since it uses reference counting, the copy is very quick. It is safe to delete *icon* after calling this function.

See also *wxIcon* (p. 894).

**wxTopLevelWindow::SetIcons**

**void SetIcons(const wxIconBundle& icons)**

Sets several icons of different sizes for this window: this allows to use different icons for different situations (e.g. task switching bar, taskbar, window title bar) instead of scaling, with possibly bad looking results, the only icon set by *SetIcon* (p. 1717).

**Parameters**

*icons*

The icons to associate with this window.

**See also**

*wxIconBundle* (p. 901).

**wxTopLevelWindow::SetLeftMenu**

**void SetLeftMenu(int id = wxID\_ANY, const wxString& label = wxEmptyString, wxMenu \* subMenu = NULL)**

Sets action or menu activated by pressing left hardware button on the smart phones. Unavailable on full keyboard machines.

**Parameters**

*id*

Identifier for this button.

*label*

Text to be displayed on the screen area dedicated to this hardware button.

*subMenu*

The menu to be opened after pressing this hardware button.

### **See also**

*wxTopLevelWindow::SetRightMenu* (p. 1719).

### **wxTopLevelWindow::SetMaxSize**

**void SetMaxSize(const wxSize& size)**

A simpler interface for setting the size hints than *SetSizeHints* (p. 1718).

### **wxTopLevelWindow::SetMinSize**

**void SetMinSize(const wxSize& size)**

A simpler interface for setting the size hints than *SetSizeHints* (p. 1718).

### **wxTopLevelWindow::SetSizeHints**

**virtual void SetSizeHints(int minW, int minH, int maxW=-1, int maxH=-1, int incW=-1, int incH=-1)**

**void SetSizeHints(const wxSize& minSize, const wxSize& maxSize=wxDefaultSize, const wxSize& incSize=wxDefaultSize)**

Allows specification of minimum and maximum window sizes, and window size increments. If a pair of values is not set (or set to -1), the default values will be used.

*incW*

Specifies the increment for sizing the width (Motif/Xt only).

*incH*

Specifies the increment for sizing the height (Motif/Xt only).

*incSize*

Increment size (Motif/Xt only).

### **Remarks**

If this function is called, the user will not be able to size the window outside the given

bounds. The resizing increments are only significant under Motif or Xt.

### **wxTopLevelWindow::SetRightMenu**

**void SetRightMenu**(int *id* = *wxID\_ANY*, const **wxString&** *label* = *wxEmptyString*, **wxMenu** \* *subMenu* = *NULL*)

Sets action or menu activated by pressing right hardware button on the smart phones. Unavailable on full keyboard machines.

#### **Parameters**

*id*

Identifier for this button.

*label*

Text to be displayed on the screen area dedicated to this hardware button.

*subMenu*

The menu to be opened after pressing this hardware button.

#### **See also**

*wxTopLevelWindow::SetLeftMenu* (p. 1717).

### **wxTopLevelWindow::SetShape**

**bool SetShape**(const **wxRegion&** *region*)

If the platform supports it, sets the shape of the window to that depicted by *region*. The system will not display or respond to any mouse event for the pixels that lie outside of the region. To reset the window to the normal rectangular shape simply call *SetShape* again with an empty region. Returns true if the operation is successful.

### **wxTopLevelWindow::SetTitle**

**virtual void SetTitle**(const **wxString&** *title*)

Sets the window title.

#### **Parameters**

*title*

The window title.

#### **See also**

*wxTopLevelWindow::GetTitle* (p. 1714)

**wxTopLevelWindow::SetTransparent****virtual bool SetTransparent(int *alpha*)**

If the platform supports it will set the window to be translucent

**Parameters***alpha*

Determines how opaque or transparent the window will be, if the platform supports the operation. A value of 0 sets the window to be fully transparent, and a value of 255 sets the window to be fully opaque.

Returns `true` if the transparency was successfully changed.

**wxTopLevelWindow::ShouldPreventAppExit****virtual bool ShouldPreventAppExit() const**

This virtual function is not meant to be called directly but can be overridden to return `false` (it returns `true` by default) to allow the application to close even if this, presumably not very important, window is still opened. By default, the application stays alive as long as there are any open top level windows.

**wxTopLevelWindow::ShowFullScreen****bool ShowFullScreen(bool *show*, long *style* = *wxFULLSCREEN\_ALL*)**

Depending on the value of *show* parameter the window is either shown full screen or restored to its normal state. *style* is a bit list containing some or all of the following values, which indicate what elements of the window to hide in full-screen mode:

- `wxFULLSCREEN_NOMENUBAR`
- `wxFULLSCREEN_NOTOOLBAR`
- `wxFULLSCREEN_NOSTATUSBAR`
- `wxFULLSCREEN_NOBORDER`
- `wxFULLSCREEN_NOCAPTION`
- `wxFULLSCREEN_ALL` (all of the above)

This function has not been tested with MDI frames.

Note that showing a window full screen also actually *Show()*s (p. 1850) if it hadn't been shown yet.

**See also***wxTopLevelWindow::IsFullScreen* (p. 1715)



## **wxTopLevelWindow::UseNativeDecorations**

**void UseNativeDecorations**(bool *native* = *true*)

**This method is specific to wxUniversal port**

Use native or custom-drawn decorations for this window only. Notice that to have any effect this method must be called before really creating the window, i.e. two step creation must be used:

```
MyFrame *frame = new MyFrame;           // use default ctor
frame->UseNativeDecorations(false);      // change from default
"true"
frame->Create(parent, title, ...);       // really create the frame
```

**See also**

*UseNativeDecorationsByDefault* (p. 1721),  
*IsUsingNativeDecorations* (p. 1716)

## **wxTopLevelWindow::UseNativeDecorationsByDefault**

**void UseNativeDecorationsByDefault**(bool *native* = *true*)

**This method is specific to wxUniversal port**

Top level windows in wxUniversal port can use either system-provided window decorations (i.e. title bar and various icons, buttons and menus in it) or draw the decorations themselves. By default the system decorations are used if they are available, but this method can be called with *native* set to *false* to change this for all windows created after this point.

Also note that if `WXDECOR` environment variable is set, then custom decorations are used by default and so it may make sense to call this method with default argument if the application can't use custom decorations at all for some reason.

**See also**

*UseNativeDecorations* (p. 1721)

## **wxTreebook**

This class is an extension of the Notebook class that allows a tree structured set of pages to be shown in a control. A classic example is a netscape preferences dialog that shows a tree of preference sections on the left and select section page on the right.

To use the class simply create it and populate with pages using *InsertPage* (p. 1725), *InsertSubPage* (p. 1725), *AddPage* (p. 1723), *AddSubPage* (p. 1723).

If your tree is no more than 1 level in depth then you could simply use *AddPage* (p. 1723) and *AddSubPage* (p. 1723) to sequentially populate your tree by adding at every step a page or a subpage to the end of the tree.

**Derived from**

`wxBookCtrlBase`  
`wxControl` (p. 285)  
`wxWindow` (p. 1795)  
`wxEvtHandler` (p. 576)  
`wxObject` (p. 1148)

### Include files

`<wx/treebook.h>`

### Event handling

To process input from a treebook control, use the following event handler macros to direct input to member functions that take a `wxTreebookEvent` (p. 1726) argument.

<b>EVT_TREEBOOK_PAGE_CHANGED(id, func)</b>	The page selection was changed. Processes a <code>wxEVT_COMMAND_TREEBOOK_PAGE_CHANGED</code> event.
<b>EVT_TREEBOOK_PAGE_CHANGING(id, func)</b>	The page selection is about to be changed. Processes a <code>wxEVT_COMMAND_TREEBOOK_PAGE_CHANGING</code> event. This event can be <i>vetoed</i> (p. 1147).
<b>EVT_TREEBOOK_NODE_COLLAPSED(id, func)</b>	The page node is going to be collapsed. Processes a <code>wxEVT_COMMAND_TREEBOOK_NODE_COLLAPSED</code> event.
<b>EVT_TREEBOOK_NODE_EXPANDED(id, func)</b>	The page node is going to be expanded. Processes a <code>wxEVT_COMMAND_TREEBOOK_NODE_EXPANDED</code> event.

### See also

`wxNotebook` (p. 1135), `wxTreebookEvent` (p. 1726), `wxImageList` (p. 933), *notebook sample* (p. 2036)

### `wxTreebook::wxTreebook`

`wxTreebook()`

Default constructor.

`wxTreebook( wxWindow* parent, wxWindowID id, const wxPoint& pos =`

```
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxBK_DEFAULT, const wxString& name = wxEmptyString)
```

Creates an empty TreeBook control.

### Parameters

*parent*

The parent window. Must be non-NULL.

*id*

The window identifier.

*pos*

The window position.

*size*

The window size.

*style*

The window style. See *wxNotebook* (p. 1135).

*name*

The name of the control (used only under Motif).

### **wxTreebook::~~wxTreebook**

**~wxTreebook()**

Destroys the wxTreebook object.

Also deletes all the pages owned by the control (inserted previously into it).

### **wxTreebook::AddPage**

```
bool AddPage( wxWindow* page, const wxString& text, bool bSelect = false, int  
imageId = wxNOT_FOUND)
```

Adds a new page. The page is placed at the topmost level after all other pages. *NULL* could be specified for page to create an empty page.

### **wxTreebook::AddSubPage**

```
bool AddSubPage(wxWindow* page, const wxString& text, bool bSelect = false, int  
imageId = wxNOT_FOUND)
```

Adds a new child-page to the last top-level page. *NULL* could be specified for page to

create an empty page.

### **wxTreebook::AssignImageList**

**void AssignImageList(wxImageList\* *imageList*)**

Sets the image list for the page control and takes ownership of the list.

#### **See also**

*wxImageList* (p. 933), *SetImageList* (p. 1726)

### **wxTreebook::CollapseNode**

**bool CollapseNode(size\_t *pageId*)**

Shortcut for *ExpandNode* (p. 1724)(*pageId*, false).

### **wxTreebook::Create**

**bool Create(wxWindow\* *parent*, wxWindowID *id*, const wxPoint& *pos* = wxDefaultPosition, const wxSize& *size* = wxDefaultSize, long *style* = wxBK\_DEFAULT, const wxString& *name* = wxEmptyString)**

Creates a treebook control. See *wxTreebook::wxTreebook* (p. 1722) for the description of the parameters.

### **wxTreebook::DeleteAllPages**

**bool DeleteAllPages()**

Deletes all pages inserted into the treebook. No event is generated.

### **wxTreebook::DeletePage**

**bool DeletePage(size\_t *pagePos*)**

Deletes the page at the specified position and all its children. Could trigger page selection change in a case when selected page is removed. In that case its parent is selected (or the next page if no parent).

### **wxTreebook::ExpandNode**

**bool ExpandNode(size\_t *pageId*, bool *expand* = true)**

Expands (collapses) the *pageId* node. Returns the previous state. May generate page changing events (if selected page is under the collapsed branch, then its parent is autoselected).

### **wxTreebook::GetPageImage**

**int GetPageImage(size\_t n) const**

Returns the image index for the given page.

**wxTreebook::GetPageParent**

**int GetPageParent(size\_t page) const**

Returns the parent page of the given one or `wxNOT_FOUND` if this is a top-level page.

**wxTreebook::GetPageText**

**wxString GetPageText(size\_t n) const**

Returns the string for the given page.

**wxTreebook::GetSelection**

**int GetSelection() const**

Returns the currently selected page, or `wxNOT_FOUND` if none was selected.

Note that this method may return either the previously or newly selected page when called from the `EVT_TREEBOOK_PAGE_CHANGED` handler depending on the platform and so `wxTreebookEvent::GetSelection` should be used instead in this case.

**wxTreebook::InsertPage**

**bool InsertPage(size\_t pagePos, wxWindow\* page, const wxString& text, bool bSelect = false, int imageId = wxNOT\_FOUND)**

Inserts a new page just before the page indicated by `pagePos`. The new page is placed before `pagePos` page and on the same level. `NULL` could be specified for page to create an empty page.

**wxTreebook::InsertSubPage**

**bool InsertSubPage(size\_t pagePos, wxWindow\* page, const wxString& text, bool bSelect = false, int imageId = wxNOT\_FOUND)**

Inserts a sub page under the specified page.

`NULL` could be specified for page to create an empty page.

**wxTreebook::IsNodeExpanded**

**bool IsNodeExpanded(size\_t pageId) const**

Gets the `pagePos` page state -- whether it is expanded or collapsed

**wxTreebook::SetImageList****void SetImageList(wxImageList\* imageList)**

Sets the image list for the page control. It does not take ownership of the image list, you must delete it yourself.

**See also**

*wxImageList* (p. 933), *AssignImageList* (p. 1724)

**wxTreebook::SetPageImage****bool SetPageImage(size\_t page, int imageId)**

Sets the image index for the given page. ImageId is an index into the image list which was set with *SetImageList* (p. 1726).

**wxTreebook::SetPageText****bool SetPageText(size\_t page, const wxString& text)**

Sets the text for the given page.

**wxTreebook::SetSelection****int SetSelection(size\_t n)**

Sets the selection for the given page, returning the previous selection.

The call to this function generates the page changing events.

This function is deprecated and should not be used in new code. Please use the *ChangeSelection* (p. 1726) function instead.

**See also**

*wxTreebook::GetSelection* (p. 1725)

**wxTreebook::ChangeSelection****int ChangeSelection(size\_t page)**

Changes the selection for the given page, returning the previous selection.

The call to this function *does not* generate the page changing events. This is the only difference with *SetSelection* (p. 1726). See *this topic* (p. 2081) for more info.

**wxTreebookEvent**

This class represents the events generated by a treebook control: currently, there are four

of them. The `PAGE_CHANGING` and `PAGE_CHANGED` - have exactly the same behaviour as `wxNotebookEvent` (p. 1144).

The other two `NODE_COLLAPSED` and `NODE_EXPANDED` are triggered when page node in the tree control is collapsed/expanded. The page index could be retrieved by calling `wxTreebookEvent::GetSelection` (p. 1728).

### Derived from

`wxBookCtrlBaseEvent`  
`wxNotifyEvent` (p. 1146)  
`wxCommandEvent` (p. 250)  
`wxEvent` (p. 572)  
`wxObject` (p. 1148)

### Include files

<treebook.h>

### Event handling

To process input from a treebook control, use the following event handler macros to direct input to member functions that take a `wxTreebookEvent` (p. 1726) argument.

<b>EVT_TREEBOOK_PAGE_CHANGED(id, func)</b>	The page selection was changed. Processes a <code>wxEVT_COMMAND_TREEBOOK_PAGE_CHANGED</code> event.
<b>EVT_TREEBOOK_PAGE_CHANGING(id, func)</b>	The page selection is about to be changed. Processes a <code>wxEVT_COMMAND_TREEBOOK_PAGE_CHANGING</code> event. This event can be <i>vetoed</i> (p. 1147).
<b>EVT_TREEBOOK_NODE_COLLAPSED(id, func)</b>	The page node is going to be collapsed. Processes a <code>wxEVT_COMMAND_TREEBOOK_NODE_COLLAPSED</code> event.
<b>EVT_TREEBOOK_NODE_EXPANDED(id, func)</b>	The page node is going to be expanded. Processes a <code>wxEVT_COMMAND_TREEBOOK_NODE_EXPANDED</code> event.

### See also

`wxNotebookEvent` (p. 1144), `wxTreebook` (p. 1721)

**wxTreebookEvent::wxTreebookEvent**

**wxTreebookEvent**(**wxEvtType** *commandType* = *wxEVT\_NULL*, **int** *id* = 0, **int** *nSel* = *wxNOT\_FOUND*, **int** *nOldSel* = *wxNOT\_FOUND*)

**See also**

*wxNotebookEvent* (p. 1144)

**wxTreebookEvent::GetOldSelection**

**int** **GetOldSelection**() **const**

Returns the page that was selected before the change, *wxNOT\_FOUND* if none was selected.

**wxTreebookEvent::GetSelection**

**int** **GetSelection**() **const**

Returns the currently selected page, or *wxNOT\_FOUND* if none was selected.

**See also**

*wxNotebookEvent::GetSelection* (p. 1145)

**wxTreeCtrl**

A tree control presents information as a hierarchy, with items that may be expanded to show further items. Items in a tree control are referenced by *wxTreeItemId* handles, which may be tested for validity by calling *wxTreeItemId::IsOk* (p. 1748).

To intercept events from a tree control, use the event table macros described in *wxTreeEvent* (p. 1748).

**Derived from**

*wxControl* (p. 285)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/treectrl.h>

**Window styles**

**wxTR\_EDIT\_LABELS**      Use this style if you wish the user to be able to edit labels in the tree control.

**wxTR\_NO\_BUTTONS**      For convenience to document that no buttons are to be



drawn.

<b>wxTR_HAS_BUTTONS</b>	Use this style to show + and - buttons to the left of parent items.
<b>wxTR_NO_LINES</b>	Use this style to hide vertical level connectors.
<b>wxTR_FULL_ROW_HIGHLIGHT</b>	Use this style to have the background colour and the selection highlight extend over the entire horizontal row of the tree control window. (This flag is ignored under Windows unless you specify wxTR_NO_LINES as well.)
<b>wxTR_LINES_AT_ROOT</b>	Use this style to show lines between root nodes. Only applicable if wxTR_HIDE_ROOT is set and wxTR_NO_LINES is not set.
<b>wxTR_HIDE_ROOT</b>	Use this style to suppress the display of the root node, effectively causing the first-level nodes to appear as a series of root nodes.
<b>wxTR_ROW_LINES</b>	Use this style to draw a contrasting border between displayed rows.
<b>wxTR_HAS_VARIABLE_ROW_HEIGHT</b>	Use this style to cause row heights to be just big enough to fit the content. If not set, all rows use the largest row height. The default is that this flag is unset. Generic only.
<b>wxTR_SINGLE</b>	For convenience to document that only one item may be selected at a time. Selecting another item causes the current selection, if any, to be deselected. This is the default.
<b>wxTR_MULTIPLE</b>	Use this style to allow a range of items to be selected. If a second range is selected, the current range, if any, is deselected.
<b>wxTR_EXTENDED</b>	Use this style to allow disjoint items to be selected. (Only partially implemented; may not work in all cases.)
<b>wxTR_DEFAULT_STYLE</b>	The set of flags that are closest to the defaults for the native control for a particular toolkit.

See also *window styles overview* (p. 2089).

### Event handling

To process input from a tree control, use these event handler macros to direct input to member functions that take a *wxTreeEvent* (p. 1748) argument.

**EVT\_TREE\_BEGIN\_DRAG(id, func)** Begin dragging with the left mouse button.

**EVT\_TREE\_BEGIN\_RDRAG(id, func)** Begin dragging with the right mouse button.

<b>EVT_TREE_END_DRAG(id, func)</b>	End dragging with the left or right mouse button.
<b>EVT_TREE_BEGIN_LABEL_EDIT(id, func)</b>	Begin editing a label. This can be prevented by calling <i>Veto()</i> (p. 1147).
<b>EVT_TREE_END_LABEL_EDIT(id, func)</b>	Finish editing a label. This can be prevented by calling <i>Veto()</i> (p. 1147).
<b>EVT_TREE_DELETE_ITEM(id, func)</b>	Delete an item.
<b>EVT_TREE_GET_INFO(id, func)</b>	Request information from the application.
<b>EVT_TREE_SET_INFO(id, func)</b>	Information is being supplied.
<b>EVT_TREE_ITEM_ACTIVATED(id, func)</b>	The item has been activated, i.e. chosen by double clicking it with mouse or from keyboard
<b>EVT_TREE_ITEM_COLLAPSED(id, func)</b>	The item has been collapsed.
<b>EVT_TREE_ITEM_COLLAPSING(id, func)</b>	The item is being collapsed. This can be prevented by calling <i>Veto()</i> (p. 1147).
<b>EVT_TREE_ITEM_EXPANDED(id, func)</b>	The item has been expanded.
<b>EVT_TREE_ITEM_EXPANDING(id, func)</b>	The item is being expanded. This can be prevented by calling <i>Veto()</i> (p. 1147).
<b>EVT_TREE_ITEM_RIGHT_CLICK(id, func)</b>	The user has clicked the item with the right mouse button.
<b>EVT_TREE_ITEM_MIDDLE_CLICK(id, func)</b>	The user has clicked the item with the middle mouse button.
<b>EVT_TREE_SEL_CHANGED(id, func)</b>	Selection has changed.
<b>EVT_TREE_SEL_CHANGING(id, func)</b>	Selection is changing. This can be prevented by calling <i>Veto()</i> (p. 1147).
<b>EVT_TREE_KEY_DOWN(id, func)</b>	A key has been pressed.
<b>EVT_TREE_ITEM_GETTOOLTIP(id, func)</b>	The opportunity to set the item tooltip is being given to the application (call <code>wxTreeEvent::SetToolTip</code> ). Windows only.
<b>EVT_TREE_ITEM_MENU(id, func)</b>	The context menu for the selected item has been requested, either by a right click or by using the menu key.
<b>EVT_TREE_STATE_IMAGE_CLICK(id, func)</b>	The state image has been clicked. Windows only.

**See also**

*wxTreeItemData* (p. 1751), *wxTreeCtrl* overview (p. 2125), *wxListBox* (p. 974), *wxListCtrl* (p. 980), *wxImageList* (p. 933), *wxTreeEvent* (p. 1748)

## Win32 notes

`wxTreeCtrl` class uses the standard common treeview control under Win32 implemented in the system library `comctl32.dll`. Some versions of this library are known to have bugs with handling the tree control colours: the usual symptom is that the expanded items leave black (or otherwise incorrectly coloured) background behind them, especially for the controls using non-default background colour. The recommended solution is to upgrade the `comctl32.dll` to a newer version:

see <http://www.microsoft.com/downloads/release.asp?ReleaseID=11916>

(<http://www.microsoft.com/downloads/release.asp?ReleaseID=11916>).

## `wxTreeCtrl::wxTreeCtrl`

### `wxTreeCtrl()`

Default constructor.

```
wxTreeCtrl(wxWindow* parent, wxWindowID id, const wxPoint& pos =  
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxTR_HAS_BUTTONS, const wxValidator& validator = wxDefaultValidator, const  
wxString& name = "treeCtrl")
```

Constructor, creating and showing a tree control.

### Parameters

*parent*

Parent window. Must not be `NULL`.

*id*

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

*pos*

Window position.

*size*

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

*style*

Window style. See `wxTreeCtrl` (p. 1728).

*validator*

Window validator.

*name*

Window name.

**See also**

*wxTreeCtrl::Create* (p. 1733), *wxValidator* (p. 1767)

**wxTreeCtrl::~~wxTreeCtrl**

**void ~wxTreeCtrl()**

Destructor, destroying the tree control.

**wxTreeCtrl::AddRoot**

**wxTreeItemId AddRoot(const wxString& text, int image = -1, int sellImage = -1, wxTreeItemData\* data = NULL)**

Adds the root node to the tree, returning the new item.

The *image* and *sellImage* parameters are an index within the normal image list specifying the image to use for unselected and selected items, respectively. If *image* > -1 and *sellImage* is -1, the same image is used for both selected and unselected items.

**wxTreeCtrl::AppendItem**

**wxTreeItemId AppendItem(const wxTreeItemId& parent, const wxString& text, int image = -1, int sellImage = -1, wxTreeItemData\* data = NULL)**

Appends an item to the end of the branch identified by *parent*, return a new item id.

The *image* and *sellImage* parameters are an index within the normal image list specifying the image to use for unselected and selected items, respectively. If *image* > -1 and *sellImage* is -1, the same image is used for both selected and unselected items.

**wxTreeCtrl::AssignButtonsImageList**

**void AssignButtonsImageList(wxImageList\* imageList)**

Sets the buttons image list. The button images assigned with this method will be automatically deleted by *wxTreeCtrl* as appropriate (i.e. it takes ownership of the list).

Setting or assigning the button image list enables the display of image buttons. Once enabled, the only way to disable the display of button images is to set the button image list to `NULL`.

This function is only available in the generic version.

See also *SetButtonsImageList* (p. 1743).

**wxTreeCtrl::AssignImageList**

**void AssignImageList(wxImageList\* imageList)**

Sets the normal image list. Image list assigned with this method will be automatically deleted by wxTreeCtrl as appropriate (i.e. it takes ownership of the list).

See also *SetImageList* (p. 1744).

**wxTreeCtrl::AssignStateImageList**

**void AssignStateImageList(wxImageList\* imageList)**

Sets the state image list. Image list assigned with this method will be automatically deleted by wxTreeCtrl as appropriate (i.e. it takes ownership of the list).

See also *SetStateImageList* (p. 1746).

**wxTreeCtrl::Collapse**

**void Collapse(const wxTreeItemId& item)**

Collapses the given item.

**wxTreeCtrl::CollapseAll**

**void CollapseAll()**

Collapses the root item.

**See also**

*ExpandAll* (p. 1735)

**wxTreeCtrl::CollapseAllChildren**

**void CollapseAllChildren(const wxTreeItemId& item)**

Collapses this item and all of its children, recursively.

**See also**

*ExpandAllChildren* (p. 1735)

**wxTreeCtrl::CollapseAndReset**

**void CollapseAndReset(const wxTreeItemId& item)**

Collapses the given item and removes all children.

**wxTreeCtrl::Create**

**bool wxTreeCtrl(wxWindow\* parent, wxWindowID id, const wxPoint& pos =**

```
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =  
wxTR_HAS_BUTTONS, const wxValidator& validator = wxDefaultValidator, const  
wxString& name = "treeCtrl")
```

Creates the tree control. See *wxTreeCtrl::wxTreeCtrl* (p. 1731) for further details.

### **wxTreeCtrl::Delete**

```
void Delete(const wxTreeItemId& item)
```

Deletes the specified item. A `EVT_TREE_DELETE_ITEM` event will be generated.

This function may cause a subsequent call to `GetNextChild` to fail.

### **wxTreeCtrl::DeleteAllItems**

```
void DeleteAllItems()
```

Deletes all items in the control. Note that this may not generate `EVT_TREE_DELETE_ITEM` events under some Windows versions although normally such event is generated for each removed item.

### **wxTreeCtrl::DeleteChildren**

```
void DeleteChildren(const wxTreeItemId& item)
```

Deletes all children of the given item (but not the item itself). Note that this will **not** generate any events unlike *Delete* (p. 1734) method.

If you have called *wxTreeCtrl::SetItemHasChildren* (p. 1745), you may need to call it again since *DeleteChildren* does not automatically clear the setting.

### **wxTreeCtrl::EditLabel**

```
void EditLabel(const wxTreeItemId& item)
```

Starts editing the label of the given item. This function generates a `EVT_TREE_BEGIN_LABEL_EDIT` event which can be vetoed so that no text control will appear for in-place editing.

If the user changed the label (i.e. s/he does not press ESC or leave the text control without changes, a `EVT_TREE_END_LABEL_EDIT` event will be sent which can be vetoed as well.

#### **See also**

*wxTreeCtrl::EndEditLabel* (p. 1734), *wxTreeEvent* (p. 1748)

### **wxTreeCtrl::EndEditLabel**

```
void EndEditLabel(bool cancelEdit)
```

Ends label editing. If *cancelEdit* is `true`, the edit will be cancelled.

This function is currently supported under Windows only.

**See also**

*wxTreeCtrl::EditLabel* (p. 1734)

**wxTreeCtrl::EnsureVisible**

**void EnsureVisible(const wxTreeItemId& item)**

Scrolls and/or expands items to ensure that the given item is visible.

**wxTreeCtrl::Expand**

**void Expand(const wxTreeItemId& item)**

Expands the given item.

**wxTreeCtrl::ExpandAll**

**void ExpandAll()**

Expands all items in the tree.

**wxTreeCtrl::ExpandAllChildren**

**void ExpandAllChildren(const wxTreeItemId& item)**

Expands the given item and all its children recursively.

**wxTreeCtrl::GetBoundingRect**

**bool GetBoundingRect(const wxTreeItemId& item, wxRect& rect, bool textOnly = false) const**

Retrieves the rectangle bounding the *item*. If *textOnly* is `true`, only the rectangle around the item's label will be returned, otherwise the item's image is also taken into account.

The return value is `true` if the rectangle was successfully retrieved or `false` if it was not (in this case *rect* is not changed) -- for example, if the item is currently invisible.

Notice that the rectangle coordinates are logical, not physical ones. So, for example, the x coordinate may be negative if the tree has a horizontal scrollbar and its position is not 0.

**wxPython note:** The wxPython version of this method requires only the *item* and *textOnly* parameters. The return value is either `awxRect` object or `None`.

**wxPerl note:** In wxPerl this method only takes the parameters *item* and *textOnly*, and returns a `Wx::Rect` ( or `undef` ).

**wxTreeCtrl::GetButtonsImageList****wxImageList\* GetButtonsImageList() const**

Returns the buttons image list (from which application-defined button images are taken).

This function is only available in the generic version.

**wxTreeCtrl::GetChildrenCount****unsigned int GetChildrenCount(const wxTreeItemId& item, bool recursively = true) const**

Returns the number of items in the branch. If *recursively* is `true`, returns the total number of descendants, otherwise only one level of children is counted.

**wxTreeCtrl::GetCount****unsigned int GetCount() const**

Returns the number of items in the control.

**wxTreeCtrl::GetEditControl****wxTextCtrl \* GetEditControl() const**

Returns the edit control being currently used to edit a label. Returns `NULL` if no label is being edited.

**NB:** It is currently only implemented for wxMSW.

**wxTreeCtrl::GetFirstChild****wxTreeItemId GetFirstChild(const wxTreeItemId& item, wxTreeItemIdValue & cookie) const**

Returns the first child; call *wxTreeCtrl::GetNextChild* (p. 1739) for the next child.

For this enumeration function you must pass in a 'cookie' parameter which is opaque for the application but is necessary for the library to make these functions reentrant (i.e. allow more than one enumeration on one and the same object simultaneously). The cookie passed to *GetFirstChild* and *GetNextChild* should be the same variable.

Returns an invalid tree item (i.e. *IsOk()* returns `false`) if there are no further children.

**See also**

*wxTreeCtrl::GetNextChild* (p. 1739), *wxTreeCtrl::GetNextSibling* (p. 1739)

**wxPython note:** In wxPython the returned *wxTreeItemId* and the new cookie value are both returned as a tuple containing the two values.



**wxPerl note:** In wxPerl this method only takes the `item` parameter, and returns a 2-element list ( `item`, `cookie` ).

### **wxTreeCtrl::GetFirstVisibleItem**

**wxTreeItemId GetFirstVisibleItem() const**

Returns the first visible item.

### **wxTreeCtrl::GetImageList**

**wxImageList\* GetImageList() const**

Returns the normal image list.

### **wxTreeCtrl::GetIndent**

**int GetIndent() const**

Returns the current tree control indentation.

### **wxTreeCtrl::GetItemBackgroundColour**

**wxColour GetItemBackgroundColour(const wxTreeItemId& *item*) const**

Returns the background colour of the item.

### **wxTreeCtrl::GetItemData**

**wxTreeItemData\* GetItemData(const wxTreeItemId& *item*) const**

Returns the tree item data associated with the item.

### **See also**

*wxTreeItemData* (p. 1751)

**wxPython note:** wxPython provides the following shortcut method:

**GetPyData(item)**

Returns the Python Object associated with the `wxTreeItemData` for the given item Id.

**wxPerl note:** wxPerl provides the following shortcut

method: **GetPIData( item )**

Returns the Perl data associated with the `Wx::TreeItemData`. It is just the same as

```
tree->GetItemData(item)->GetData(
).
```

### **wxTreeCtrl::GetItemFont**

**wxFont GetItemFont(const wxTreeItemId& item) const**

Returns the font of the item label.

### **wxTreeCtrl::GetItemImage**

**int GetItemImage(const wxTreeItemId& item, wxTreeItemIcon which = wxTreeItemIcon\_Normal) const**

Gets the specified item image. The value of *which* may be:

- `_Normal` to get the normal item image
- `_Selected` to get the selected item image (i.e. the image which is shown when the item is currently selected)
- `_Expanded` to get the expanded image (this only makes sense for items which have children - then this image is shown when the item is expanded and the normal image is shown when it is collapsed)
- `_SelectedExpanded` to get the selected expanded image (which is shown when an expanded item is currently selected)

### **wxTreeCtrl::GetItemText**

**wxString GetItemText(const wxTreeItemId& item) const**

Returns the item label.

### **wxTreeCtrl::GetItemTextColour**

**wxColour GetItemTextColour(const wxTreeItemId& item) const**

Returns the colour of the item label.

### **wxTreeCtrl::GetLastChild**

**wxTreeItemId GetLastChild(const wxTreeItemId& item) const**

Returns the last child of the item (or an invalid tree item if this item has no children).

### **See also**

*GetFirstChild* (p. 1736), *wxTreeCtrl::GetNextSibling* (p. 1739), *GetLastChild* (p. 1738)

**wxTreeCtrl::GetNextChild**

**wxTreeItemId GetNextChild(const wxTreeItemId& item, wxTreeItemIdValue & cookie) const**

Returns the next child; call *wxTreeCtrl::GetFirstChild* (p. 1736) for the first child.

For this enumeration function you must pass in a 'cookie' parameter which is opaque for the application but is necessary for the library to make these functions reentrant (i.e. allow more than one enumeration on one and the same object simultaneously). The cookie passed to *GetFirstChild* and *GetNextChild* should be the same.

Returns an invalid tree item if there are no further children.

**See also**

*wxTreeCtrl::GetFirstChild* (p. 1736)

**wxPython note:** In wxPython the returned *wxTreeItemId* and the new cookie value are both returned as a tuple containing the two values.

**wxPerl note:** In wxPerl this method returns a 2-element list ( *item*, *cookie* ), instead of modifying its parameters.

**wxTreeCtrl::GetNextSibling**

**wxTreeItemId GetNextSibling(const wxTreeItemId& item) const**

Returns the next sibling of the specified item; call *wxTreeCtrl::GetPrevSibling* (p. 1739) for the previous sibling.

Returns an invalid tree item if there are no further siblings.

**See also**

*wxTreeCtrl::GetPrevSibling* (p. 1739)

**wxTreeCtrl::GetNextVisible**

**wxTreeItemId GetNextVisible(const wxTreeItemId& item) const**

Returns the next visible item.

**wxTreeCtrl::GetItemParent**

**wxTreeItemId GetItemParent(const wxTreeItemId& item) const**

Returns the item's parent.

**wxTreeCtrl::GetPrevSibling**

**wxTreeItemId GetPrevSibling(const wxTreeItemId& item) const**

Returns the previous sibling of the specified item; call *wxTreeCtrl::GetNextSibling* (p. 1739) for the next sibling.

Returns an invalid tree item if there are no further children.

**See also**

*wxTreeCtrl::GetNextSibling* (p. 1739)

**wxTreeCtrl::GetPrevVisible**

**wxTreeItemId GetPrevVisible(const wxTreeItemId& item) const**

Returns the previous visible item.

**wxTreeCtrl::GetQuickBestSize**

**bool GetQuickBestSize() const**

Returns true if the control will use a quick calculation for the best size, looking only at the first and last items. The default is false.

**See also**

*wxTreeCtrl::SetQuickBestSize* (p. 1746)

**wxTreeCtrl::GetRootItem**

**wxTreeItemId GetRootItem() const**

Returns the root item for the tree control.

**wxTreeCtrl::GetItemSelectedImage**

**int GetItemSelectedImage(const wxTreeItemId& item) const**

Gets the selected item image (this function is obsolete, use *GetItemImage(item, wxTreeItemIcon\_Selected)* instead).

**wxTreeCtrl::GetSelection**

**wxTreeItemId GetSelection() const**

Returns the selection, or an invalid item if there is no selection. This function only works with the controls without *wxTR\_MULTIPLE* style, use *GetSelections* (p. 1740) for the controls which do have this style.

**wxTreeCtrl::GetSelections**

**unsigned int GetSelections(wxArrayTreeItemIds& selection) const**

Fills the array of tree items passed in with the currently selected items. This function can be called only if the control has the `wxTR_MULTIPLE` style.

Returns the number of selected items.

**wxPython note:** The wxPython version of this method accepts no parameters and returns a Python list of `wxTreeItemIds`.

**wxPerl note:** In wxPerl this method takes no parameters and returns a list of `Wx::TreeItemIds`.

### **wxTreeCtrl::GetStateImageList**

**wxImageList\* GetStateImageList() const**

Returns the state image list (from which application-defined state images are taken).

### **wxTreeCtrl::HitTest**

**wxTreeItemId HitTest(const wxPoint& point, int& flags) const**

Calculates which (if any) item is under the given point, returning the tree item id at this point plus extra information *flags*. *flags* is a bitlist of the following:

`wxTREE_HITTEST_ABOVE` Above the client area.

`wxTREE_HITTEST_BELOW` Below the client area.

`wxTREE_HITTEST_NOWHERE` In the client area but below the last item.

`wxTREE_HITTEST_ONITEMBUTTON` On the button associated with an item.

`wxTREE_HITTEST_ONITEMICON` On the bitmap associated with an item.

`wxTREE_HITTEST_ONITEMINDENT` In the indentation associated with an item.

`wxTREE_HITTEST_ONITEMLABEL` On the label (string) associated with an item.

`wxTREE_HITTEST_ONITEMRIGHT` In the area to the right of an item.

`wxTREE_HITTEST_ONITEMSTATEICON` On the state icon for a tree view item that is in a user-defined state.

`wxTREE_HITTEST_TOLEFT` To the right of the client area.

`wxTREE_HITTEST_TORIGHT` To the left of the client area.

**wxPython note:** in wxPython both the `wxTreeItemId` and the *flags* are returned as a tuple.

**wxPerl note:** In wxPerl this method only takes the `point` parameter and returns a 2-element list ( `item`, `flags` ).

### **wxTreeCtrl::InsertItem**

**wxTreeItemId InsertItem(const wxTreeItemId& parent, const wxTreeItemId& previous, const wxString& text, int image = -1, int selImage = -1, wxTreeItemData\* data = NULL)**

**wxTreeItemId InsertItem(const wxTreeItemId& parent, size\_t before, const wxString& text, int image = -1, int selImage = -1, wxTreeItemData\* data = NULL)**

Inserts an item after a given one (*previous*) or before one identified by its position (*before*). *before* must be less than the number of children.

The *image* and *selImage* parameters are an index within the normal image list specifying the image to use for unselected and selected items, respectively. If *image* > -1 and *selImage* is -1, the same image is used for both selected and unselected items.

**wxPython note:** The second form of this method is called `InsertItemBefore` in wxPython.

### **wxTreeCtrl::IsBold**

**bool IsBold(const wxTreeItemId& item) const**

Returns `true` if the given item is in bold state.

See also: *SetItemBold* (p. 1744)

### **wxTreeCtrl::IsEmpty**

**bool IsEmpty() const**

Returns `true` if the control is empty (i.e. has no items, even no root one).

### **wxTreeCtrl::IsExpanded**

**bool IsExpanded(const wxTreeItemId& item) const**

Returns `true` if the item is expanded (only makes sense if it has children).

### **wxTreeCtrl::IsSelected**

**bool IsSelected(const wxTreeItemId& item) const**

Returns `true` if the item is selected.

### **wxTreeCtrl::IsVisible**

**bool IsVisible(const wxTreeItemId& item) const**

Returns `true` if the item is visible (it might be outside the view, or not expanded).

### **wxTreeCtrl::ItemHasChildren**

**bool ItemHasChildren(const wxTreeItemId& item) const**

Returns `true` if the item has children.

### **wxTreeCtrl::OnCompareItems**

**int OnCompareItems(const wxTreeItemId& item1, const wxTreeItemId& item2)**

Override this function in the derived class to change the sort order of the items in the tree control. The function should return a negative, zero or positive value if the first item is less than, equal to or greater than the second one.

Please note that you **must** use wxRTTI macros `DECLARE_DYNAMIC_CLASS` (p. 1966) and `IMPLEMENT_DYNAMIC_CLASS` (p. 1967) if you override this function because otherwise the base class considers that it is not overridden and uses the default comparison, i.e. sorts the items alphabetically, which allows it optimize away the calls to the virtual function completely.

See also: *SortChildren* (p. 1746)

### **wxTreeCtrl::PrependItem**

**wxTreeItemId PrependItem(const wxTreeItemId& parent, const wxString& text, int image = -1, int selImage = -1, wxTreeItemData\* data = NULL)**

Appends an item as the first child of *parent*, return a new item id.

The *image* and *selImage* parameters are an index within the normal image list specifying the image to use for unselected and selected items, respectively. If *image* > -1 and *selImage* is -1, the same image is used for both selected and unselected items.

### **wxTreeCtrl::ScrollTo**

**void ScrollTo(const wxTreeItemId& item)**

Scrolls the specified item into view.

### **wxTreeCtrl::SelectItem**

**void SelectItem(const wxTreeItemId& item, bool select = true)**

Selects the given item. In multiple selection controls, can be also used to deselect a currently selected item if the value of *select* is false.

### **wxTreeCtrl::SetButtonsImageList**

**void SetButtonsImageList(wxImageList\* imageList)**

Sets the buttons image list (from which application-defined button images are taken). The button images assigned with this method will **not** be deleted by wxTreeCtrl's destructor, you must delete it yourself.

Setting or assigning the button image list enables the display of image buttons. Once enabled, the only way to disable the display of button images is to set the button image list to `NULL`.

This function is only available in the generic version.

See also *AssignButtonsImageList* (p. 1732).

### **wxTreeCtrl::SetIndent**

**void SetIndent(int indent)**

Sets the indentation for the tree control.

### **wxTreeCtrl::SetImageList**

**void SetImageList(wxImageList\* imageList)**

Sets the normal image list. Image list assigned with this method will **not** be deleted by wxTreeCtrl's destructor, you must delete it yourself.

See also *AssignImageList* (p. 1732).

### **wxTreeCtrl::SetItemBackgroundColour**

**void SetItemBackgroundColour(const wxTreeItemId& item, const wxColour& col)**

Sets the colour of the item's background.

### **wxTreeCtrl::SetItemBold**

**void SetItemBold(const wxTreeItemId& item, bool bold = true)**

Makes item appear in bold font if *bold* parameter is `true` or resets it to the normal state.

See also: *IsBold* (p. 1742)

### **wxTreeCtrl::SetItemData**

**void SetItemData(const wxTreeItemId& item, wxTreeItemData\* data)**

Sets the item client data.

**wxPython note:** wxPython provides the following shortcut method:

<b>SetPyData(item, obj)</b>	Associate the given Python Object with the wxTreeItemData for the given item Id.
-----------------------------	--

<b>wxPerl note:</b> wxPerl provides the following shortcut method:	<b>SetPIData( item, data )</b>	Sets the Perl data associated with
--	--------------------------------	------------------------------------



the `Wx::TreeItemData`. It is just the same as `tree->GetItemData(item)->SetData(data)`.

### **`wxTreeCtrl::SetItemDropHighlight`**

**`void SetItemDropHighlight(const wxTreeItemId& item, bool highlight = true)`**

Gives the item the visual feedback for Drag'n'Drop actions, which is useful if something is dragged from the outside onto the tree control (as opposed to a DnD operation within the tree control, which already is implemented internally).

### **`wxTreeCtrl::SetItemFont`**

**`void SetItemFont(const wxTreeItemId& item, const wxFont& font)`**

Sets the item's font. All items in the tree should have the same height to avoid text clipping, so the fonts height should be the same for all of them, although font attributes may vary.

#### **See also**

*SetItemBold* (p. 1744)

### **`wxTreeCtrl::SetItemHasChildren`**

**`void SetItemHasChildren(const wxTreeItemId& item, bool hasChildren = true)`**

Force appearance of the button next to the item. This is useful to allow the user to expand the items which don't have any children now, but instead adding them only when needed, thus minimizing memory usage and loading time.

### **`wxTreeCtrl::SetItemImage`**

**`void SetItemImage(const wxTreeItemId& item, int image, wxTreeItemIcon which = wxTreeItemIcon_Normal)`**

Sets the specified item image. See *GetItemImage* (p. 1738) for the description of the *which* parameter.

### **`wxTreeCtrl::SetItemSelectedImage`**

**`void SetItemSelectedImage(const wxTreeItemId& item, int selImage)`**

Sets the selected item image (this function is obsolete, use `SetItemImage(item, wxTreeItemIcon_Selected)` instead).

### **`wxTreeCtrl::SetItemText`**

**`void SetItemText(const wxTreeItemId& item, const wxString& text)`**

Sets the item label.

### **wxTreeCtrl::SetItemTextColour**

**void SetItemTextColour(const wxTreeItemId& *item*, const wxColour& *col*)**

Sets the colour of the item's text.

### **wxTreeCtrl::SetQuickBestSize**

**void SetQuickBestSize(bool *quickBestSize*)**

If true is passed, specifies that the control will use a quick calculation for the best size, looking only at the first and last items. Otherwise, it will look at all items. The default is false.

#### **See also**

*wxTreeCtrl::GetQuickBestSize* (p. 1740)

### **wxTreeCtrl::SetStateImageList**

**void SetStateImageList(wxImageList\* *imageList*)**

Sets the state image list (from which application-defined state images are taken). Image list assigned with this method will **not** be deleted by wxTreeCtrl's destructor, you must delete it yourself.

See also *AssignStateImageList* (p. 1733).

### **wxTreeCtrl::SetWindowStyle**

**void SetWindowStyle(long *styles*)**

Sets the mode flags associated with the display of the tree control. The new mode takes effect immediately. (Generic only; MSW ignores changes.)

### **wxTreeCtrl::SortChildren**

**void SortChildren(const wxTreeItemId& *item*)**

Sorts the children of the given item using *OnCompareItems* (p. 1743) method of wxTreeCtrl. You should override that method to change the sort order (the default is ascending case-sensitive alphabetical order).

#### **See also**

*wxTreeItemData* (p. 1751), *OnCompareItems* (p. 1743)

### **wxTreeCtrl::Toggle**

**void Toggle(const wxTreeItemId& item)**

Toggles the given item between collapsed and expanded states.

### **wxTreeCtrl::ToggleItemSelection**

**void ToggleItemSelection(const wxTreeItemId& item)**

Toggles the given item between selected and unselected states. For multiselection controls only.

### **wxTreeCtrl::Unselect**

**void Unselect()**

Removes the selection from the currently selected item (if any).

### **wxTreeCtrl::UnselectAll**

**void UnselectAll()**

This function either behaves the same as *Unselect* (p. 1747) if the control doesn't have `wxTR_MULTIPLE` style, or removes the selection from all items if it does have this style.

### **wxTreeCtrl::UnselectItem**

**void UnselectItem(const wxTreeItemId& item)**

Unselects the given item. This works in multiselection controls only.

## **wxTreeItemId**

An opaque reference to a tree item.

### **Derived from**

None

### **Include files**

<wx/treebase.h>

### **See also**

*wxTreeCtrl* (p. 1728), *wxTreeItemData* (p. 1751),  
*wxTreeCtrl* overview (p. 2125)

### **wxTreeItemId::wxTreeItemId**

**wxTreeItemId()**

Default constructor. `wxTreeItemIds` are not meant to be constructed explicitly by the user; they are returned by the `wxTreeCtrl` (p. 1728) functions instead.

**wxTreeItemId::IsOk****bool IsOk() const**

Returns `true` if this instance is referencing a valid tree item.

**Operators****void operator !() const**

Synonym for `IsOk` (p. 1748)

**bool operator ==(const wxTreeItemId& item) const****bool operator !=(const wxTreeItemId& item) const**

Operators for comparison between `wxTreeItemId` (p. 1747) objects.

**wxTreeEvent**

A tree event holds information about events associated with `wxTreeCtrl` objects.

**Derived from**

`wxNotifyEvent` (p. 1146)  
`wxCommandEvent` (p. 250)  
`wxEvent` (p. 572)  
`wxObject` (p. 1148)

**Include files**

<wx/treectrl.h>

**Event table macros**

To process input from a tree control, use these event handler macros to direct input to member functions that take a `wxTreeEvent` argument.

**EVT\_TREE\_BEGIN\_DRAG(id, func)**

The user has started dragging an item with the left mouse button. The event handler must call **wxTreeEvent::Allow()** for the drag operation to continue.

**EVT\_TREE\_BEGIN\_RDRAG(id, func)**

The user has started dragging an item with the right mouse button. The event handler must call

	<b>wxTreeEvent::Allow()</b> for the drag operation to continue.
<b>EVT_TREE_BEGIN_LABEL_EDIT(id, func)</b>	Begin editing a label. This can be prevented by calling <i>Veto()</i> (p. 1147).
<b>EVT_TREE_END_DRAG(id, func)</b>	The user has released the mouse after dragging an item.
<b>EVT_TREE_END_LABEL_EDIT(id, func)</b>	The user has finished editing a label. This can be prevented by calling <i>Veto()</i> (p. 1147).
<b>EVT_TREE_DELETE_ITEM(id, func)</b>	A tree item has been deleted.
<b>EVT_TREE_ITEM_ACTIVATED(id, func)</b>	An item has been activated (e.g. double clicked).
<b>EVT_TREE_ITEM_COLLAPSED(id, func)</b>	The item has been collapsed.
<b>EVT_TREE_ITEM_COLLAPSING(id, func)</b>	The item is being collapsed. This can be prevented by calling <i>Veto()</i> (p. 1147).
<b>EVT_TREE_ITEM_EXPANDED(id, func)</b>	The item has been expanded.
<b>EVT_TREE_ITEM_EXPANDING(id, func)</b>	The item is being expanded. This can be prevented by calling <i>Veto()</i> (p. 1147).
<b>EVT_TREE_ITEM_RIGHT_CLICK(id, func)</b>	The user has clicked the item with the right mouse button.
<b>EVT_TREE_ITEM_MIDDLE_CLICK(id, func)</b>	The user has clicked the item with the middle mouse button.
<b>EVT_TREE_KEY_DOWN(id, func)</b>	A key has been pressed.
<b>EVT_TREE_SEL_CHANGED(id, func)</b>	Selection has changed.
<b>EVT_TREE_SEL_CHANGING(id, func)</b>	Selection is changing. This can be prevented by calling <i>Veto()</i> (p. 1147).
<b>EVT_TREE_KEY_DOWN(id, func)</b>	A key has been pressed.
<b>EVT_TREE_ITEM_GETTOOLTIP(id, func)</b>	The opportunity to set the item tooltip is being given to the application (call <i>wxTreeEvent::SetToolTip()</i> ). Windows only.
<b>EVT_TREE_ITEM_MENU(id, func)</b>	The context menu for the selected

item has been requested, either by a right click or by using the menu key.

**EVT\_TREE\_STATE\_IMAGE\_CLICK(id, func)**

The state image has been clicked. Windows only.

**See also**

*wxTreeCtrl* (p. 1728)

**wxTreeEvent::wxTreeEvent**

**wxTreeEvent(wxEventType commandType, wxTreeCtrl \*tree, const wxTreeItemId& item = wxTreeItemId())**

Constructor, used by wxWidgets itself only.

**wxTreeEvent::GetKeyCode**

**int GetKeyCode() const**

Returns the key code if the event is a key event. Use *GetKeyEvent* (p. 1750) to get the values of the modifier keys for this event (i.e. Shift or Ctrl).

**wxTreeEvent::GetItem**

**wxTreeItemId GetItem() const**

Returns the item (valid for all events).

**wxTreeEvent::GetKeyEvent**

**const wxKeyEvent& GetKeyEvent() const**

Returns the key event for `EVT_TREE_KEY_DOWN` events.

**wxTreeEvent::GetLabel**

**const wxString& GetLabel() const**

Returns the label if the event is a begin or end edit label event.

**wxTreeEvent::GetOldItem**

**wxTreeItemId GetOldItem() const**

Returns the old item index (valid for `EVT_TREE_ITEM_CHANGING` and `CHANGED` events)

**wxTreeEvent::GetPoint****wxPoint GetPoint() const**

Returns the position of the mouse pointer if the event is a drag or menu-context event. In both cases the position is in client coordinates - i.e. relative to the wxTreeCtrl window (so that you can pass it directly to e.g. *wxWindow::PopupMenu* (p. 1828)).

**wxTreeEvent::IsEditCancelled****bool IsEditCancelled() const**

Returns true if the label edit was cancelled. This should be called from within an EVT\_TREE\_END\_LABEL\_EDIT handler.

**wxTreeEvent::SetToolTip****void SetToolTip(const wxString& tooltip)**

Set the tooltip for the item (valid for EVT\_TREE\_ITEM\_GETTOOLTIP events). Windows only.

**wxTreeItemData**

wxTreeItemData is some (arbitrary) user class associated with some item. The main advantage of having this class is that wxTreeItemData objects are destroyed automatically by the tree and, as this class has virtual destructor, it means that the memory and any other resources associated with a tree item will be automatically freed when it is deleted. Note that we don't use wxObject as the base class for wxTreeItemData because the size of this class is critical: in many applications, each tree leaf will have wxTreeItemData associated with it and the number of leaves may be quite big.

Also please note that because the objects of this class are deleted by the tree using the operator `delete`, they must always be allocated on the heap using `new`.

**Derived from**

*wxClientData* (p. 193)

**Include files**

<wx/treectrl.h>

**See also**

*wxTreeCtrl* (p. 1728)

**wxTreeItemData::wxTreeItemData**

**wxTreeItemData()**

Default constructor.

**wxPython note:** The wxPython version of this constructor optionally accepts any Python object as a parameter. This object is then associated with the tree item using the wxTreeItemData as a container.

In addition, the following methods are added in wxPython for accessing the object:

<b>GetData()</b>	Returns a reference to the Python Object
<b>SetData(obj)</b>	Associates a new Python Object with the wxTreeItemData

**wxPerl note:** In wxPerl the constructor accepts as parameter an optional scalar, and stores it as client data. You may retrieve this data by calling **GetData()**, and set it by calling **SetData( data )**.

**wxTreeItemData::~wxTreeItemData****void ~wxTreeItemData()**

Virtual destructor.

**wxTreeItemData::GetId****const wxTreeItemId& GetId()**

Returns the item associated with this node.

**wxTreeItemData::SetId****void SetId(const wxTreeItemId& id)**

Sets the item associated with this node.

**wxUpdateUIEvent**

This class is used for pseudo-events which are called by wxWidgets to give an application the chance to update various user interface elements.

**Derived from**

*wxCommandEvent* (p. 250)

*wxEvent* (p. 572)

*wxObject* (p. 1148)



**Include files**

<wx/event.h>

**Event table macros**

To process an update event, use these event handler macros to direct input to member functions that take a `wxUpdateUIEvent` argument.

**EVT\_UPDATE\_UI(id, func)** Process a `wxEVT_UPDATE_UI` event for the command with the given id.

**EVT\_UPDATE\_UI\_RANGE(id1, id2, func)** Process a `wxEVT_UPDATE_UI` event for any command with id included in the given range.

**Remarks**

Without update UI events, an application has to work hard to check/uncheck, enable/disable, show/hide, and set the text for elements such as menu items and toolbar buttons. The code for doing this has to be mixed up with the code that is invoked when an action is invoked for a menu item or button.

With update UI events, you define an event handler to look at the state of the application and change UI elements accordingly. `wxWidgets` will call your member functions in idle time, so you don't have to worry where to call this code. In addition to being a clearer and more declarative method, it also means you don't have to worry whether you're updating a toolbar or menubar identifier. The same handler can update a menu item and toolbar button, if the identifier is the same.

Instead of directly manipulating the menu or button, you call functions in the event object, such as `wxUpdateUIEvent::Check` (p. 1754). `wxWidgets` will determine whether such a call has been made, and which UI element to update.

These events will work for popup menus as well as menubars. Just before a menu is popped up, `wxMenu::UpdateUI` (p. 1087) is called to process any UI events for the window that owns the menu.

If you find that the overhead of UI update processing is affecting your application, you can do one or both of the following:

1. Call `wxUpdateUIEvent::SetMode` (p. 1756) with a value of `wxUPDATE_UI_PROCESS_SPECIFIED`, and set the extra style `wxWS_EX_PROCESS_UPDATE_EVENTS` for every window that should receive update events. No other windows will receive update events.
2. Call `wxUpdateUIEvent::SetUpdateInterval` (p. 1757) with a millisecond value to set the delay between updates. You may need to call `wxWindow::UpdateWindowUI` (p. 1852) at critical points, for example when a dialog is about to be shown, in case the user sees a slight delay before windows are updated.

Note that although events are sent in idle time, defining a `wxIdleEvent` handler for a window does not affect this because the events are sent from `wxWindow::OnInternalIdle`

(p. 1827) which is **always** called in idle time.

`wxWidgets` tries to optimize update events on some platforms. On Windows and GTK+, events for menubar items are only sent when the menu is about to be shown, and not in idle time.

#### See also

*Event handling overview* (p. 2077)

### **wxUpdateUIEvent::wxUpdateUIEvent**

**wxUpdateUIEvent**(`wxWindowID` *commandId* = 0)

Constructor.

### **wxUpdateUIEvent::CanUpdate**

**static bool** **CanUpdate**(`wxWindow*` *window*)

Returns `true` if it is appropriate to update (send UI update events to) this window.

This function looks at the mode used (see `wxUpdateUIEvent::SetMode` (p. 1756)), the `wxWS_EX_PROCESS_UPDATE_EVENTS` flag in *window*, the time update events were last sent in idle time, and the update interval, to determine whether events should be sent to this window now. By default this will always return `true` because the update mode is initially `wxUPDATE_UI_PROCESS_ALL` and the interval is set to 0; so update events will be sent as often as possible. You can reduce the frequency that events are sent by changing the mode and/or setting an update interval.

#### See also

`wxUpdateUIEvent::ResetUpdateTime` (p. 1756), `wxUpdateUIEvent::SetUpdateInterval` (p. 1757), `wxUpdateUIEvent::SetMode` (p. 1756)

### **wxUpdateUIEvent::Check**

**void** **Check**(`bool` *check*)

Check or uncheck the UI element.

### **wxUpdateUIEvent::Enable**

**void** **Enable**(`bool` *enable*)

Enable or disable the UI element.

### **wxUpdateUIEvent::Show**

**void Show(bool show)**

Show or hide the UI element.

**wxUpdateUIEvent::GetChecked**

**bool GetChecked() const**

Returns true if the UI element should be checked.

**wxUpdateUIEvent::GetEnabled**

**bool GetEnabled() const**

Returns true if the UI element should be enabled.

**wxUpdateUIEvent::GetShown**

**bool GetShown() const**

Returns true if the UI element should be shown.

**wxUpdateUIEvent::GetSetChecked**

**bool GetSetChecked() const**

Returns true if the application has called *wxUpdateUIEvent::Check* (p. 1754). For wxWidgets internal use only.

**wxUpdateUIEvent::GetSetEnabled**

**bool GetSetEnabled() const**

Returns true if the application has called *wxUpdateUIEvent::Enable* (p. 1754). For wxWidgets internal use only.

**wxUpdateUIEvent::GetSetShown**

**bool GetSetShown() const**

Returns true if the application has called *wxUpdateUIEvent::Show* (p. 1754). For wxWidgets internal use only.

**wxUpdateUIEvent::GetSetText**

**bool GetSetText() const**

Returns true if the application has called *wxUpdateUIEvent::SetText* (p. 1757). For wxWidgets internal use only.

**wxUpdateUIEvent::GetText****wxString GetText() const**

Returns the text that should be set for the UI element.

**wxUpdateUIEvent::GetMode****static wxUpdateUIMode GetMode()**

Static function returning a value specifying how wxWidgets will send update events: to all windows, or only to those which specify that they will process the events.

See *wxUpdateUIEvent::SetMode* (p. 1756).

**wxUpdateUIEvent::GetUpdateInterval****static long GetUpdateInterval()**

Returns the current interval between updates in milliseconds. -1 disables updates, 0 updates as frequently as possible.

See *wxUpdateUIEvent::SetUpdateInterval* (p. 1757).

**wxUpdateUIEvent::ResetUpdateTime****static void ResetUpdateTime()**

Used internally to reset the last-updated time to the current time. It is assumed that update events are normally sent in idle time, so this is called at the end of idle processing.

**See also**

*wxUpdateUIEvent::CanUpdate* (p. 1754), *wxUpdateUIEvent::SetUpdateInterval* (p. 1757), *wxUpdateUIEvent::SetMode* (p. 1756)

**wxUpdateUIEvent::SetMode****static void SetMode(wxUpdateUIMode mode)**

Specify how wxWidgets will send update events: to all windows, or only to those which specify that they will process the events.

*mode* may be one of the following values. The default is `wxUPDATE_UI_PROCESS_ALL`.

```
enum wxUpdateUIMode
{
    // Send UI update events to all windows
    wxUPDATE_UI_PROCESS_ALL,

    // Send UI update events to windows that have
    // the wxWS_EX_PROCESS_UI_UPDATES flag specified
    wxUPDATE_UI_PROCESS_SPECIFIED
}
```

```
};
```

### **wxUpdateUIEvent::SetText**

**void SetText(const wxString& text)**

Sets the text for this UI element.

### **wxUpdateUIEvent::SetUpdateInterval**

**static void SetUpdateInterval(long updateInterval)**

Sets the interval between updates in milliseconds. Set to -1 to disable updates, or to 0 to update as frequently as possible. The default is 0.

Use this to reduce the overhead of UI update events if your application has a lot of windows. If you set the value to -1 or greater than 0, you may also need to call *wxWindow::UpdateWindowUI* (p. 1852) at appropriate points in your application, such as when a dialog is about to be shown.

## **wxURI**

wxURI is used to extract information from a URI (Uniform Resource Identifier).

For information about URIs, see RFC 3986 (<http://www.ietf.org/rfc/rfc3986.txt>).

In short, a URL *is* a URI. In other words, URL is a subset of a URI - all acceptable URLs are also acceptable URIs.

wxURI automatically escapes invalid characters in a string, so there is no chance of wxURI "failing" on construction/creation.

wxURI supports copy construction and standard assignment operators. wxURI can also be inherited from to provide further functionality.

### **Derived from**

*wxObject* (p. 1148)

### **Include files**

<wx/uri.h>

### **See also**

*wxURL* (p. 1763)

## **Obtaining individual components**

To obtain individual components you can use one of the following methods

*GetScheme* (p. 1761)  
*GetUserInfo* (p. 1761)  
*GetServer* (p. 1761)  
*GetPort* (p. 1760)  
*GetPath* (p. 1760)  
*GetQuery* (p. 1760)  
*GetFragment* (p. 1759)

However, you should check *HasXXX* before calling a get method, which determines whether or not the component referred to by the method is defined according to RFC 2396.

Consider an undefined component equivalent to a NULL C string.

*HasScheme* (p. 1762)  
*HasUserInfo* (p. 1762)  
*HasServer* (p. 1762)  
*HasPort* (p. 1762)  
*HasPath* (p. 1761)  
*HasQuery* (p. 1762)  
*HasFragment* (p. 1761)

```
Example://protocol will hold the http protocol (i.e. "http")
wxString protocol;
wxURI myuri(wxT("http://mysite.com"));
if(myuri.HasScheme())
    protocol = myuri.GetScheme();
```

## Deviations from the RFC

Note that on URIs with a "file" scheme *wxURI* does not parse the userinfo, server, or port portion. This is to keep compatibility with *wxFileSystem*, the old *wxURL*, and older url specifications.

## **wxURI::wxURI**

### **wxURI()**

Creates an empty URI.

### **wxURI(const wxChar\* uri)**

Constructor for quick creation.

*uri*

string to initialize with

### **wxURI(const wxURI& uri)**

Copies this URI from another URI.

*uri*

URI (Uniform Resource Identifier) to initialize with

### **wxURI::BuildURI**

#### **wxString BuildURI() const**

Builds the URI from its individual components and adds proper separators.

If the URI is not a reference or is not resolved, the URI that is returned from `Get` is the same one passed to `Create`.

### **wxURI::BuildUnescapedURI**

#### **wxString BuildUnescapedURI() const**

Builds the URI from its individual components, adds proper separators, and returns escape sequences to normal characters.

Note that it is preferred to call this over `Unescape(BuildURI())` since *BuildUnescapedURI* (p. 1759) performs some optimizations over the plain method.

### **wxURI::Create**

#### **const wxChar\* Create(const wxString&uri)**

Creates this URI from the string *uri*.

Returns the position at which parsing stopped (there is no such thing as an "invalid" `wxURI`).

*uri*

string to initialize from

### **wxURI::GetFragment**

#### **const wxString& GetFragment() const**

Obtains the fragment of this URI.

The fragment of a URI is the last value of the URI, and is the value after a `#` character after the path of the URI.

`http://mysite.com/mypath#<fragment>`

### **wxURI::GetHostType**

#### **const HostType& GetHostType() const**

Obtains the host type of this URI, which is of type `wxURI::HostType`:

<b>wxURI_REGNAME</b>	Server is a host name, or the Server component itself is undefined.
<b>wxURI_IPV4ADDRESS</b>	Server is a IP version 4 address (XXX.XXX.XXX.XXX)
<b>wxURI_IPV6ADDRESS</b>	Server is a IP version 6 address ((XXX:.)XXX::(XXX)XXX:XXX)
<b>wxURI_IPVFUTURE</b>	Server is an IP address, but not versions 4 or 6

### **wxURI::GetPassword**

#### **const wxString& GetPassword() const**

Returns the password part of the userinfo component of this URI. Note that this is explicitly deprecated by RFC 1396 and should generally be avoided if possible.

`http://<user>:<password>@mysite.com/mypath`

### **wxURI::GetPath**

#### **const wxString& GetPath() const**

Returns the (normalized) path of the URI.

The path component of a URI comes directly after the scheme component if followed by zero or one slashes ('/'), or after the server/port component.

Absolute paths include the leading '/' character.

`http://mysite.com<path>`

### **wxURI::GetPort**

#### **const wxString& GetPort() const**

Returns a string representation of the URI's port.

The Port of a URI is a value after the server, and must come after a colon (:).

`http://mysite.com:<port>`

Note that you can easily get the numeric value of the port by using `wxAtoi` or `wxString::Format`.

### **wxURI::GetQuery**

#### **const wxString& GetQuery() const**

Returns the Query component of the URI.

The query component is what is commonly passed to a cgi application, and must come



after the path component, and after a '?' character.

```
http://mysite.com/mypath?<query>
```

### **wxURI::GetScheme**

**const wxString& GetScheme() const**

Returns the Scheme component of the URI.

The first part of the uri.

```
<scheme>://mysite.com
```

### **wxURI::GetServer**

**const wxString& GetServer() const**

Returns the Server component of the URI.

The server of the uri can be a server name or a type of ip address. See *GetHostType* (p. 1759) for the possible values for the host type of the server component.

```
http://<server>/mypath
```

### **wxURI::GetUser**

**const wxString& GetUser() const**

Returns the username part of the userinfo component of this URI. Note that this is explicitly deprecated by RFC 1396 and should generally be avoided if possible.

```
http://<user>:<password>@mysite.com/mypath
```

### **wxURI::GetUserInfo**

**const wxString& GetUserInfo() const**

Returns the UserInfo component of the URI.

The component of a URI before the server component that is postfixed by a '@' character.

```
http://<userinfo>@mysite.com/mypath
```

### **wxURI::HasFragment**

**bool HasFragment() const**

Returns `true` if the Fragment component of the URI exists.

### **wxURI::HasPath**

**bool HasPath() const**

Returns `true` if the Path component of the URI exists.

**wxURI::HasPort****bool HasPort() const**

Returns `true` if the Port component of the URI exists.

**wxURI::HasQuery****bool HasQuery() const**

Returns `true` if the Query component of the URI exists.

**wxURI::HasScheme****bool HasScheme() const**

Returns `true` if the Scheme component of the URI exists.

**wxURI::HasServer****bool HasServer() const**

Returns `true` if the Server component of the URI exists.

**wxURI::HasUser****bool HasUser() const**

Returns `true` if the User component of the URI exists.

**wxURI::IsReference****bool IsReference() const**

Returns `true` if a valid [absolute] URI, otherwise this URI is a URI reference and not a full URI, and `IsReference` returns `false`.

**wxURI::operator ==****void operator ==(const wxURI& uricomp)**

Compares this URI to another URI, and returns `true` if this URI equals *uricomp*, otherwise it returns `false`.

*uricomp*

URI to compare to

### **wxURI::Resolve**

**void Resolve(const wxURI& base, int flags = wxURI\_STRICT)**

Inherits this URI from a base URI - components that do not exist in this URI are copied from the base, and if this URI's path is not an absolute path (prefixed by a '/'), then this URI's path is merged with the base's path.

For instance, resolving "../mydir" from "http://mysite.com/john/doe" results in the scheme (http) and server (mysite.com) being copied into this URI, since they do not exist. In addition, since the path of this URI is not absolute (does not begin with '/'), the path of the base's is merged with this URI's path, resulting in the URI "http://mysite.com/john/mydir".

*base*

Base URI to inherit from. Must be a full URI and not a reference

*flags*

Currently either wxURI\_STRICT or 0, in non-strict mode some compatibility layers are enabled to allow loopholes from RFCs prior to 2396

### **wxURI::Unescape**

**wxString Unescape(const wxString& uri)**

Translates all escape sequences (normal characters and returns the result.

This is the preferred over deprecated wxURL::ConvertFromURI.

If you want to unescape an entire wxURI, use *BuildUnescapedURI* (p. 1759) instead, as it performs some optimizations over this method.

*uri*

string with escaped characters to convert

## **wxURL**

wxURL is a specialization of wxURI (p. 1757) for parsing URLs. Please look at wxURI (p. 1757) documentation for more info about the functions you can use to retrieve the various parts of the URL (scheme, server, port, etc).

Supports standard assignment operators, copy constructors, and comparison operators.

### **Derived from**

wxURI (p. 1757)

wxObject (p. 1148)

**Include files**

<wx/url.h>

**See also**

*wxSocketBase* (p. 1472), *wxProtocol* (p. 1235)

**wxURL::wxURL**

**wxURL(const wxString& url = wxEmptyString)**

Constructs a URL object from the string. The URL must be valid according to RFC 1738. In particular, file URLs must be of the format `file://hostname/path/to/file` otherwise *GetError* (p. 1764) will return a value different from `wxURL_NOERR`.

It is valid to leave out the hostname but slashes must remain in place - i.e. a file URL without a hostname must contain three consecutive slashes (e.g. `file:///somepath/myfile`).

**Parameters**

*url*

Url string to parse.

**wxURL::~~wxURL**

**~wxURL()**

Destroys the URL object.

**wxURL::GetProtocol**

**wxProtocol& GetProtocol()**

Returns a reference to the protocol which will be used to get the URL.

**wxURL::GetError**

**wxURLError GetError() const**

Returns the last error. This error refers to the URL parsing or to the protocol. It can be one of these errors:

<b>wxURL_NOERR</b>	No error.
<b>wxURL_SNTAXERR</b>	Syntax error in the URL string.
<b>wxURL_NOPROTO</b>	Found no protocol which can get this URL.

<b>wxURL_NOHOST</b>	An host name is required for this protocol.
<b>wxURL_NOPATH</b>	A path is required for this protocol.
<b>wxURL_CONNERR</b>	Connection error.
<b>wxURL_PROTOERR</b>	An error occurred during negotiation.

### **wxURL::GetInputStream**

#### **wxInputStream \* GetInputStream()**

Creates a new input stream on the specified URL. You can use all but seek functionality of `wxStream`. Seek isn't available on all streams. For example, HTTP or FTP streams don't deal with it.

Note that this method is somewhat deprecated, all future `wxWidgets` applications should really use `wxFileSystem` (p. 633) instead.

Example:

```
wxURL url("http://a.host/a.dir/a.file");
if (url.GetError() == wxURL_NOERR)
{
    wxInputStream *in_stream;

    in_stream = url.GetInputStream();
    // Then, you can use all IO calls of in_stream (See wxStream)
}
```

#### **Return value**

Returns the initialized stream. You will have to delete it yourself.

#### **See also**

*wxInputStream* (p. 941)

### **wxURL::IsOk**

#### **bool IsOk() const**

Returns `true` if this object is correctly initialized, i.e. if *GetError* (p. 1764) returns `wxURL_NOERR`.

### **wxURL::SetDefaultProxy**

#### **static void SetDefaultProxy(const wxString& url\_proxy)**

Sets the default proxy server to use to get the URL. The string specifies the proxy like this: `<hostname>:<port number>`.

#### **Parameters**

*url\_proxy*

Specifies the proxy to use

**See also**

*wxURL::SetProxy* (p. 1766)

**wxURL::SetProxy**

**void SetProxy(const wxString& url\_proxy)**

Sets the proxy to use for this URL.

**See also**

*wxURL::SetDefaultProxy* (p. 1765)

**wxURL::SetURL**

**wxURLError SetURL(const wxString& url)**

Initializes this object with the given URL and returns `wxURL_NOERR` if it's valid (see *GetError* (p. 1764) for more info).

**wxURLDataObject**

`wxURLDataObject` is a *wxDataObject* (p. 311) containing an URL and can be used e.g. when you need to put an URL on or retrieve it from the clipboard:  
`wxTheClipboard->SetData(new wxURLDataObject(url));`

**Derived from**

Under MSW:

*wxDataObjectComposite* (p. 334)

*wxDataObject* (p. 311)

Under the other platforms:

*wxTextDataObject* (p. 1653)

*wxDataObjectSimple* (p. 335)

*wxDataObject* (p. 311)

**Include files**

`<wx/dataobj.h>`

**See also**

*Clipboard and drag and drop overview* (p. 2150),  
*wxDataObject* (p. 311)

**wxURLDataObject::wxURLDataObject****wxURLDataObject(const wxString& url = wxEmptyString)**

Constructor, may be used to initialize the URL. If *url* is empty, *SetURL* (p. 1767) can be used later.

**wxURLDataObject::GetURL****wxString GetURL() const**

Returns the URL stored by this object, as a string.

**wxURLDataObject::SetURL****void SetURL(const wxString& url)**

Sets the URL stored by this object.

**wxValidator**

wxValidator is the base class for a family of validator classes that mediate between a class of control, and application data.

A validator has three major roles:

1. to transfer data from a C++ variable or own storage to and from a control;
2. to validate data in a control, and show an appropriate error message;
3. to filter events (such as keystrokes), thereby changing the behaviour of the associated control.

Validators can be plugged into controls dynamically.

To specify a default, 'null' validator, use the symbol **wxDefaultValidator**.

For more information, please see *Validator overview* (p. 2092).

**wxPython note:** If you wish to create a validator class in wxPython you should derive the class from `wxPyValidator` in order to get Python-aware capabilities for the various virtual methods.

**Derived from**

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

**Include files**

<wx/validate.h>

### See also

*Validator overview* (p. 2092), *wxTextValidator* (p. 1668), *wxGenericValidator* (p. 714),

## **wxValidator::wxValidator**

**wxValidator()**

Constructor.

## **wxValidator::~~wxValidator**

**~wxValidator()**

Destructor.

## **wxValidator::Clone**

**virtual wxObject\* Clone() const**

All validator classes must implement the **Clone** function, which returns an identical copy of itself. This is because validators are passed to control constructors as references which must be copied. Unlike objects such as pens and brushes, it does not make sense to have a reference counting scheme to do this cloning, because all validators should have separate data.

This base function returns NULL.

## **wxValidator::GetWindow**

**wxWindow\* GetWindow() const**

Returns the window associated with the validator.

## **wxValidator::SetBellOnError**

**void SetBellOnError(bool dolt = true)**

This functions switches on or turns off the error sound produced by the validators if an invalid key is pressed.

## **wxValidator::SetWindow**

**void SetWindow(wxWindow\* window)**

Associates a window with the validator.



**wxValidator::TransferFromWindow****virtual bool TransferToWindow()**

This overridable function is called when the value in the window must be transferred to the validator. Return false if there is a problem.

**wxValidator::TransferToWindow****virtual bool TransferToWindow()**

This overridable function is called when the value associated with the validator must be transferred to the window. Return false if there is a problem.

**wxValidator::Validate****virtual bool Validate(wxWindow\* parent)**

This overridable function is called when the value in the associated window must be validated. Return false if the value in the window is not valid; you may pop up an error dialog.

**wxVariant**

The **wxVariant** class represents a container for any type. A variant's value can be changed at run time, possibly to a different type of value.

As standard, wxVariant can store values of type bool, wxChar, double, long, string, string list, time, date, void pointer, list of strings, and list of variants. However, an application can extend wxVariant's capabilities by deriving from the class *wxVariantData* (p. 1778) and using the wxVariantData form of the wxVariant constructor or assignment operator to assign this data to a variant. Actual values for user-defined types will need to be accessed via the wxVariantData object, unlike the case for basic data types where convenience functions such as *GetLong* (p. 1773) can be used.

Pointers to any *wxObject* (p. 1148) derived class can also easily be stored in a wxVariant. wxVariant will then use wxWidgets' built-in RTTI system to set the type name (returned by *GetType* (p. 1774)) and to perform type-safety checks at runtime.

This class is useful for reducing the programming for certain tasks, such as an editor for different data types, or a remote procedure call protocol.

An optional name member is associated with a wxVariant. This might be used, for example, in CORBA or OLE automation classes, where named parameters are required.

Note that as of wxWidgets 2.7.1, wxVariant is *reference counted* (p. 2046). Additionally, the convenience macros **DECLARE\_VARIANT\_OBJECT** and **IMPLEMENT\_VARIANT\_OBJECT** were added so that adding (limited) support for conversion to and from wxVariant can be very easily implemented without modifying either wxVariant or the class to be stored by wxVariant. Since assignment operators cannot be declared outside the class, the shift left operators are used like this:

```
// in the header file
DECLARE_VARIANT_OBJECT(MyClass)

// in the implementation file
IMPLEMENT_VARIANT_OBJECT(MyClass)

// in the user code
wxVariant variant;
MyClass value;
variant << value;

// or
value << variant;
```

For this to work, `MyClass` must derive from `wxObject` (p. 1148), implement the `wxWidgets RTTI system` (p. 2044) and support the assignment operator and equality operator for itself. Ideally, it should also be reference counted to make copying operations cheap and fast. This can be most easily implemented using the reference counting support offered by `wxObject` (p. 1148) itself. By default, `wxWidgets` already implements the shift operator conversion for a few of its drawing related classes:

```
IMPLEMENT_VARIANT_OBJECT(wxColour)
IMPLEMENT_VARIANT_OBJECT(wxImage)
IMPLEMENT_VARIANT_OBJECT(wxIcon)
IMPLEMENT_VARIANT_OBJECT(wxBitmap)
```

### Derived from

`wxObject` (p. 1148)

### Include files

<wx/variant.h>

### See also

`wxVariantData` (p. 1778)

## wxVariant::wxVariant

### wxVariant()

Default constructor.

### wxVariant(const wxVariant& variant)

Copy constructor, uses *reference counting* (p. 2046).

### wxVariant(const wxChar\* value, const wxString& name = "")

### wxVariant(const wxString& value, const wxString& name = "")

Construction from a string value.

**wxVariant(wxChar value, const wxString& name = "")**

Construction from a character value.

**wxVariant(long value, const wxString& name = "")**

Construction from an integer value. You may need to cast to (long) to avoid confusion with other constructors (such as the bool constructor).

**wxVariant(bool value, const wxString& name = "")**

Construction from a boolean value.

**wxVariant(double value, const wxString& name = "")**

Construction from a double-precision floating point value.

**wxVariant(const wxList& value, const wxString& name = "")**

Construction from a list of wxVariant objects. This constructor copies *value*, the application is still responsible for deleting *value* and its contents.

**wxVariant(void\* value, const wxString& name = "")**

Construction from a void pointer.

**wxVariant(wxObject\* value, const wxString& name = "")**

Construction from a wxObject pointer.

**wxVariant(wxVariantData\* data, const wxString& name = "")**

Construction from user-defined data. The variant holds onto the *data* pointer.

**wxVariant(wxDateTime& val, const wxString& name = "")**

Construction from a *wxDateTime* (p. 348).

**wxVariant(wxArrayString& val, const wxString& name = "")**

Construction from an array of strings. This constructor copies *value* and its contents.

**wxVariant(ODBC\_DATE\_STRUCT\* val, const wxString& name = "")**

Construction from a odbc date value. Represented internally by a *wxDateTime* (p. 348) value.

**wxVariant(ODBC\_TIME\_STRUCT\* val, const wxString& name = "")**

Construction from a odbc time value. Represented internally by a *wxDateTime* (p. 348) value.

**wxVariant(ODBC\_TIMESTAMP\_STRUCT\* val, const wxString& name = "")**

Construction from a odbc timestamp value. Represented internally by a *wxDateTime* (p. 348) value.

**wxVariant::~~wxVariant****~wxVariant()**

Destructor.

Note that destructor is protected, so `wxVariantData` cannot usually be deleted. Instead, *DecRef* (p. 1778) should be called. See *reference-counted object destruction* (p. 2046) for more info.

**wxVariant::Append****void Append(const wxVariant& value)**

Appends a value to the list.

**wxVariant::Clear****void Clear()**Makes the variant null by deleting the internal data and set the name to *wxEmptyString*.**wxVariant::ClearList****void ClearList()**

Deletes the contents of the list.

**wxVariant::Convert****bool Convert(long\* value) const****bool Convert(bool\* value) const****bool Convert(double\* value) const****bool Convert(wxString\* value) const****bool Convert(wxChar\* value) const****bool Convert(wxDateTime\* value) const**Retrieves and converts the value of this variant to the type that *value* is.**wxVariant::GetCount****size\_t GetCount() const**

Returns the number of elements in the list.

**wxVariant::Delete**

**bool Delete(size\_t item)**

Deletes the zero-based *item* from the list.

**wxVariant::GetArrayString**

**wxArrayString GetArrayString() const**

Returns the string array value.

**wxVariant::GetBool**

**bool GetBool() const**

Returns the boolean value.

**wxVariant::GetChar**

**wxChar GetChar() const**

Returns the character value.

**wxVariant::GetData**

**wxVariantData\* GetData() const**

Returns a pointer to the internal variant data. To take ownership of this data, you must call its *IncRef* (p. 1779) method. When you stop using it, *DecRef* (p. 1778) must be likewise called.

**wxVariant::GetDateTime**

**wxDateTime GetDateTime() const**

Returns the date value.

**wxVariant::GetDouble**

**double GetDouble() const**

Returns the floating point value.

**wxVariant::GetLong**

**long GetLong() const**

Returns the integer value.

**wxVariant::GetName**

**const wxString& GetName() const**

Returns a constant reference to the variant name.

**wxVariant::GetString**

**wxString GetString() const**

Gets the string value.

**wxVariant::GetType**

**wxString GetType() const**

Returns the value type as a string. The built-in types are: bool, char, date, double, list, long, string, stringlist, time, void\*.

If the variant is null, the value type returned is the string "null" (not the empty string).

**wxVariant::GetVoidPtr**

**void\* GetVoidPtr() const**

Gets the void pointer value.

**wxVariant::GetWxObjectPtr**

**wxObject\* GetWxObjectPtr() const**

Gets the wxObject pointer value.

**wxVariant::Insert**

**void Insert(const wxVariant& value)**

Inserts a value at the front of the list.

**wxVariant::IsNull**

**bool IsNull() const**

Returns true if there is no data associated with this variant, false if there is data.

**wxVariant::IsType**

**bool IsType(const wxString& type) const**

Returns true if *type* matches the type of the variant, false otherwise.

**wxVariant::IsValueKindOf**

**bool IsValueKindOf(const wxClassInfo\* type type) const**

Returns true if the data is derived from the class described by *type*, false otherwise.

**wxVariant::MakeNull**

**void MakeNull()**

Makes the variant null by deleting the internal data.

**wxVariant::MakeString**

**wxString MakeString() const**

Makes a string representation of the variant value (for any type).

**wxVariant::Member**

**bool Member(const wxVariant& value) const**

Returns true if *value* matches an element in the list.

**wxVariant::NullList**

**void NullList()**

Makes an empty list. This differs from a null variant which has no data; a null list is of type list, but the number of elements in the list is zero.

**wxVariant::SetData**

**void SetData(wxVariantData\* data)**

Sets the internal variant data, deleting the existing data if there is any.

**wxVariant::operator =**

**void operator =(const wxVariant& value)**

**void operator =(wxVariantData\* value)**

**void operator =(const wxString& value)**

**void operator =(const wxChar\* value)**

**void operator =(wxChar value)**

**void operator =(const long value)**

**void operator =(const bool value)**

**void operator =(const double value)**

```
void operator =(void* value)
void operator =(wxObject* value)
void operator =(const wxList& value)
void operator =(const wxDateTime& value)
void operator =(const wxArrayString& value)
void operator =(const DATE_STRUCT* value)
void operator =(const TIME_STRUCT* value)
void operator =(const TIMESTAMP_STRUCT* value)
```

Assignment operators, using *reference counting* (p. 2046) when possible.

**wxVariant::operator ==**

```
bool operator ==(const wxVariant& value) const
bool operator ==(const wxString& value) const
bool operator ==(const wxChar* value) const
bool operator ==(wxChar value) const
bool operator ==(const long value) const
bool operator ==(const bool value) const
bool operator ==(const double value) const
bool operator ==(void* value) const
bool operator ==(wxObject* value) const
bool operator ==(const wxList& value) const
bool operator ==(const wxArrayString& value) const
bool operator ==(const wxDateTime& value) const
```

Equality test operators.

**wxVariant::operator !=**

```
bool operator !=(const wxVariant& value) const
bool operator !=(const wxString& value) const
bool operator !=(const wxChar* value) const
bool operator !=(wxChar value) const
```



**bool operator !=(const long *value*) const**

**bool operator !=(const bool *value*) const**

**bool operator !=(const double *value*) const**

**bool operator !=(void\* *value*) const**

**bool operator !=(wxObject\* *value*) const**

**bool operator !=(const wxList& *value*) const**

**bool operator !=(const wxString& *value*) const**

**bool operator !=(const wxDateTime& *value*) const**

Inequality test operators.

**wxVariant::operator []**

**wxVariant operator [](*size\_t idx*) const**

Returns the value at *idx* (zero-based).

**wxVariant& operator [](*size\_t idx*)**

Returns a reference to the value at *idx* (zero-based). This can be used to change the value at this index.

**wxVariant::operator wxChar**

**char operator wxChar() const**

Operator for implicit conversion to a wxChar, using *wxVariant::GetChar* (p. 1773).

**wxVariant::operator double**

**double operator double() const**

Operator for implicit conversion to a double, using *wxVariant::GetDouble* (p. 1773).

**long operator long() const**

Operator for implicit conversion to a long, using *wxVariant::GetLong* (p. 1773).

**wxVariant::operator wxString**

**wxString operator wxString() const**

Operator for implicit conversion to a string, using *wxVariant::MakeString* (p. 1775).

**wxVariant::operator void\***

**void\* operator void\*() const**

Operator for implicit conversion to a pointer to a void, using *wxVariant::GetVoidPtr* (p. 1774).

**wxVariant::operator wxDateTime****void\* operator wxDateTime() const**

Operator for implicit conversion to a pointer to a *wxDateTime* (p. 348), using *wxVariant::GetDateTime* (p. 1773).

**wxVariantData**

The **wxVariantData** class is used to implement a new type for *wxVariant* (p. 1769). Derive from *wxVariantData*, and override the pure virtual functions.

*wxVariantData* is *reference counted* (p. 2046), but you don't normally have to care about this, as *wxVariant* manages the count automatically. However, in case your application needs to take ownership of *wxVariantData*, be aware that the object is created with reference count of 1, and passing it to *wxVariant* will not increase this. In other words, *IncRef* (p. 1779) needs to be called only if you both take ownership of *wxVariantData* and pass it to a *wxVariant*. Also note that the destructor is protected, so you can never explicitly delete a *wxVariantData* instance. Instead, *DecRef* (p. 1778) will delete the object automatically when the reference count reaches zero.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/variant.h>

**See also**

*wxVariant* (p. 1769)

**wxVariantData::wxVariantData****wxVariantData()**

Default constructor.

**wxVariantData::DecRef****void DecRef()**

Decreases reference count. If the count reaches zero, the object is automatically deleted.

Note that destructor of `wxVariantData` is protected, so `delete` cannot be used as normal. Instead, *DecRef* (p. 1778) should be called.

### **wxVariantData::Eq**

**bool Eq(wxVariantData& data) const**

Returns true if this object is equal to *data*.

### **wxVariantData::GetType**

**wxString GetType() const**

Returns the string type of the data.

### **wxVariantData::GetValueClassInfo**

**wxClassInfo\* GetValueClassInfo() const**

If the data is a `wxObject` returns a pointer to the objects `wxClassInfo` structure, if the data isn't a `wxObject` the method returns `NULL`.

### **wxVariantData::IncRef**

**void IncRef()**

Increases reference count. Note that initially `wxVariantData` has reference count of 1.

### **wxVariantData::Read**

**bool Read(ostream& stream)**

**bool Read(wxString& string)**

Reads the data from *stream* or *string*.

### **wxVariantData::Write**

**bool Write(ostream& stream) const**

**bool Write(wxString& string) const**

Writes the data to *stream* or *string*.

### **wxGetVariantCast**

**classname \* wxGetVariantCast(wxVariant&, classname)**

This macro returns the data stored in *variant* cast to the type *classname* \* if the data is of this type (the check is done during the run-time) or `NULL` otherwise.

**See also**

*RTTI overview* (p. 2044)  
*wxDynamicCast* (p. 1969)

**wxView**

The view class can be used to model the viewing and editing component of an application's file-based data. It is part of the document/view framework supported by *wxWidgets*, and cooperates with the *wxDocument* (p. 545), *wxDocTemplate* (p. 540) and *wxDocManager* (p. 527) classes.

**Derived from**

*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/docview.h>

**See also**

*wxView overview* (p. 2134), *wxDocument* (p. 545), *wxDocTemplate* (p. 540),  
*wxDocManager* (p. 527)

**wxView::m\_viewDocument****wxDocument\* m\_viewDocument**

The document associated with this view. There may be more than one view per document, but there can never be more than one document for one view.

**wxView::m\_viewFrame****wxFrame\* m\_viewFrame**

Frame associated with the view, if any.

**wxView::m\_viewTypeName****wxString m\_viewTypeName**

The view type name given to the *wxDocTemplate* constructor, copied to this variable when the view is created. Not currently used by the framework.

**wxView::wxView****wxView()**

Constructor. Define your own default constructor to initialize application-specific data.

### **wxView::~~wxView**

**~wxView()**

Destructor. Removes itself from the document's list of views.

### **wxView::Activate**

**virtual void Activate(bool activate)**

Call this from your view frame's OnActivate member to tell the framework which view is currently active. If your windowing system doesn't call OnActivate, you may need to call this function from 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.

This function calls wxView::OnActivateView.

### **wxView::Close**

**virtual bool Close(bool deleteWindow = true)**

Closes the view by calling OnClose. If *deleteWindow* is true, this function should delete the window associated with the view.

### **wxView::GetDocument**

**wxDocument\* GetDocument() const**

Gets a pointer to the document associated with the view.

### **wxView::GetDocumentManager**

**wxDocManager\* GetDocumentManager() const**

Returns a pointer to the document manager instance associated with this view.

### **wxView::GetFrame**

**wxWindow \* GetFrame()**

Gets the frame associated with the view (if any). Note that this "frame" is not a wxFrame at all in the generic MDI implementation which uses the notebook pages instead of the frames and this is why this method returns a wxWindow and not a wxFrame.

### **wxView::GetViewName**

**wxString GetViewName() const**

Gets the name associated with the view (passed to the wxDocTemplate constructor). Not currently used by the framework.

**wxView::OnActivateView**

**virtual void OnActivateView**(bool *activate*, wxView \**activeView*, wxView \**deactiveView*)

Called when a view is activated by means of wxView::Activate. The default implementation does nothing.

**wxView::OnChangeFilename**

**virtual void OnChangeFilename()**

Called when the filename has changed. The default implementation constructs a suitable title and sets the title of the view frame (if any).

**wxView::OnClose**

**virtual bool OnClose**(bool *deleteWindow*)

Implements closing behaviour. The default implementation calls wxDocument::Close to close the associated document. Does not delete the view. The application may wish to do some cleaning up operations in this function, *if* a call to wxDocument::Close succeeded. For example, if your views all share the same window, you need to disassociate the window from the view and perhaps clear the window. If *deleteWindow* is true, delete the frame associated with the view.

**wxView::OnClosingDocument**

**virtual void OnClosingDoocument()**

Override this to clean up the view when the document is being closed.

**wxView::OnCreate**

**virtual bool OnCreate**(wxDocument\* *doc*, long *flags*)

wxDocManager or wxDocument creates a wxView via a wxDocTemplate. Just after the wxDocTemplate creates the wxView, it calls wxView::OnCreate. In its OnCreate member function, the wxView can create a wxDocChildFrame or a derived class. This wxDocChildFrame provides user interface elements to view and/or edit the contents of the wxDocument.

By default, simply returns true. If the function returns false, the view will be deleted.

**wxView::OnCreatePrintout**

**virtual wxPrintout\* OnCreatePrintout()**

If the printing framework is enabled in the library, this function returns a *wxPrintout* (p. 1214) object for the purposes of printing. It should create a new object every time 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::OnDraw****virtual void OnDraw(wxDC\* dc)**

Override this function to render the view on the given device context.

**wxView::OnUpdate****virtual void OnUpdate(wxView\* sender, wxObject\* hint)**

Called when the view should be updated. *sender* is a pointer to the view that sent the update request, or NULL if no single view requested the update (for instance, when the document is opened). *hint* is as yet unused but may in future contain application-specific information for making updating more efficient.

**wxView::SetDocument****void SetDocument(wxDocument\* doc)**

Associates the given document with the view. Normally called by the framework.

**wxView::SetFrame****void SetFrame(wxWindow\* frame)**

Sets the frame associated with this view. The application should call this if possible, to tell the view about the frame.

See *GetFrame* (p. 1781) for the explanation about the mismatch between the "Frame" in the method name and the type of its parameter.

**wxView::SetViewName****void SetViewName(const wxString& name)**

Sets the view type name. Should only be called by the framework.

**wxVListBox**

`wxVListBox` is a listbox-like control with the following two main differences from a regular listbox: it can have an arbitrarily huge number of items because it doesn't store them itself but uses `OnDrawItem()` (p. 1787) callback to draw them (so it is a **V**irtual listbox) and its items can have variable height as determined by `OnMeasureItem()` (p. 1788) (so it is also a listbox with the lines of **V**ariable height).

Also, as a consequence of its virtual nature, it doesn't have any methods to append or insert items in it as it isn't necessary to do it: you just have to call `SetItemCount()` (p. 1789) to tell the control how many items it should display. Of course, this also means that you will never use this class directly because it has pure virtual functions, but will need to derive your own class, such as `wxHtmlListBox` (p. 853), from it.

However it emits the same events as `wxListBox` (p. 974) and the same event macros may be used with it.

### Derived from

`wxVScrolledWindow` (p. 1790)  
`wxPanel` (p. 1170)  
`wxWindow` (p. 1795)  
`wxEvtHandler` (p. 576)  
`wxObject` (p. 1148)

### Include files

<wx/vlbox.h>

### See also

`wxSimpleHtmlListBox` (p. 855), `wxHtmlListBox` (p. 853)

## `wxVListBox::wxVListBox`

**`wxVListBox(wxWindow* parent, wxWindowID id = wxID_ANY, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0, const wxString& name = wxVListBoxNameStr)`**

Normal constructor which calls `Create()` (p. 1784) internally.

**`wxVListBox()`**

Default constructor, you must call `Create()` (p. 1784) later.

## `wxVListBox::Clear`

**`void Clear()`**

Deletes all items from the control.

## `wxVListBox::Create`



**bool Create(wxWindow\* parent, wxWindowID id = wxID\_ANY, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0, const wxString& name = wxVListBoxNameStr)**

Creates the control. To finish creating it you also should call *SetItemCount()* (p. 1789) to let it know about the number of items it contains.

The only special style which may be used with `wxVListBox` is `wxLB_MULTIPLE` which indicates that the listbox should support multiple selection.

Returns `true` on success or `false` if the control couldn't be created

### **wxVListBox::DeselectAll**

**bool DeselectAll()**

Deselects all the items in the listbox.

Returns `true` if any items were changed, i.e. if there had been any selected items before, or `false` if all the items were already deselected.

This method is only valid for multi selection listboxes.

### **See also**

*SelectAll* (p. 1789), *Select* (p. 1788)

### **wxVListBox::GetFirstSelected**

**int GetFirstSelected(unsigned long& cookie) const**

Returns the index of the first selected item in the listbox or `wxNOT_FOUND` if no items are currently selected.

*cookie* is an opaque parameter which should be passed to the subsequent calls to *GetNextSelected* (p. 1786). It is needed in order to allow parallel iterations over the selected items.

```
Here is a typical example of using these functions:
unsigned long cookie;
int item = hlbox->GetFirstSelected(cookie);
while ( item != wxNOT_FOUND )
{
    ... process item ...
    item = hlbox->GetNextSelected(cookie);
}
```

This method is only valid for multi selection listboxes.

### **wxVListBox::GetItemCount**

**size\_t GetItemCount() const**

Get the number of items in the control.

**See also**

*SetItemCount()* (p. 1789)

**wxVListBox::GetMargins****wxPoint GetMargins() const**

Returns the margins used by the control. The *x* field of the returned point is the horizontal margin and the *y* field is the vertical one.

**See also**

*SetMargins* (p. 1789)

**wxVListBox::GetNextSelected****int GetNextSelected(unsigned long& cookie) const**

Returns the index of the next selected item or `wxNOT_FOUND` if there are no more.

This method is only valid for multi selection listboxes.

**See also**

*GetFirstSelected* (p. 1785)

**wxVListBox::GetSelectedCount****size\_t GetSelectedCount() const**

Returns the number of the items currently selected.

It is valid for both single and multi selection controls. In the former case it may only return 0 or 1 however.

**See also**

*IsSelected* (p. 1787),  
*GetFirstSelected* (p. 1785),  
*GetNextSelected* (p. 1786)

**wxVListBox::GetSelection****int GetSelection() const**

Get the currently selected item or `wxNOT_FOUND` if there is no selection.

**wxVListBox::GetSelectionBackground****const wxColour& GetSelectionBackground() const**

Returns the background colour used for the selected cells. By default the standard system colour is used.

**See also**

*wxSystemSettings::GetColour* (p. 1594),  
*SetSelectionBackground* (p. 1790)

**wxVListBox::HasMultipleSelection****bool HasMultipleSelection() const**

Returns `true` if the listbox was created with `wxLB_MULTIPLE` style and so supports multiple selection or `false` if it is a single selection listbox.

**wxVListBox::IsCurrent****bool IsCurrent(size\_t item) const**

Returns `true` if this item is the current one, `false` otherwise.

Current item is always the same as selected one for the single selection listbox and in this case this method is equivalent to *IsSelected* (p. 1787) but they are different for multi selection listboxes where many items may be selected but only one (at most) is current.

**wxVListBox::IsSelected****bool IsSelected(size\_t item) const**

Returns `true` if this item is selected, `false` otherwise.

**wxVListBox::OnDrawBackground****void OnDrawBackground(wxDC& dc, const wxRect& rect, size\_t n) const**

This method is used to draw the items background and, maybe, a border around it.

The base class version implements a reasonable default behaviour which consists in drawing the selected item with the standard background colour and drawing a border around the item if it is either selected or current.

**wxVListBox::OnDrawItem****void OnDrawItem(wxDC& dc, const wxRect& rect, size\_t n) const**

The derived class must implement this function to actually draw the item with the given index on the provided DC.

**Parameters**

*dc*

The device context to use for drawing

*rect*

The bounding rectangle for the item being drawn (DC clipping region is set to this rectangle before calling this function)

*n*

The index of the item to be drawn

### **wxVListBox::OnDrawSeparator**

**void OnDrawSeparator(wxDC& dc, wxRect& rect, size\_t n) const**

This method may be used to draw separators between the lines. The rectangle passed to it may be modified, typically to deflate it a bit before passing to *OnDrawItem()* (p. 1787).

The base class version of this method doesn't do anything.

#### **Parameters**

*dc*

The device context to use for drawing

*rect*

The bounding rectangle for the item

*n*

The index of the item

### **wxVListBox::OnMeasureItem**

**wxCoord OnMeasureItem(size\_t n) const**

The derived class must implement this method to return the height of the specified item (in pixels).

### **wxVListBox::Select**

**bool Select(size\_t item, bool select = true)**

Selects or deselects the specified item which must be valid (i.e. not equal to `wxNOT_FOUND`).

Return `true` if the items selection status has changed or `false` otherwise.

This function is only valid for the multiple selection listboxes, use *SetSelection* (p. 1789) for the single selection ones.

**wxVListBox::SelectAll****bool SelectAll()**

Selects all the items in the listbox.

Returns `true` if any items were changed, i.e. if there had been any unselected items before, or `false` if all the items were already selected.

This method is only valid for multi selection listboxes.

**See also**

*DeselectAll* (p. 1785), *Select* (p. 1788)

**wxVListBox::SelectRange****bool SelectRange(size\_t from, size\_t to)**

Selects all items in the specified range which may be given in any order.

Return `true` if the items selection status has changed or `false` otherwise.

This method is only valid for multi selection listboxes.

**See also**

*SelectAll* (p. 1789), *Select* (p. 1788)

**wxVListBox::SetItemCount****void SetItemCount(size\_t count)**

Set the number of items to be shown in the control.

This is just a synonym for *wxVScrolledWindow::SetLineCount()* (p. 1795).

**wxVListBox::SetMargins****void SetMargins(const wxPoint& pt)****void SetMargins(wxCoord x, wxCoord y)**

Set the margins: horizontal margin is the distance between the window border and the item contents while vertical margin is half of the distance between items.

By default both margins are 0.

**wxVListBox::SetSelection****void SetSelection(int selection)**

Set the selection to the specified item, if it is -1 the selection is unset. The selected item will

be automatically scrolled into view if it isn't currently visible.

This method may be used both with single and multiple selection listboxes.

### **wxVListBox::SetSelectionBackground**

**void SetSelectionBackground(const wxColour& col)**

Sets the colour to be used for the selected cells background. The background of the standard cells may be changed by simply calling *SetBackgroundColour* (p. 1835).

#### **See also**

*GetSelectionBackground* (p. 1786)

### **wxVListBox::Toggle**

**void Toggle(size\_t item)**

Toggles the state of the specified *item*, i.e. selects it if it was unselected and deselects it if it was selected.

This method is only valid for multi selection listboxes.

#### **See also**

*Select* (p. 1788)

## **wxVScrolledWindow**

In the name of this class, "V" may stand for "variable" because it can be used for scrolling lines of variable heights; "virtual" because it is not necessary to know the heights of all lines in advance -- only those which are shown on the screen need to be measured; or, even, "vertical" because this class only supports scrolling in one direction currently (this could and probably will change in the future however).

In any case, this is a generalization of the *wxScrolledWindow* (p. 1414) class which can be only used when all lines have the same height. It lacks some other *wxScrolledWindow* features however, notably there is currently no support for horizontal scrolling; it can't scroll another window nor only a rectangle of the window and not its entire client area.

To use this class, you need to derive from it and implement *OnGetLineHeight()* (p. 1793) pure virtual method. You also must call *SetLineCount* (p. 1795) to let the base class know how many lines it should display but from that moment on the scrolling is handled entirely by *wxVScrolledWindow*, you only need to draw the visible part of contents in your *OnPaint()* method as usual. You should use *GetFirstVisibleLine()* (p. 1792) and *GetLastVisibleLine()* (p. 1792) to select the lines to display. Note that the device context origin is not shifted so the first visible line always appears at the point (0, 0) in physical as well as logical coordinates.

#### **Derived from**

*wxPanel* (p. 1170)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

### Include files

<wx/vscroll.h>

## **wxVScrolledWindow::wxVScrolledWindow**

**wxVScrolledWindow**(*wxWindow\** parent, *wxWindowID* id = *wxID\_ANY*, **const** *wxPoint&* pos = *wxDefaultPosition*, **const** *wxSize&* size = *wxDefaultSize*, **long** style = 0, **const** *wxString&* name = *wxPanelNameStr*)

This is the normal constructor, no need to call *Create()* after using this one.

Note that *wxVSCROLL* is always automatically added to our style, there is no need to specify it explicitly.

### **wxVScrolledWindow()**

Default constructor, you must call *Create()* (p. 1791) later.

### Parameters

*parent*

The parent window, must not be *NULL*

*id*

The identifier of this window, *wxID\_ANY* by default

*pos*

The initial window position

*size*

The initial window size

*style*

The window style. There are no special style bits defined for this class.

*name*

The name for this window; usually not used

## **wxVScrolledWindow::Create**

**bool Create(wxWindow\* parent, wxWindowID id = wxID\_ANY, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long style = 0, const wxString& name = wxPanelNameStr)**

Same as the *non default ctor* (p. 1791) but returns status code: `true` if ok, `false` if the window couldn't have been created.

Just as with the ctor above, `wxVSCROLL` style is always used, there is no need to specify it explicitly.

### **wxVScrolledWindow::EstimateTotalHeight**

**virtual wxCoord EstimateTotalHeight() const**

This protected function is used internally by `wxVScrolledWindow` to estimate the total height of the window when *SetLineCount* (p. 1795) is called. The default implementation uses the brute force approach if the number of the items in the control is small enough. Otherwise, it tries to find the average line height using some lines in the beginning, middle and the end.

If it is undesirable to access all these lines (some of which might be never shown) just for the total height calculation, you may override the function and provide your own guess better and/or faster.

Note that although returning a totally wrong value would still work, it risks to result in very strange scrollbar behaviour so this function should really try to make the best guess possible.

### **wxVScrolledWindow::GetFirstVisibleLine**

**size\_t GetFirstVisibleLine() const**

Returns the index of the first currently visible line.

This is same as *GetVisibleBegin* (p. 1793) and exists only for symmetry with *GetLastVisibleLine* (p. 1792).

### **wxVScrolledWindow::GetLastVisibleLine**

**size\_t GetLastVisibleLine() const**

Returns the index of the last currently visible line. Note that this method returns `(size_t)-1` (i.e. a huge positive number) if the control is empty so if this is possible you should use *GetVisibleEnd* (p. 1793) instead.

### **See also**

*GetFirstVisibleLine* (p. 1792)

### **wxVScrolledWindow::GetLineCount**



**size\_t GetLineCount() const**

Get the number of lines this window contains (previously set by *SetLineCount()* (p. 1795))

**wxVScrolledWindow::GetVisibleBegin****size\_t GetVisibleBegin() const**

Returns the index of the first currently visible line.

**See also**

*GetVisibleEnd* (p. 1793)

**wxVScrolledWindow::GetVisibleEnd****size\_t GetVisibleEnd() const**

Returns the index of the first line after the currently visible one. If the return value is 0 it means that no lines are currently shown (which only happens if the control is empty). Note that the index returned by this method is not always a valid index as it may be equal to *GetLineCount* (p. 1792).

**See also**

*GetVisibleBegin* (p. 1793)

**wxVScrolledWindow::HitTest****int HitTest(wxCoord x, wxCoord y) const****int HitTest(const wxPoint& pt) const**

Return the item at the specified (in physical coordinates) position or `wxNOT_FOUND` if none, i.e. if it is below the last item.

**wxVScrolledWindow::IsVisible****bool IsVisible(size\_t line) const**

Returns `true` if the given line is (at least partially) visible or `false` otherwise.

**wxVScrolledWindow::OnGetLineHeight****virtual wxCoord OnGetLineHeight(size\_t n) const**

This protected virtual function must be overridden in the derived class and it should return the height of the given line in pixels.

**See also**

*OnGetLinesHint* (p. 1794)

**wxVScrolledWindow::OnGetLinesHint****virtual void OnGetLinesHint(size\_t lineMin, size\_t lineMax) const**

This function doesn't have to be overridden but it may be useful to do it if calculating the lines heights is a relatively expensive operation as it gives the user code a possibility to calculate several of them at once.

`OnGetLinesHint()` is normally called just before `OnGetLineHeight()` (p. 1793) but you shouldn't rely on the latter being called for all lines in the interval specified here. It is also possible that `OnGetLineHeight()` will be called for the lines outside of this interval, so this is really just a hint, not a promise.

Finally note that *lineMin* is inclusive, while *lineMax* is exclusive, as usual.

**wxVScrolledWindow::RefreshLine****void RefreshLine(size\_t line)**

Refreshes the specified line -- it will be redrawn during the next main loop iteration.

**See also**

*RefreshLines* (p. 1794)

**wxVScrolledWindow::RefreshLines****void RefreshLines(size\_t from, size\_t to)**

Refreshes all lines between *from* and *to*, inclusive. *from* should be less than or equal to *to*.

**See also**

*RefreshLine* (p. 1794)

**wxVScrolledWindow::RefreshAll****void RefreshAll()**

This function completely refreshes the control, recalculating the number of items shown on screen and repainting them. It should be called when the values returned by *OnGetLineHeight* (p. 1793) change for some reason and the window must be updated to reflect this.

**wxVScrolledWindow::ScrollLines****bool ScrollLines(int lines)**

Scroll by the specified number of lines which may be positive (to scroll down) or negative (to scroll up).

Returns `true` if the window was scrolled, `false` otherwise (for example if we're trying to

scroll down but we are already showing the last line).

**See also**

*LineUp* (p. 1825), *LineDown* (p. 1825)

**wxVScrolledWindow::ScrollPages**

**bool ScrollPages**(int *pages*)

Scroll by the specified number of pages which may be positive (to scroll down) or negative (to scroll up).

**See also**

*ScrollLines* (p. 1794),  
*PageUp* (p. 1828), *PageDown* (p. 1828)

**wxVScrolledWindow::ScrollToLine**

**bool ScrollToLine**(size\_t *line*)

Scroll to the specified line: it will become the first visible line in the window.

Return `true` if we scrolled the window, `false` if nothing was done.

**wxVScrolledWindow::SetLineCount**

**void SetLineCount**(size\_t *count*)

Set the number of lines the window contains: the derived class must provide the heights for all lines with indices up to the one given here in its *OnGetLineHeight()* (p. 1793).

## wxWindow

`wxWindow` is the base class for all windows and represents any visible object on screen. All controls, top level windows and so on are windows. Sizers and device contexts are not, however, as they don't appear on screen themselves.

Please note that all children of the window will be deleted automatically by the destructor before the window itself is deleted which means that you don't have to worry about deleting them manually. Please see the *window deletion overview* (p. 2089) for more information.

Also note that in this, and many others, `wxWidgets` classes some `GetXXX()` methods may be overloaded (as, for example, *GetSize* (p. 1818) or *GetClientSize* (p. 1811)). In this case, the overloads are non-virtual because having multiple virtual functions with the same name results in a virtual function name hiding at the derived class level (in English, this means that the derived class has to override all overloaded variants if it overrides any of them). To allow overriding them in the derived class, `wxWidgets` uses a unique protected virtual `DoGetXXX()` method and all `GetXXX()` ones are forwarded to it, so overriding the

former changes the behaviour of the latter.

### Derived from

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### Include files

<wx/window.h>

### Window styles

The following styles can apply to all windows, although they will not always make sense for a particular window class or on all platforms.

<b>wxBORDER_SIMPLE</b>	Displays a thin border around the window. wxSIMPLE_BORDER is the old name for this style.
<b>wxBORDER_DOUBLE</b>	Displays a double border. wxDOUBLE_BORDER is the old name for this style. Windows and Mac only.
<b>wxBORDER_SUNKEN</b>	Displays a sunken border. wxSUNKEN_BORDER is the old name for this style.
<b>wxBORDER_RAISED</b>	Displays a raised border. wxRAISED_BORDER is the old name for this style.
<b>wxBORDER_STATIC</b>	Displays a border suitable for a static control. wxSTATIC_BORDER is the old name for this style. Windows only.
<b>wxBORDER_THEME</b>	Displays a themed border where possible. Currently this has an effect on Windows XP and above only. For more information on themed borders, please see <i>Themed borders on Windows</i> (p. 2235).
<b>wxBORDER_NONE</b>	Displays no border, overriding the default border style for the window. wxNO_BORDER is the old name for this style.
<b>wxTRANSPARENT_WINDOW</b>	The window is transparent, that is, it will not receive paint events. Windows only.
<b>wxTAB_TRAVERSAL</b>	Use this to enable tab traversal for non-dialog windows.
<b>wxWANTS_CHARS</b>	Use this to indicate that the window wants to get all char/key events for all keys - even for keys like TAB or ENTER which are usually used for dialog navigation and which wouldn't be generated without this style. If you need to use this style in order to get the arrows or etc., but would still like to have normal keyboard navigation take place, you should create and send a <code>wxNavigationKeyEvent</code> in response to the key events for Tab and Shift-Tab.

- wxNO\_FULL\_REPAINT\_ON\_RESIZE** On Windows, this style used to disable repainting the window completely when its size is changed. Since this behaviour is now the default, the style is now obsolete and no longer has an effect.
- wxVSCROLL** Use this style to enable a vertical scrollbar.
- wxHSCROLL** Use this style to enable a horizontal scrollbar.
- wxALWAYS\_SHOW\_SB** If a window has scrollbars, disable them instead of hiding them when they are not needed (i.e. when the size of the window is big enough to not require the scrollbars to navigate it). This style is currently implemented for wxMSW, wxGTK and wxUniversal and does nothing on the other platforms.
- wxCLIP\_CHILDREN** Use this style to eliminate flicker caused by the background being repainted, then children being painted over them. Windows only.
- wxFULL\_REPAINT\_ON\_RESIZE** Use this style to force a complete redraw of the window whenever it is resized instead of redrawing just the part of the window affected by resizing. Note that this was the behaviour by default before 2.5.1 release and that if you experience redraw problems with code which previously used to work you may want to try this. Currently this style applies on GTK+ 2 and Windows only, and full repainting is always done on other platforms.

See also *window styles overview* (p. 2089).

### Extra window styles

The following are extra styles, set using `wxWindow::SetExtraStyle` (p. 1839).

- wxWS\_EX\_VALIDATE\_RECURSIVELY** By default, `Validate/TransferDataTo/FromWindow()` only work on direct children of the window (compatible behaviour). Set this flag to make them recursively descend into all subwindows.
- wxWS\_EX\_BLOCK\_EVENTS** `wxCommandEvent`s and the objects of the derived classes are forwarded to the parent window and so on recursively by default. Using this flag for the given window allows to block this propagation at this window, i.e. prevent the events from being propagated further upwards. Dialogs have this flag on by default.
- wxWS\_EX\_TRANSIENT** Don't use this window as an implicit parent for the other windows: this must be used with transient windows as otherwise there is the risk of creating a dialog/frame with this window as a parent which would lead to a crash if the parent is destroyed before the child.

**wxWS\_EX\_PROCESS\_IDLE** This window should always process idle events, even if the mode set by *wxIdleEvent::SetMode* (p. 905) is *wxIDLE\_PROCESS\_SPECIFIED*.

**wxWS\_EX\_PROCESS\_UI\_UPDATES** This window should always process UI update events, even if the mode set by *wxUpdateUIEvent::SetMode* (p. 1756) is *wxUPDATE\_UI\_PROCESS\_SPECIFIED*.

### See also

*Event handling overview* (p. 2077)

*Window sizing overview* (p. 2042)

## wxWindow::wxWindow

### wxWindow()

Default constructor.

**wxWindow**(*wxWindow\** parent, **wxWindowID** id, **const wxPoint&** pos = *wxDefaultPosition*, **const wxSize&** size = *wxDefaultSize*, **long** style = 0, **const wxString&** name = *wxPanelNameStr*)

Constructs a window, which can be a child of a frame, dialog or any other non-control window.

### Parameters

*parent*

Pointer to a parent window.

*id*

Window identifier. If *wxID\_ANY*, will automatically create an identifier.

*pos*

Window position. *wxDefaultPosition* indicates that *wxWidgets* should generate a default position for the window. If using the *wxWindow* class directly, supply an actual position.

*size*

Window size. *wxDefaultSize* indicates that *wxWidgets* should generate a default size for the window. If no suitable size can be found, the window will be sized to 20x20 pixels so that the window is visible but obviously not correctly sized.

*style*

Window style. For generic window styles, please see *wxWindow* (p. 1795).

*name*

Window name.

## **wxWindow::~~wxWindow**

### **~wxWindow()**

Destructor. Deletes all subwindows, then deletes itself. Instead of using the **delete** operator explicitly, you should normally use *wxWindow::Destroy* (p. 1804) so that *wxWidgets* can delete a window only when it is safe to do so, in idle time.

### **See also**

*Window deletion overview* (p. 2089), *wxWindow::Destroy* (p. 1804), *wxCloseEvent* (p. 200)

## **wxWindow::AddChild**

### **virtual void AddChild(wxWindow\* child)**

Adds a child window. This is called automatically by window creation functions so should not be required by the application programmer.

Notice that this function is mostly internal to *wxWidgets* and shouldn't be called by the user code.

### **Parameters**

*child*

Child window to add.

## **wxWindow::CacheBestSize**

### **void CacheBestSize(const wxSize& size) const**

Sets the cached best size value.

## **wxWindow::CaptureMouse**

### **virtual void CaptureMouse()**

Directs all mouse input to this window. Call *wxWindow::ReleaseMouse* (p. 1831) to release the capture.

Note that *wxWidgets* maintains the stack of windows having captured the mouse and when the mouse is released the capture returns to the window which had had captured it previously and it is only really released if there were no previous window. In particular, this means that you must release the mouse as many times as you capture it, unless the window receives the *wxMouseCaptureLostEvent* (p. 1119) event.

Any application which captures the mouse in the beginning of some operation *must* handle *wxMouseCaptureLostEvent* (p. 1119) and cancel this operation when it receives the event. The event handler must not recapture mouse.

**See also**

*wxWindow::ReleaseMouse* (p. 1831) *wxMouseCaptureLostEvent* (p. 1119)

**wxWindow::Center**

**void Center**(int *direction*)

A synonym for *Centre* (p. 1800).

**wxWindow::CenterOnParent**

**void CenterOnParent**(int *direction*)

A synonym for *CentreOnParent* (p. 1801).

**wxWindow::CenterOnScreen**

**void CenterOnScreen**(int *direction*)

A synonym for *CentreOnScreen* (p. 1801).

**wxWindow::Centre**

**void Centre**(int *direction* = *wxBOTH*)

Centres the window.

**Parameters**

*direction*

Specifies the direction for the centering. May be *wxHORIZONTAL*, *wxVERTICAL* or *wxBOTH*. It may also include *wxCENTRE\_ON\_SCREEN* flag if you want to center the window on the entire screen and not on its parent window.

The flag *wxCENTRE\_FRAME* is obsolete and should not be used any longer (it has no effect).

**Remarks**

If the window is a top level one (i.e. doesn't have a parent), it will be centered relative to the screen anyhow.

**See also**

*wxWindow::Center* (p. 1800)



**wxWindow::CentreOnParent****void CentreOnParent**(int *direction* = *wxBOTH*)

Centres the window on its parent. This is a more readable synonym for *Centre* (p. 1800).

**Parameters***direction*

Specifies the direction for the centering. May be *wxHORIZONTAL*, *wxVERTICAL* or *wxBOTH*.

**Remarks**

This methods provides for a way to center top level windows over their parents instead of the entire screen. If there is no parent or if the window is not a top level window, then behaviour is the same as *wxWindow::Centre* (p. 1800).

**See also***wxWindow::CentreOnScreen* (p. 1800)**wxWindow::CentreOnScreen****void CentreOnScreen**(int *direction* = *wxBOTH*)

Centres the window on screen. This only works for top level windows - otherwise, the window will still be centered on its parent.

**Parameters***direction*

Specifies the direction for the centering. May be *wxHORIZONTAL*, *wxVERTICAL* or *wxBOTH*.

**See also***wxWindow::CentreOnParent* (p. 1800)**wxWindow::ClearBackground****void ClearBackground**()

Clears the window by filling it with the current background colour. Does not cause an erase background event to be generated.

**wxWindow::ClientToScreen****virtual void ClientToScreen**(int\* *x*, int\* *y*) **const**

**wxPerl note:** In wxPerl this method returns a 2-element list instead of modifying its

parameters.

**virtual wxPoint ClientToScreen(const wxPoint& pt) const**

Converts to screen coordinates from coordinates relative to this window.

*x*

A pointer to a integer value for the x coordinate. Pass the client coordinate in, and a screen coordinate will be passed out.

*y*

A pointer to a integer value for the y coordinate. Pass the client coordinate in, and a screen coordinate will be passed out.

*pt*

The client position for the second form of the function.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

<b>ClientToScreen(point)</b>	Accepts and returns a wxPoint
<b>ClientToScreenXY(x, y)</b>	Returns a 2-tuple, (x, y)

## **wxWindow::ClientToWindowSize**

**wxSize ClientToWindowSize(const wxSize& size)**

Converts client area size *size* to corresponding window size. In other words, the returned value is what would *GetSize* (p. 1818) return if this window had client area of given size. Components with wxDefaultCoord value are left unchanged.

Note that the conversion is not always exact, it assumes that non-client area doesn't change and so doesn't take into account things like menu bar (un)wrapping or (dis)appearance of the scrollbars.

This function is new since wxWidgets version 2.8.8

### **See also**

*wxWindow::WindowToClientSize* (p. 1853)

## **wxWindow::Close**

**bool Close(bool force = false)**

This function simply generates a *wxCloseEvent* (p. 200) whose handler usually tries to close the window. It doesn't close the window itself, however.

### **Parameters**

*force*

`false` if the window's close handler should be able to veto the destruction of this window, `true` if it cannot.

### Remarks

`Close` calls the *close handler* (p. 200) for the window, providing an opportunity for the window to choose whether to destroy the window. Usually it is only used with the top level windows (`wxFrame` and `wxDIALOG` classes) as the others are not supposed to have any special `OnClose()` logic.

The close handler should check whether the window is being deleted forcibly, using `wxCloseEvent::CanVeto` (p. 201), in which case it should destroy the window using `wxWindow::Destroy` (p. 1804).

*Note* that calling `Close` does not guarantee that the window will be destroyed; but it provides a way to simulate a manual close of a window, which may or may not be implemented by destroying the window. The default implementation of `wxDIALOG::OnCloseWindow` does not necessarily delete the dialog, since it will simply simulate an `wxID_CANCEL` event which is handled by the appropriate button event handler and may do anything at all.

To guarantee that the window will be destroyed, call `wxWindow::Destroy` (p. 1804) instead

### See also

*Window deletion overview* (p. 2089), `wxWindow::Destroy` (p. 1804), `wxCloseEvent` (p. 200)

## **wxWindow::ConvertDialogToPixels**

**wxPoint** `ConvertDialogToPixels(const wxPoint& pt)`

**wxSize** `ConvertDialogToPixels(const wxSize& sz)`

Converts a point or size from dialog units to pixels.

For the x dimension, the dialog units are multiplied by the average character width and then divided by 4.

For the y dimension, the dialog units are multiplied by the average character height and then divided by 8.

### Remarks

Dialog units are used for maintaining a dialog's proportions even if the font changes.

You can also use these functions programmatically. A convenience macro is defined:

```
#define wxDLG_UNIT(parent, pt) parent->ConvertDialogToPixels(pt)
```

**See also**

*wxWindow::ConvertPixelsToDialog* (p. 1804)

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

**ConvertDialogPointToPixels(point)**    Accepts and returns a wxPoint

**ConvertDialogSizeToPixels(size)**    Accepts and returns a wxSize

Additionally, the following helper functions are defined:

**wxDLG\_PNT(win, point)**    Converts a wxPoint from dialog units to pixels

**wxDLG\_SIZE(win, size)**    Converts a wxSize from dialog units to pixels

**wxWindow::ConvertPixelsToDialog**

**wxPoint ConvertPixelsToDialog(const wxPoint& pt)**

**wxSize ConvertPixelsToDialog(const wxSize& sz)**

Converts a point or size from pixels to dialog units.

For the x dimension, the pixels are multiplied by 4 and then divided by the average character width.

For the y dimension, the pixels are multiplied by 8 and then divided by the average character height.

**Remarks**

Dialog units are used for maintaining a dialog's proportions even if the font changes.

**See also**

*wxWindow::ConvertDialogToPixels* (p. 1803)

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

**ConvertDialogPointToPixels(point)**    Accepts and returns a wxPoint

**ConvertDialogSizeToPixels(size)**    Accepts and returns a wxSize

**wxWindow::Destroy**

**virtual bool Destroy()**

Destroys the window safely. Use this function instead of the delete operator, since

different window classes can be destroyed differently. Frames and dialogs are not destroyed immediately when this function is called -- they are added to a list of windows to be deleted on idle time, when all the window's events have been processed. This prevents problems with events being sent to non-existent windows.

**Return value**

`true` if the window has either been successfully deleted, or it has been added to the list of windows pending real deletion.

**`wxWindow::DestroyChildren`****`virtual void DestroyChildren()`**

Destroys all children of a window. Called automatically by the destructor.

**`wxWindow::Disable`****`bool Disable()`**

Disables the window, same as *Enable(false)* (p. 1806).

**Return value**

Returns `true` if the window has been disabled, `false` if it had been already disabled before the call to this function.

**`wxWindow::DoGetBestSize`****`virtual wxSize DoGetBestSize() const`**

Gets the size which best suits the window: for a control, it would be the minimal size which doesn't truncate the control, for a panel - the same size as it would have after a call to *Fit()* (p. 1808).

**`wxWindow::DoUpdateWindowUI`****`virtual void DoUpdateWindowUI(wxUpdateUIEvent& event)`**

Does the window-specific updating after processing the update event. This function is called by *wxWindow::UpdateWindowUI* (p. 1852) in order to check return values in the *wxUpdateUIEvent* (p. 1752) and act appropriately. For example, to allow frame and dialog title updating, *wxWidgets* implements this function as follows:

```
// do the window-specific processing after processing the update event
void wxTopLevelWindowBase::DoUpdateWindowUI(wxUpdateUIEvent& event)
{
    if ( event.GetSetEnabled() )
        Enable(event.GetEnabled());

    if ( event.GetSetText() )
    {
```

```
        if ( event.GetText() != GetTitle() )  
            SetTitle(event.GetText());  
    }  
}
```

### **wxWindow::DragAcceptFiles**

**virtual void DragAcceptFiles(bool accept)**

Enables or disables eligibility for drop file events (OnDropFiles).

#### **Parameters**

*accept*

If `true`, the window is eligible for drop file events. If `false`, the window will not accept drop file events.

#### **Remarks**

Windows only.

### **wxWindow::Enable**

**virtual bool Enable(bool enable = true)**

Enable or disable the window for user input. Note that when a parent window is disabled, all of its children are disabled as well and they are reenabled again when the parent is.

#### **Parameters**

*enable*

If `true`, enables the window for input. If `false`, disables the window.

#### **Return value**

Returns `true` if the window has been enabled or disabled, `false` if nothing was done, i.e. if the window had already been in the specified state.

#### **See also**

*wxWindow::IsEnabled* (p. 1823), *wxWindow::Disable* (p. 1805), *wxRadioBox::Enable* (p. 1243)

### **wxWindow::FindFocus**

**static wxWindow\* FindFocus()**

Finds the window or control which currently has the keyboard focus.

**Remarks**

Note that this is a static function, so it can be called without needing a `wxWindow` pointer.

**See also**

`wxWindow::SetFocus` (p. 1839)

**wxWindow::FindWindow**

**wxWindow\* FindWindow(long id) const**

Find a child of this window, by identifier.

**wxWindow\* FindWindow(const wxString& name) const**

Find a child of this window, by name.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

**FindWindowById(id)**      Accepts an integer

**FindWindowByName(name)**    Accepts a string

**wxWindow::FindWindowById**

**static wxWindow\* FindWindowById(long id, wxWindow\* parent = NULL)**

Find the first window with the given *id*.

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.

**See also**

`FindWindow` (p. 1807)

**wxWindow::FindWindowByLabel**

**static wxWindow\* FindWindowByLabel(const wxString& label, wxWindow\* parent = NULL)**

Find a window by its label. Depending on the type of window, the label may be a window title or panel item label. If *parent* is `NULL`, the search will start from all top-level frames and dialog boxes; if non-`NULL`, the search will be limited to the given window hierarchy. The search is recursive in both cases.

**See also**

`FindWindow` (p. 1807)

**wxWindow::FindWindowByName**

**static wxWindow\* FindWindowByName(const wxString& name, wxWindow\* parent = NULL)**

Find a window by its name (as given in a window constructor or **Create** function call). If *parent* is NULL, the search will start from all top-level frames and dialog boxes; if non-NULL, the search will be limited to the given window hierarchy. The search is recursive in both cases.

If no window with such name is found, *FindWindowByLabel* (p. 1807) is called.

**See also**

*FindWindow* (p. 1807)

**wxWindow::Fit**

**virtual void Fit()**

Sizes the window so that it fits around its subwindows. This function won't do anything if there are no subwindows and will only really work correctly if the sizers are used for the subwindows layout. Also, if the window has exactly one subwindow it is better (faster and the result is more precise as *Fit* adds some margin to account for fuzziness of its calculations) to call

```
window->SetClientSize(child->GetSize());
```

instead of calling *Fit*.

**wxWindow::FitInside**

**virtual void FitInside()**

Similar to *Fit* (p. 1808), but sizes the interior (virtual) size of a window. Mainly useful with scrolled windows to reset scrollbars after sizing changes that do not trigger a size event, and/or scrolled windows without an interior sizer. This function similarly won't do anything if there are no subwindows.

**wxWindow::Freeze**

**virtual void Freeze()**

Freezes the window or, in other words, prevents any updates from taking place on screen, the window is not redrawn at all. *Thaw* (p. 1850) must be called to reenables window redrawing. Calls to these two functions may be nested.

This method is useful for visual appearance optimization (for example, it is a good idea to use it before doing many large text insertions in a row into a *wxTextCtrl* under *wxGTK*) but is not implemented on all platforms nor for all controls so it is mostly just a hint to *wxWidgets* and not a mandatory directive.



**See also**

*wxWindowUpdateLocker* (p. 1854)

**wxWindow::GetAcceleratorTable**

**wxAcceleratorTable\* GetAcceleratorTable() const**

Gets the accelerator table for this window. See *wxAcceleratorTable* (p. 23).

**wxWindow::GetAccessible**

**wxAccessible\* GetAccessible()**

Returns the accessible object for this window, if any.

See also *wxAccessible* (p. 25).

**wxWindow::GetAdjustedBestSize**

**wxSize GetAdjustedBestSize() const**

This method is deprecated, use *GetEffectiveMinSize* (p. 1810) instead.

**wxWindow::GetBackgroundColour**

**virtual wxColour GetBackgroundColour() const**

Returns the background colour of the window.

**See also**

*wxWindow::SetBackgroundColour* (p. 1835), *wxWindow::SetForegroundColour* (p. 1840), *wxWindow::GetForegroundColour* (p. 1813)

**wxWindow::GetBackgroundStyle**

**virtual wxBackgroundStyle GetBackgroundStyle() const**

Returns the background style of the window. The background style indicates whether background colour should be determined by the system (*wxBG\_STYLE\_SYSTEM*), be set to a specific colour (*wxBG\_STYLE\_COLOUR*), or should be left to the application to implement (*wxBG\_STYLE\_CUSTOM*).

On GTK+, use of *wxBG\_STYLE\_CUSTOM* allows the flicker-free drawing of a custom background, such as a tiled bitmap. Currently the style has no effect on other platforms.

**See also**

*wxWindow::SetBackgroundColour* (p. 1835), *wxWindow::GetForegroundColour* (p. 1813), *wxWindow::SetBackgroundStyle* (p. 1836)

**wxWindow::GetEffectiveMinSize****wxSize GetEffectiveMinSize() const**

Merges the window's best size into the min size and returns the result. This is the value used by sizers to determine the appropriate ammount of sapce to allocate for the widget.

**See also**

*wxWindow::GetBestSize* (p. 1810), *wxWindow::SetInitialSize* (p. 1836)

**wxWindow::GetBestSize****wxSize GetBestSize() const**

This functions returns the best acceptable minimal size for the window. For example, for a static control, it will be the minimal size such that the control label is not truncated. For windows containing subwindows (typically *wxPanel* (p. 1170)), the size returned by this function will be the same as the size the window would have had after calling *Fit* (p. 1808).

**wxWindow::GetCapture****static wxWindow \* GetCapture()**

Returns the currently captured window.

**See also**

*wxWindow::HasCapture* (p. 1821), *wxWindow::CaptureMouse* (p. 1799), *wxWindow::ReleaseMouse* (p. 1831), *wxMouseCaptureLostEvent* (p. 1119) *wxMouseCaptureChangedEvent* (p. 1118)

**wxWindow::GetCaret****wxCaret \* GetCaret() const**

Returns the *caret* (p. 177) associated with the window.

**wxWindow::GetCharHeight****virtual int GetCharHeight() const**

Returns the character height for this window.

**wxWindow::GetCharWidth****virtual int GetCharWidth() const**

Returns the average character width for this window.

**wxWindow::GetChildren**

**wxWindowList& GetChildren()****const wxWindowList& GetChildren() const**

Returns a reference to the list of the window's children. `wxWindowList` is a type-safe `wxList` (p. 966)-like class whose elements are of type `wxWindow *`.

**wxWindow::GetClassDefaultAttributes**

**static wxVisualAttributes GetClassDefaultAttributes(wxWindowVariant variant = `wxWINDOW_VARIANT_NORMAL`)**

Returns the default font and colours which are used by the control. This is useful if you want to use the same font or colour in your own control as in a standard control -- which is a much better idea than hard coding specific colours or fonts which might look completely out of place on the users system, especially if it uses themes.

The *variant* parameter is only relevant under Mac currently and is ignore under other platforms. Under Mac, it will change the size of the returned font. See `wxWindow::SetWindowVariant` (p. 1849) for more about this.

This static method is "overridden" in many derived classes and so calling, for example, `wxButton` (p. 164)::`GetClassDefaultAttributes()` will typically return the values appropriate for a button which will be normally different from those returned by, say, `wxListCtrl` (p. 980)::`GetClassDefaultAttributes()`.

The `wxVisualAttributes` structure has at least the fields `font`, `colFg` and `colBg`. All of them may be invalid if it was not possible to determine the default control appearance or, especially for the background colour, if the field doesn't make sense as is the case for `colBg` for the controls with themed background.

**See also**

*InheritAttributes* (p. 1822)

**wxWindow::GetClientSize**

**void GetClientSize(int\* width, int\* height) const**

**wxPerl note:** In wxPerl this method takes no parameter and returns a 2-element list (`width`, `height`).

**wxSize GetClientSize() const**

This gets the size of the window 'client area' in pixels. The client area is the area which may be drawn on by the programmer, excluding title bar, border, scrollbars, etc.

**Parameters**

*width*

Receives the client width in pixels.

*height*

Receives the client height in pixels.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

<b>GetClientSizeTuple()</b>	Returns a 2-tuple of (width, height)
<b>GetClientSize()</b>	Returns a wxSize object

**See also**

*GetSize* (p. 1818), *GetVirtualSize* (p. 1821)

**wxWindow::GetConstraints**

**wxLayoutConstraints\* GetConstraints() const**

Returns a pointer to the window's layout constraints, or NULL if there are none.

**wxWindow::GetContainingSizer**

**const wxSizer \* GetContainingSizer() const**

Return the sizer that this window is a member of, if any, otherwise NULL.

**wxWindow::GetCursor**

**const wxCursor& GetCursor() const**

Return the cursor associated with this window.

**See also**

*wxWindow::SetCursor* (p. 1837)

**wxWindow::GetDefaultAttributes**

**virtual wxVisualAttributes GetDefaultAttributes() const**

Currently this is the same as calling *GetClassDefaultAttributes* (p. 1811)(*GetWindowVariant* (p. 1821)()).

One advantage of using this function compared to the static version is that the call is automatically dispatched to the correct class (as usual with virtual functions) and you don't have to specify the class name explicitly.

The other one is that in the future this function could return different results, for example it might return a different font for an "Ok" button than for a generic button if the users GUI is configured to show such buttons in bold font. Of course, the down side is that it is

impossible to call this function without actually having an object to apply it to whereas the static version can be used without having to create an object first.

### **wxWindow::GetDropTarget**

**wxDropTarget\* GetDropTarget() const**

Returns the associated drop target, which may be NULL.

#### **See also**

*wxWindow::SetDropTarget* (p. 1838), *Drag and drop overview* (p. 2150)

### **wxWindow::GetEventHandler**

**wxEvtHandler\* GetEventHandler() const**

Returns the event handler for this window. By default, the window is its own event handler.

#### **See also**

*wxWindow::SetEventHandler* (p. 1838), *wxWindow::PushEventHandler* (p. 1829),  
*wxWindow::PopEventHandler* (p. 1829), *wxEvtHandler::ProcessEvent* (p. 580),  
*wxEvtHandler* (p. 576)

### **wxWindow::GetExtraStyle**

**long GetExtraStyle() const**

Returns the extra style bits for the window.

### **wxWindow::GetFont**

**wxFont GetFont() const**

Returns the font for this window.

#### **See also**

*wxWindow::SetFont* (p. 1840)

### **wxWindow::GetForegroundColour**

**virtual wxColour GetForegroundColour()**

Returns the foreground colour of the window.

#### **Remarks**

The interpretation of foreground colour is open to interpretation according to the window class; it may be the text colour or other colour, or it may not be used at all.

**See also**

*wxWindow::SetForegroundColour* (p. 1840), *wxWindow::SetBackgroundColour* (p. 1835), *wxWindow::GetBackgroundColour* (p. 1809)

**wxWindow::GetGrandParent**

**wxWindow\* GetGrandParent() const**

Returns the grandparent of a window, or NULL if there isn't one.

**wxWindow::GetHandle**

**void\* GetHandle() const**

Returns the platform-specific handle of the physical window. Cast it to an appropriate handle, such as **HWND** for Windows, **Widget** for Motif, **GtkWidget** for GTK or **WinHandle** for PalmOS.

**wxPython note:** This method will return an integer in wxPython.

**wxPerl note:** This method will return an integer in wxPerl.

**wxWindow::GetHelpTextAtPoint**

**virtual wxString GetHelpTextAtPoint(const wxPoint&point, wxHelpEvent::Origin origin) const**

Gets the help text to be used as context-sensitive help for this window. This method should be overridden if the help message depends on the position inside the window, otherwise *GetHelpText* (p. 1814) can be used.

**Parameters**

*point*

Coordinates of the mouse at the moment of help event emission.

*origin*

Help event origin, see also *wxHelpEvent::GetOrigin* (p. 816).

This function is new since wxWidgets version 2.7.0

**wxWindow::GetHelpText**

**virtual wxString GetHelpText() const**

Gets the help text to be used as context-sensitive help for this window.

Note that the text is actually stored by the current *wxHelpProvider* (p. 817) implementation, and not in the window object itself.

**See also**

*SetHelpText* (p. 1841), *GetHelpTextAtPoint* (p. 1814), *wxHelpProvider* (p. 817)

**wxWindow::GetId**

**int GetId() const**

Returns the identifier of the window.

**Remarks**

Each window has an integer identifier. If the application has not provided one (or the default `wxID_ANY`) a unique identifier with a negative value will be generated.

**See also**

*wxWindow::SetId* (p. 1841), *Window identifiers* (p. 2082)

**wxWindow::GetLabel**

**virtual wxString GetLabel() const**

Generic way of getting a label from any window, for identification purposes.

**Remarks**

The interpretation of this function differs from class to class. For frames and dialogs, the value returned is the title. For buttons or static text controls, it is the button text. This function can be useful for meta-programs (such as testing tools or special-needs access programs) which need to identify windows by name.

**wxWindow::GetMaxSize**

**wxSize GetMaxSize() const**

Returns the maximum size of the window, an indication to the sizer layout mechanism that this is the maximum possible size.

**wxWindow::GetMinSize**

**virtual wxSize GetMinSize() const**

Returns the minimum size of the window, an indication to the sizer layout mechanism that this is the minimum required size. It normally just returns the value set by *SetMinSize* (p. 1842), but it can be overridden to do the calculation on demand.

**wxWindow::GetName**

**virtual wxString GetName() const**

Returns the window's name.

**Remarks**

This name is not guaranteed to be unique; it is up to the programmer to supply an appropriate name in the window constructor or via *wxWindow::SetName* (p. 1842).

**See also**

*wxWindow::SetName* (p. 1842)

**wxWindow::GetNextSibling**

**wxWindow \* GetNextSibling() const**

Returns the next window after this one among the parent children or `NULL` if this window is the last child.

This function is new since wxWidgets version 2.8.8

**See also**

*GetPrevSibling* (p. 1817)

**wxWindow::GetParent**

**virtual wxWindow\* GetParent() const**

Returns the parent of the window, or `NULL` if there is no parent.

**wxWindow::GetPosition**

**virtual void GetPosition(int\* x, int\* y) const**

**wxPoint GetPosition() const**

This gets the position of the window in pixels, relative to the parent window for the child windows or relative to the display origin for the top level windows.

**Parameters**

*x*

Receives the x position of the window if non-`NULL`.

*y*

Receives the y position of the window if non-`NULL`.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

<b>GetPosition()</b>	Returns a wxPoint
<b>GetPositionTuple()</b>	Returns a tuple (x, y)



**wxPerl note:** In wxPerl there are two methods instead of a single overloaded method:

<b>GetPosition()</b>	Returns a <code>Wx::Point</code>
<b>GetPositionXY()</b>	Returns a 2-element list ( <code>x</code> , <code>y</code> )

**See also**

*GetScreenPosition* (p. 1817)

**wxWindow::GetPrevSibling**

**wxWindow \* GetPrevSibling() const**

Returns the previous window before this one among the parent children or `NULL` if this window is the first child.

This function is new since wxWidgets version 2.8.8

**See also**

*GetNextSibling* (p. 1816)

**wxWindow::GetRect**

**virtual wxRect GetRect() const**

Returns the size and position of the window as a *wxRect* (p. 1252) object.

**See also**

*GetScreenRect* (p. 1818)

**wxWindow::GetScreenPosition**

**virtual void GetScreenPosition(int\* x, int\* y) const**

**wxPoint GetScreenPosition() const**

Returns the window position in screen coordinates, whether the window is a child window or a top level one.

**Parameters**

*x*

Receives the x position of the window on the screen if non-`NULL`.

*y*

Receives the y position of the window on the screen if non-`NULL`.

**See also**

*GetPosition* (p. 1816)

**wxWindow::GetScreenRect**

**virtual wxRect GetScreenRect() const**

Returns the size and position of the window on the screen as a *wxRect* (p. 1252) object.

**See also**

*GetRect* (p. 1817)

**wxWindow::GetScrollPos**

**virtual int GetScrollPos(int orientation)**

Returns the built-in scrollbar position.

**See also**

See *wxWindow::SetScrollbar* (p. 1843)

**wxWindow::GetScrollRange**

**virtual int GetScrollRange(int orientation)**

Returns the built-in scrollbar range.

**See also**

*wxWindow::SetScrollbar* (p. 1843)

**wxWindow::GetScrollThumb**

**virtual int GetScrollThumb(int orientation)**

Returns the built-in scrollbar thumb size.

**See also**

*wxWindow::SetScrollbar* (p. 1843)

**wxWindow::GetSize**

**void GetSize(int\* width, int\* height) const**

**wxSize GetSize() const**

This gets the size of the entire window in pixels, including title bar, border, scrollbars, etc.

**Parameters**

*width*

Receives the window width.

*height*

Receives the window height.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

<b>GetSize()</b>	Returns a wxSize
<b>GetSizeTuple()</b>	Returns a 2-tuple (width, height)

**wxPerl note:** In wxPerl there are two methods instead of a single overloaded method:

<b>GetSize()</b>	Returns a Wx::Size
<b>GetSizeWH()</b>	Returns a 2-element list ( width, height )

#### See also

*GetClientSize* (p. 1811), *GetVirtualSize* (p. 1821)

#### wxWindow::GetSizer

**wxSizer \* GetSizer() const**

Return the sizer associated with the window by a previous call to *SetSizer()* (p. 1846) or NULL.

#### wxWindow::GetTextExtent

**virtual void GetTextExtent(const wxString& string, int\* x, int\* y, int\* descent = NULL, int\* externalLeading = NULL, const wxFont\* font = NULL, bool use16 = false) const**

Gets the dimensions of the string as it would be drawn on the window with the currently selected font.

#### Parameters

*string*

String whose extent is to be measured.

*x*

Return value for width.

*y*

Return value for height.

*descent*

Return value for descent (optional).

*externalLeading*

Return value for external leading (optional).

*font*

Font to use instead of the current window font (optional).

*use16*

If `true`, *string* contains 16-bit characters. The default is `false`.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

**GetTextExtent(string)** Returns a 2-tuple, (width, height)

**GetFullTextExtent(string, font=NULL)** Returns a 4-tuple, (width, height, descent, externalLeading)

**wxPerl note:** In wxPerl this method takes only the *string* and optionally *font* parameters, and returns a 4-element list ( *x*, *y*, *descent*, *externalLeading* ).

### **wxWindow::GetToolTip**

**wxToolTip\* GetToolTip() const**

Get the associated tooltip or NULL if none.

### **wxWindow::GetUpdateRegion**

**virtual wxRegion GetUpdateRegion() const**

Returns the region specifying which parts of the window have been damaged. Should only be called within an *wxPaintEvent* (p. 1164) handler.

### **See also**

*wxRegion* (p. 1264), *wxRegionIterator* (p. 1269)

### **wxWindow::GetValidator**

**wxValidator\* GetValidator() const**

Returns a pointer to the current validator for the window, or NULL if there is none.

**wxWindow::GetVirtualSize****void GetVirtualSize(int\* width, int\* height) const****wxSize GetVirtualSize() const**

This gets the virtual size of the window in pixels. By default it returns the client size of the window, but after a call to *SetVirtualSize* (p. 1848) it will return that size.

**Parameters***width*

Receives the window virtual width.

*height*

Receives the window virtual height.

*GetSize* (p. 1818), *GetClientSize* (p. 1811)

**wxWindow::GetWindowBorderSize****wxSize GetWindowBorderSize() const**

Returns the size of the left/right and top/bottom borders of this window in x and y components of the result respectively.

**wxWindow::GetWindowStyleFlag****long GetWindowStyleFlag() const**

Gets the window style that was passed to the constructor or **Create** method.

**GetWindowStyle()** is another name for the same function.

**wxWindow::GetWindowVariant****wxWindowVariant GetWindowVariant() const**

Returns the value previously passed to *wxWindow::SetWindowVariant* (p. 1849).

**wxWindow::HasCapture****virtual bool HasCapture() const**

Returns `true` if this window has the current mouse capture.

**See also**

*wxWindow::CaptureMouse* (p. 1799), *wxWindow::ReleaseMouse* (p. 1831), *wxMouseCaptureLostEvent* (p. 1119) *wxMouseCaptureChangedEvent* (p. 1118)

**wxWindow::HasFlag****bool HasFlag(int *flag*) const**

Returns `true` if the window has the given *flag* bit set.

**wxWindow::HasMultiplePages****virtual bool HasMultiplePages() const**

This method should be overridden to return `true` if this window has multiple pages. All standard class with multiple pages such as *wxNotebook* (p. 1135), *wxListbook* (p. 974) and *wxTreebook* (p. 1721) already override it to return `true` and user-defined classes with similar behaviour should do it as well to allow the library to handle such windows appropriately.

**wxWindow::HasScrollbar****virtual bool HasScrollbar(int *orient*) const**

Returns `true` if this window has a scroll bar for this orientation.

**Parameters**

*orient*

Orientation to check, either `wxHORIZONTAL` or `wxVERTICAL`.

**wxWindow::HasTransparentBackground****virtual bool HasTransparentBackground() const**

Returns `true` if this window background is transparent (as, for example, for *wxStaticText*) and should show the parent window background.

This method is mostly used internally by the library itself and you normally shouldn't have to call it. You may, however, have to override it in your *wxWindow*-derived class to ensure that background is painted correctly.

**wxWindow::Hide****bool Hide()**

Equivalent to calling *Show* (p. 1850)(`false`).

**wxWindow::InheritAttributes****void InheritAttributes()**

This function is (or should be, in case of custom controls) called during window creation to intelligently set up the window visual attributes, that is the font and the foreground and

background colours.

By "intelligently" the following is meant: by default, all windows use their own *default* (p. 1811) attributes. However if some of the parents attributes are explicitly (that is, using *SetFont* (p. 1840) and not *SetOwnFont* (p. 1843)) changed *and* if the corresponding attribute hadn't been explicitly set for this window itself, then this window takes the same value as used by the parent. In addition, if the window overrides *ShouldInheritColours* (p. 1849) to return `false`, the colours will not be changed no matter what and only the font might.

This rather complicated logic is necessary in order to accommodate the different usage scenarios. The most common one is when all default attributes are used and in this case, nothing should be inherited as in modern GUIs different controls use different fonts (and colours) than their siblings so they can't inherit the same value from the parent. However it was also deemed desirable to allow to simply change the attributes of all children at once by just changing the font or colour of their common parent, hence in this case we do inherit the parents attributes.

### **wxWindow::InitDialog**

#### **void InitDialog()**

Sends an `wxEVT_INIT_DIALOG` event, whose handler usually transfers data to the dialog via validators.

### **wxWindow::InvalidateBestSize**

#### **void InvalidateBestSize()**

Resets the cached best size value so it will be recalculated the next time it is needed.

### **wxWindow::IsDoubleBuffered**

#### **virtual bool IsDoubleBuffered() const**

Returns `true` if the window contents is double-buffered by the system, i.e. if any drawing done on the window is really done on a temporary backing surface and transferred to the screen all at once later.

#### **See also**

*wxBufferedDC* (p. 157)

### **wxWindow::IsEnabled**

#### **virtual bool IsEnabled() const**

Returns `true` if the window is enabled for input, `false` otherwise.

#### **See also**

*wxWindow::Enable* (p. 1806)

### **wxWindow::IsExposed**

**bool IsExposed(int x, int y) const**

**bool IsExposed(wxPoint &pt) const**

**bool IsExposed(int x, int y, int w, int h) const**

**bool IsExposed(wxRect &rect) const**

Returns `true` if the given point or rectangle area has been exposed since the last repaint. Call this in an paint event handler to optimize redrawing by only redrawing those areas, which have been exposed.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

**IsExposed(x,y, w=0,h=0)**

**IsExposedPoint(pt)**

**IsExposedRect(rect)**

### **wxWindow::IsFrozen**

**virtual bool IsFrozen() const**

Returns `true` if the window is currently frozen by a call to *Freeze()* (p. 1808).

#### **See also**

*Thaw()* (p. 1850)

### **wxWindow::IsRetained**

**virtual bool IsRetained() const**

Returns `true` if the window is retained, `false` otherwise.

#### **Remarks**

Retained windows are only available on X platforms.

### **wxWindow::IsShown**

**virtual bool IsShown() const**

Returns `true` if the window is shown, `false` if it has been hidden.

#### **See also**



*wxWindow::IsShownOnScreen* (p. 1825)

### **wxWindow::IsShownOnScreen**

**virtual bool IsShownOnScreen() const**

Returns `true` if the window is physically visible on the screen, i.e. it is shown and all its parents up to the toplevel window are shown as well.

#### **See also**

*wxWindow::IsShown* (p. 1824)

### **wxWindow::IsTopLevel**

**bool IsTopLevel() const**

Returns `true` if the given window is a top-level one. Currently all frames and dialogs are considered to be top-level windows (even if they have a parent window).

### **wxWindow::Layout**

**void Layout()**

Invokes the constraint-based layout algorithm or the sizer-based algorithm for this window.

See *wxWindow::SetAutoLayout* (p. 1834): when auto layout is on, this function gets called automatically when the window is resized.

### **wxWindow::LineDown**

This is just a wrapper for *ScrollLines* (p. 1833)(1).

### **wxWindow::LineUp**

This is just a wrapper for *ScrollLines* (p. 1833)(-1).

### **wxWindow::Lower**

**void Lower()**

Lowers the window to the bottom of the window hierarchy (z-order).

#### **See also**

*Raise* (p. 1830)

### **wxWindow::MakeModal**

**virtual void MakeModal(*bool flag*)**

Disables all other windows in the application so that the user can only interact with this window.

**Parameters**

*flag*

If `true`, this call disables all other windows in the application so that the user can only interact with this window. If `false`, the effect is reversed.

**wxWindow::Move**

**void Move(int *x*, int *y*)**

**void Move(const wxPoint& *pt*)**

Moves the window to the given position.

**Parameters**

*x*

Required x position.

*y*

Required y position.

*pt*

*wxPoint* (p. 1193) object representing the position.

**Remarks**

Implementations of `SetSize` can also implicitly implement the `wxWindow::Move` function, which is defined in the base `wxWindow` class as the call:

```
SetSize(x, y, wxDefaultCoord, wxDefaultCoord,  
wxSIZE_USE_EXISTING);
```

**See also**

`wxWindow::SetSize` (p. 1845)

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

<b>Move(<i>point</i>)</b>	Accepts a <code>wxPoint</code>
<b>MoveXY(<i>x</i>, <i>y</i>)</b>	Accepts a pair of integers

**wxWindow::MoveAfterInTabOrder****void MoveAfterInTabOrder(wxWindow \*win)**

Moves this window in the tab navigation order after the specified *win*. This means that when the user presses `TAB` key on that other window, the focus switches to this window.

Default tab order is the same as creation order, this function and *MoveBeforeInTabOrder()* (p. 1827) allow to change it after creating all the windows.

**Parameters***win*

A sibling of this window which should precede it in tab order, must not be `NULL`

**wxWindow::MoveBeforeInTabOrder****void MoveBeforeInTabOrder(wxWindow \*win)**

Same as *MoveAfterInTabOrder* (p. 1827) except that it inserts this window just before *win* instead of putting it right after it.

**wxWindow::Navigate****bool Navigate(int flags = wxNavigationKeyEvent::IsForward)**

Does keyboard navigation from this window to another, by sending a `wxNavigationKeyEvent`.

**Parameters***flags*

A combination of `wxNavigationKeyEvent::IsForward` and `wxNavigationKeyEvent::WinChange`.

**Remarks**

You may wish to call this from a text control custom keypress handler to do the default navigation behaviour for the tab key, since the standard default behaviour for a multiline text control with the `wxTE_PROCESS_TAB` style is to insert a tab and not navigate to the next control.

**wxWindow::OnInternalIdle****virtual void OnInternalIdle()**

This virtual function is normally only used internally, but sometimes an application may need it to implement functionality that should not be disabled by an application defining an `OnIdle` handler in a derived class.

This function may be used to do delayed painting, for example, and most implementations

call `wxWindow::UpdateWindowUI` (p. 1852) in order to send update events to the window in idle time.

### **wxWindow::PageDown**

This is just a wrapper for `ScrollPages()` (p. 1833)(1).

### **wxWindow::PageUp**

This is just a wrapper for `ScrollPages()` (p. 1833)(-1).

### **wxWindow::PopEventHandler**

**wxEvtHandler\* PopEventHandler**(**bool** *deleteHandler* = *false*) **const**

Removes and returns the top-most event handler on the event handler stack.

#### **Parameters**

*deleteHandler*

If this is `true`, the handler will be deleted after it is removed. The default value is `false`.

#### **See also**

`wxWindow::SetEventHandler` (p. 1838), `wxWindow::GetEventHandler` (p. 1813), `wxWindow::PushEventHandler` (p. 1829), `wxEvtHandler::ProcessEvent` (p. 580), `wxEvtHandler` (p. 576)

### **wxWindow::PopupMenu**

**bool PopupMenu**(**wxMenu\*** *menu*, **const wxPoint&** *pos* = *wxDefaultPosition*)

**bool PopupMenu**(**wxMenu\*** *menu*, **int** *x*, **int** *y*)

Pops up the given menu at the specified coordinates, relative to this window, and returns control when the user has dismissed the menu. If a menu item is selected, the corresponding menu event is generated and will be processed as usually. If the coordinates are not specified, the current mouse cursor position is used.

#### **Parameters**

*menu*

Menu to pop up.

*pos*

The position where the menu will appear.

*x*

Required x position for the menu to appear.

*y*

Required y position for the menu to appear.

### See also

*wxMenu* (p. 1074)

### Remarks

Just before the menu is popped up, *wxMenu::UpdateUI* (p. 1087) is called to ensure that the menu items are in the correct state. The menu does not get deleted by the window.

It is recommended to not explicitly specify coordinates when calling *PopupMenu* in response to mouse click, because some of the ports (namely, *wxGTK*) can do a better job of positioning the menu in that case.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

***PopupMenu(menu, point)*** Specifies position with a *wxPoint*

***PopupMenuXY(menu, x, y)*** Specifies position with two integers (*x*, *y*)

## **wxWindow::PushEventHandler**

***void PushEventHandler(wxEvtHandler\* handler)***

Pushes this event handler onto the event stack for the window.

### Parameters

*handler*

Specifies the handler to be pushed.

### Remarks

An event handler is an object that is capable of processing the events sent to a window. By default, the window is its own event handler, but an application may wish to substitute another, for example to allow central implementation of event-handling for a variety of different window classes.

*wxWindow::PushEventHandler* (p. 1829) allows an application to set up a chain of event handlers, where an event not handled by one event handler is handed to the next one in the chain. Use *wxWindow::PopEventHandler* (p. 1828) to remove the event handler.

### See also

*wxWindow::SetEventHandler* (p. 1838), *wxWindow::GetEventHandler* (p. 1813), *wxWindow::PopEventHandler* (p. 1829), *wxEvtHandler::ProcessEvent* (p. 580),

*wxEvtHandler* (p. 576)

### **wxWindow::Raise**

**void Raise()**

Raises the window to the top of the window hierarchy (z-order).

In current version of wxWidgets this works both for managed and child windows.

#### **See also**

*Lower* (p. 1825)

### **wxWindow::Refresh**

**virtual void Refresh**(*bool eraseBackground = true*, **const wxRect\*** *rect = NULL*)

Causes this window, and all of its children recursively (except under wxGTK1 where this is not implemented), to be repainted. Note that repainting doesn't happen immediately but only during the next event loop iteration, if you need to update the window immediately you should use *Update* (p. 1852) instead.

#### **Parameters**

*eraseBackground*

If *true*, the background will be erased.

*rect*

If non-NULL, only the given rectangle will be treated as damaged.

#### **See also**

*wxWindow::RefreshRect* (p. 1830)

### **wxWindow::RefreshRect**

**void RefreshRect**(**const wxRect&** *rect*, **bool** *eraseBackground = true*)

Redraws the contents of the given rectangle: only the area inside it will be repainted.

This is the same as *Refresh* (p. 1830) but has a nicer syntax as it can be called with a temporary *wxRect* object as argument like `thisRefreshRect(wxRect(x, y, w, h))`.

### **wxWindow::RegisterHotKey**

**bool RegisterHotKey**(**int** *hotkeyId*, **int** *modifiers*, **int** *virtualKeyCode*)

Registers a system wide hotkey. Every time the user presses the hotkey registered here, this window will receive a hotkey event. It will receive the event even if the application is in

the background and does not have the input focus because the user is working with some other application.

### Parameters

*hotkeyId*

Numeric identifier of the hotkey. For applications this must be between 0 and 0xBFFF. If this function is called from a shared DLL, it must be a system wide unique identifier between 0xC000 and 0xFFFF. This is a MSW specific detail.

*modifiers*

A bitwise combination of `wxMOD_SHIFT`, `wxMOD_CONTROL`, `wxMOD_ALT` or `wxMOD_WIN` specifying the modifier keys that have to be pressed along with the key.

*virtualKeyCode*

The virtual key code of the hotkey.

### Return value

`true` if the hotkey was registered successfully. `false` if some other application already registered a hotkey with this modifier/virtualKeyCode combination.

### Remarks

Use `EVT_HOTKEY(hotkeyId, fnc)` in the event table to capture the event. This function is currently only implemented under Windows. It is used in the *Windows CE port* (p. 2235) for detecting hardware button presses.

### See also

`wxWindow::UnregisterHotKey` (p. 1851)

### **wxWindow::ReleaseMouse**

**virtual void ReleaseMouse()**

Releases mouse input captured with `wxWindow::CaptureMouse` (p. 1799).

### See also

`wxWindow::CaptureMouse` (p. 1799), `wxWindow::HasCapture` (p. 1821), `wxWindow::ReleaseMouse` (p. 1831), `wxMouseCaptureLostEvent` (p. 1119) `wxMouseCaptureChangedEvent` (p. 1118)

### **wxWindow::RemoveChild**

**virtual void RemoveChild(wxWindow\* child)**

Removes a child window. This is called automatically by window deletion functions so should not be required by the application programmer.

Notice that this function is mostly internal to `wxWidgets` and shouldn't be called by the user code.

### Parameters

*child*

Child window to remove.

### **wxWindow::RemoveEventHandler**

**bool RemoveEventHandler(*wxEvtHandler* \**handler*)**

Find the given *handler* in the windows event handler chain and remove (but not delete) it from it.

### Parameters

*handler*

The event handler to remove, must be non-NULL and must be present in this windows event handlers chain

### Return value

Returns `true` if it was found and `false` otherwise (this also results in an assert failure so this function should only be called when the handler is supposed to be there).

### See also

*PushEventHandler* (p. 1829), *PopEventHandler* (p. 1828)

### **wxWindow::Reparent**

**virtual bool Reparent(*wxWindow*\* *newParent*)**

Reparents the window, i.e the window will be removed from its current parent window (e.g. a non-standard toolbar in a `wxFrame`) and then re-inserted into another.

### Parameters

*newParent*

New parent.

### **wxWindow::ScreenToClient**

**virtual void ScreenToClient(int\* *x*, int\* *y*) const**

**virtual wxPoint ScreenToClient(const wxPoint& *pt*) const**

Converts from screen to client window coordinates.



**Parameters***x*

Stores the screen x coordinate and receives the client x coordinate.

*y*

Stores the screen x coordinate and receives the client x coordinate.

*pt*

The screen position for the second form of the function.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

**ScreenToClient(point)**      Accepts and returns a wxPoint

**ScreenToClientXY(x, y)**      Returns a 2-tuple, (x, y)

**wxWindow::ScrollLines**

**virtual bool ScrollLines(int lines)**

Scrolls the window by the given number of lines down (if *lines* is positive) or up.

**Return value**

Returns `true` if the window was scrolled, `false` if it was already on top/bottom and nothing was done.

**Remarks**

This function is currently only implemented under MSW and wxTextCtrl under wxGTK (it also works for wxScrolledWindow derived classes under all platforms).

**See also**

*ScrollPages* (p. 1833)

**wxWindow::ScrollPages**

**virtual bool ScrollPages(int pages)**

Scrolls the window by the given number of pages down (if *pages* is positive) or up.

**Return value**

Returns `true` if the window was scrolled, `false` if it was already on top/bottom and nothing was done.

**Remarks**

This function is currently only implemented under MSW and wxGTK.

**See also**

*ScrollLines* (p. 1833)

**wxWindow::ScrollWindow**

**virtual void ScrollWindow**(int *dx*, int *dy*, const wxRect\* *rect* = NULL)

Physically scrolls the pixels in the window and move child windows accordingly.

**Parameters**

*dx*

Amount to scroll horizontally.

*dy*

Amount to scroll vertically.

*rect*

Rectangle to scroll, if it is NULL, the whole window is scrolled (this is always the case under wxGTK which doesn't support this parameter)

**Remarks**

Note that you can often use *wxScrolledWindow* (p. 1414) instead of using this function directly.

**wxWindow::SetAcceleratorTable**

**virtual void SetAcceleratorTable**(const wxAcceleratorTable& *accel*)

Sets the accelerator table for this window. See *wxAcceleratorTable* (p. 23).

**wxWindow::SetAccessible**

**void SetAccessible**(wxAccessible\* *accessible*)

Sets the accessible for this window. Any existing accessible for this window will be deleted first, if not identical to *accessible*.

See also *wxAccessible* (p. 25).

**wxWindow::SetAutoLayout**

**void SetAutoLayout**(bool *autoLayout*)

Determines whether the *wxWindow::Layout* (p. 1825) function will be called automatically when the window is resized. Please note that this only happens for the windows usually

used to contain children, namely *wxPanel* (p. 1170) and *wxTopLevelWindow* (p. 1713) (and the classes deriving from them).

This method is called implicitly by *wxWindow::SetSizer* (p. 1846) but if you use *wxWindow::SetConstraints* (p. 1837) you should call it manually or otherwise the window layout won't be correctly updated when its size changes.

### Parameters

*autoLayout*

Set this to `true` if you wish the *Layout* function to be called automatically when the window is resized.

### See also

*wxWindow::SetConstraints* (p. 1837)

## **wxWindow::SetBackgroundColour**

**virtual bool SetBackgroundColour(const wxColour& colour)**

Sets the background colour of the window.

Please see *InheritAttributes* (p. 1822) for explanation of the difference between this method and *SetOwnBackgroundColour* (p. 1842).

### Parameters

*colour*

The colour to be used as the background colour, pass `wxNullColour` to reset to the default colour.

### Remarks

The background colour is usually painted by the default *wxEraseEvent* (p. 571) event handler function under Windows and automatically under GTK.

Note that setting the background colour does not cause an immediate refresh, so you may wish to call *wxWindow::ClearBackground* (p. 1801) or *wxWindow::Refresh* (p. 1830) after calling this function.

Using this function will disable attempts to use themes for this window, if the system supports them. Use with care since usually the themes represent the appearance chosen by the user to be used for all applications on the system.

### See also

*wxWindow::GetBackgroundColour* (p. 1809), *wxWindow::SetForegroundColour* (p. 1840), *wxWindow::GetForegroundColour* (p. 1813), *wxWindow::ClearBackground* (p. 1801), *wxWindow::Refresh* (p. 1830), *wxEraseEvent* (p. 571)

**wxWindow::SetBackgroundStyle****virtual void SetBackgroundStyle(wxBackgroundStyle style)**

Sets the background style of the window. The background style indicates whether background colour should be determined by the system (`wxBG_STYLE_SYSTEM`), be set to a specific colour (`wxBG_STYLE_COLOUR`), or should be left to the application to implement (`wxBG_STYLE_CUSTOM`).

On GTK+, use of `wxBG_STYLE_CUSTOM` allows the flicker-free drawing of a custom background, such as a tiled bitmap. Currently the style has no effect on other platforms.

**See also**

*wxWindow::SetBackgroundColour* (p. 1835), *wxWindow::GetForegroundColour* (p. 1813), *wxWindow::GetBackgroundStyle* (p. 1809)

**wxWindow::SetInitialSize****void SetInitialSize(const wxSize& size = wxDefaultSize)**

A *smart* `SetSize` that will fill in default size components with the window's *best* size values. Also sets the window's minsize to the value passed in for use with sizers. This means that if a full or partial size is passed to this function then the sizers will use that size instead of the results of `GetBestSize` to determine the minimum needs of the window for layout.

Most controls will use this to set their initial size, and their min size to the passed in value (if any.)

**See also**

*wxWindow::SetSize* (p. 1845), *wxWindow::GetBestSize* (p. 1810), *wxWindow::GetEffectiveMinSize* (p. 1810)

**wxWindow::SetCaret****void SetCaret(wxCaret \*caret) const**

Sets the *caret* (p. 177) associated with the window.

**wxWindow::SetClientSize****virtual void SetClientSize(int width, int height)****virtual void SetClientSize(const wxSize& size)**

This sets the size of the window client area in pixels. Using this function to size a window tends to be more device-independent than *wxWindow::SetSize* (p. 1845), since the application need not worry about what dimensions the border or title bar have when trying to fit the window around panel items, for example.

**Parameters**

*width*

The required client area width.

*height*

The required client area height.

*size*

The required client size.

**wxPython note:** In place of a single overloaded method name, wxPython implements the following methods:

**SetClientSize(size)**                      Accepts a wxSize

**SetClientSizeWH(width, height)**

### **wxWindow::SetConstraints**

**void SetConstraints(wxLayoutConstraints\* constraints)**

Sets the window to have the given layout constraints. The window will then own the object, and will take care of its deletion. If an existing layout constraints object is already owned by the window, it will be deleted.

#### **Parameters**

*constraints*

The constraints to set. Pass NULL to disassociate and delete the window's constraints.

#### **Remarks**

You must call *wxWindow::SetAutoLayout* (p. 1834) to tell a window to use the constraints automatically in *OnSize*; otherwise, you must override *OnSize* and call *Layout()* explicitly. When setting both a *wxLayoutConstraints* and a *wxSizer* (p. 1444), only the sizer will have effect.

### **wxWindow::SetContainingSizer**

**void SetContainingSizer(wxSizer\* sizer)**

This normally does not need to be called by user code. It is called when a window is added to a sizer, and is used so the window can remove itself from the sizer when it is destroyed.

### **wxWindow::SetCursor**

**virtual void SetCursor(const wxCursor&cursor)**

Sets the window's cursor. Notice that the window cursor also sets it for the children of the window implicitly.

The *cursor* may be `wxNullCursor` in which case the window cursor will be reset back to default.

### Parameters

*cursor*

Specifies the cursor that the window should normally display.

### See also

`::wxSetCursor` (p. 1946), `wxCursor` (p. 297)

### **wxWindow::SetDropTarget**

**void SetDropTarget(wxDropTarget\* target)**

Associates a drop target with this window.

If the window already has a drop target, it is deleted.

### See also

`wxWindow::GetDropTarget` (p. 1813), *Drag and drop overview* (p. 2150)

### **wxWindow::SetInitialBestSize**

**virtual void SetInitialBestSize(const wxSize& size)**

Sets the initial window size if none is given (i.e. at least one of the components of the size passed to `ctor/Create()` is `wxDefaultCoord`).

### **wxWindow::SetEventHandler**

**void SetEventHandler(wxEvtHandler\* handler)**

Sets the event handler for this window.

### Parameters

*handler*

Specifies the handler to be set.

### Remarks

An event handler is an object that is capable of processing the events sent to a window. By default, the window is its own event handler, but an application may wish to substitute another, for example to allow central implementation of event-handling for a variety of different window classes.

It is usually better to use `wxWindow::PushEventHandler` (p. 1829) since this sets up a chain of event handlers, where an event not handled by one event handler is handed to the next one in the chain.

### See also

`wxWindow::GetEventHandler` (p. 1813), `wxWindow::PushEventHandler` (p. 1829),  
`wxWindow::PopEventHandler` (p. 1829), `wxEvtHandler::ProcessEvent` (p. 580),  
`wxEvtHandler` (p. 576)

## **wxWindow::SetExtraStyle**

**void SetExtraStyle(long *exStyle*)**

Sets the extra style bits for the window. The currently defined extra style bits are:

**wxWS\_EX\_VALIDATE\_RECURSIVELY**     `TransferDataTo/FromWindow()` and `Validate()` methods will recursively descend into all children of the window if it has this style flag set.

**wxWS\_EX\_BLOCK\_EVENTS**     Normally, the command events are propagated upwards to the window parent recursively until a handler for them is found. Using this style allows to prevent them from being propagated beyond this window. Notice that `wxDialog` has this style on by default for the reasons explained in *the event processing overview* (p. 2078).

**wxWS\_EX\_TRANSIENT**     This can be used to prevent a window from being used as an implicit parent for the dialogs which were created without a parent. It is useful for the windows which can disappear at any moment as creating children of such windows results in fatal problems.

**wxWS\_EX\_CONTEXTHELP**     Under Windows, puts a query button on the caption. When pressed, Windows will go into a context-sensitive help mode and `wxWidgets` will send a `wxEVT_HELP` event if the user clicked on an application window. This style cannot be used together with `wxMAXIMIZE_BOX` or `wxMINIMIZE_BOX`, so these two styles are automatically turned off if this one is used.

**wxWS\_EX\_PROCESS\_IDLE**     This window should always process idle events, even if the mode set by `wxIdleEvent::SetMode` (p. 905) is `wxIDLE_PROCESS_SPECIFIED`.

**wxWS\_EX\_PROCESS\_UI\_UPDATES**     This window should always process UI update events, even if the mode set by `wxUpdateUIEvent::SetMode` (p. 1756) is `wxUPDATE_UI_PROCESS_SPECIFIED`.

## **wxWindow::SetFocus**

**virtual void SetFocus()**

This sets the window to receive keyboard input.

**See also**

*wxFocusEvent* (p. 655) *wxPanel::SetFocus* (p. 1172) *wxPanel::SetFocusIgnoringChildren* (p. 1172)

**wxWindow::SetFocusFromKbd****virtual void SetFocusFromKbd()**

This function is called by wxWidgets keyboard navigation code when the user gives the focus to this window from keyboard (e.g. using `TAB` key). By default this method simply calls *SetFocus* (p. 1839) but can be overridden to do something in addition to this in the derived classes.

**wxWindow::SetFont****void SetFont(const wxFont& font)**

Sets the font for this window. This function should not be called for the parent window if you don't want its font to be inherited by its children, use *SetOwnFont* (p. 1843) instead in this case and see *InheritAttributes* (p. 1822) for more explanations.

Please notice that the given font is *not* automatically used for *wxPaintDC* (p. 1164) objects associated with this window, you need to call *wxDC::SetFont()* (p. 473) too. However this font is used by any standard controls for drawing their text as well as by *wxWindow::GetTextExtent()* (p. 1819).

**Parameters**

*font*

Font to associate with this window, pass `wxNullFont` to reset to the default font.

**See also**

*wxWindow::GetFont* (p. 1813),  
*InheritAttributes* (p. 1822)

**wxWindow::SetForegroundColour****virtual void SetForegroundColour(const wxColour& colour)**

Sets the foreground colour of the window.

Please see *InheritAttributes* (p. 1822) for explanation of the difference between this method and *SetOwnForegroundColour* (p. 1843).

**Parameters**



*colour*

The colour to be used as the foreground colour, pass `wxNullColour` to reset to the default colour.

### Remarks

The interpretation of foreground colour is open to interpretation according to the window class; it may be the text colour or other colour, or it may not be used at all.

Using this function will disable attempts to use themes for this window, if the system supports them. Use with care since usually the themes represent the appearance chosen by the user to be used for all applications on the system.

### See also

`wxWindow::GetForegroundColour` (p. 1813), `wxWindow::SetBackgroundColour` (p. 1835), `wxWindow::GetBackgroundColour` (p. 1809), `wxWindow::ShouldInheritColours` (p. 1849)

## **wxWindow::SetHelpText**

**virtual void SetHelpText(const wxString& helpText)**

Sets the help text to be used as context-sensitive help for this window.

Note that the text is actually stored by the current `wxHelpProvider` (p. 817) implementation, and not in the window object itself.

### See also

`GetHelpText` (p. 1814), `wxHelpProvider` (p. 817)

## **wxWindow::SetId**

**void SetId(int id)**

Sets the identifier of the window.

### Remarks

Each window has an integer identifier. If the application has not provided one, an identifier will be generated. Normally, the identifier should be provided on creation and should not be modified subsequently.

### See also

`wxWindow::GetId` (p. 1815), *Window identifiers* (p. 2082)

## **wxWindow::SetLabel**

**virtual void SetLabel(const wxString& label)**

Sets the window's label.

**Parameters**

*label*

The window label.

**See also**

*wxWindow::GetLabel* (p. 1815)

**wxWindow::SetMaxSize**

**void SetMaxSize(const wxSize& size)**

Sets the maximum size of the window, to indicate to the sizer layout mechanism that this is the maximum possible size.

**wxWindow::SetMinSize**

**void SetMinSize(const wxSize& size)**

Sets the minimum size of the window, to indicate to the sizer layout mechanism that this is the minimum required size. You may need to call this if you change the window size after construction and before adding to its parent sizer.

**wxWindow::SetName**

**virtual void SetName(const wxString& name)**

Sets the window's name.

**Parameters**

*name*

A name to set for the window.

**See also**

*wxWindow::GetName* (p. 1815)

**wxWindow::SetOwnBackgroundColour**

**void SetOwnBackgroundColour(const wxColour& colour)**

Sets the background colour of the window but prevents it from being inherited by the children of this window.

**See also**

*SetBackgroundColour* (p. 1835), *InheritAttributes* (p. 1822)

**wxWindow::SetOwnFont****void SetOwnFont(const wxFont& font)**

Sets the font of the window but prevents it from being inherited by the children of this window.

**See also**

*SetFont* (p. 1840), *InheritAttributes* (p. 1822)

**wxWindow::SetOwnForegroundColour****void SetOwnForegroundColour(const wxColour& colour)**

Sets the foreground colour of the window but prevents it from being inherited by the children of this window.

**See also**

*SetForegroundColour* (p. 1840), *InheritAttributes* (p. 1822)

**wxWindow::SetPalette****virtual void SetPalette(wxPalette\* palette)**

Obsolete - use *wxDC::SetPalette* (p. 475) instead.

**wxWindow::SetScrollbar****virtual void SetScrollbar(int orientation, int position, int thumbSize, int range, bool refresh = true)**

Sets the scrollbar properties of a built-in scrollbar.

**Parameters**

*orientation*

Determines the scrollbar whose page size is to be set. May be *wxHORIZONTAL* or *wxVERTICAL*.

*position*

The position of the scrollbar in scroll units.

*thumbSize*

The size of the thumb, or visible portion of the scrollbar, in scroll units.

*range*

The maximum position of the scrollbar.

*refresh*

`true` to redraw the scrollbar, `false` otherwise.

### Remarks

Let's say you wish to display 50 lines of text, using the same font. The window is sized so that you can only see 16 lines at a time.

You would use:

```
SetScrollbar(wxVERTICAL, 0, 16, 50);
```

Note that with the window at this size, the thumb position can never go above 50 minus 16, or 34.

You can determine how many lines are currently visible by dividing the current view size by the character height in pixels.

When defining your own scrollbar behaviour, you will always need to recalculate the scrollbar settings when the window size changes. You could therefore put your scrollbar calculations and `SetScrollbar` call into a function named `AdjustScrollbars`, which can be called initially and also from your `wxSizeEvent` (p. 1443) handler function.

### See also

*Scrolling overview* (p. 2115), *wxScrollBar* (p. 1408), *wxScrolledWindow* (p. 1414), *wxScrollWinEvent* (p. 1426)

## **wxWindow::SetScrollPos**

**virtual void SetScrollPos**(*int orientation*, *int pos*, **bool** *refresh = true*)

Sets the position of one of the built-in scrollbars.

### Parameters

*orientation*

Determines the scrollbar whose position is to be set. May be `wxHORIZONTAL` or `wxVERTICAL`.

*pos*

Position in scroll units.

*refresh*

`true` to redraw the scrollbar, `false` otherwise.

### Remarks

This function does not directly affect the contents of the window: it is up to the application to take note of scrollbar attributes and redraw contents accordingly.

**See also**

*wxWindow::SetScrollbar* (p. 1843), *wxWindow::GetScrollPos* (p. 1818),  
*wxWindow::GetScrollThumb* (p. 1818), *wxScrollBar* (p. 1408), *wxScrolledWindow* (p. 1414)

**wxWindow::SetSize**

**virtual void SetSize(int x, int y, int width, int height, int sizeFlags = wxSIZE\_AUTO)**

**virtual void SetSize(const wxRect& rect)**

Sets the size and position of the window in pixels.

**virtual void SetSize(int width, int height)**

**virtual void SetSize(const wxSize& size)**

Sets the size of the window in pixels.

**Parameters**

*x*

Required x position in pixels, or *wxDefaultCoord* to indicate that the existing value should be used.

*y*

Required y position in pixels, or *wxDefaultCoord* to indicate that the existing value should be used.

*width*

Required width in pixels, or *wxDefaultCoord* to indicate that the existing value should be used.

*height*

Required height position in pixels, or *wxDefaultCoord* to indicate that the existing value should be used.

*size*

*wxSize* (p. 1440) object for setting the size.

*rect*

*wxRect* (p. 1252) object for setting the position and size.

*sizeFlags*

Indicates the interpretation of other parameters. It is a bit list of the following:

**wxSIZE\_AUTO\_WIDTH**: a wxDefaultCoord width value is taken to indicate a wxWidgets-supplied default width.

**wxSIZE\_AUTO\_HEIGHT**: a wxDefaultCoord height value is taken to indicate a wxWidgets-supplied default height.

**wxSIZE\_AUTO**: wxDefaultCoord size values are taken to indicate a wxWidgets-supplied default size.

**wxSIZE\_USE\_EXISTING**: existing dimensions should be used if wxDefaultCoord values are supplied.

**wxSIZE\_ALLOW\_MINUS\_ONE**: allow negative dimensions (ie. value of wxDefaultCoord) to be interpreted as real dimensions, not default

values. **wxSIZE\_FORCE**: normally, if the position and the size of the window are already the same as the parameters of this function, nothing is done. but with this flag a window resize may be forced even in this case (supported in wx 2.6.2 and later and only implemented for MSW and ignored elsewhere currently)

### Remarks

The second form is a convenience for calling the first form with default x and y parameters, and must be used with non-default width and height values.

The first form sets the position and optionally size, of the window. Parameters may be wxDefaultCoord to indicate either that a default should be supplied by wxWidgets, or that the current value of the dimension should be used.

### See also

*wxWindow::Move* (p. 1826)

**wxPython note**: In place of a single overloaded method name, wxPython implements the following methods:

**SetDimensions(x, y, width, height, sizeFlags=wxSIZE\_AUTO)**

**SetSize(size)**

**SetPosition(point)**

### wxWindow::SetSizeHints

Use of this function for windows which are not toplevel windows (such as wxDialog or wxFrame) is discouraged. Please use *SetMinSize* (p. 1842) and *SetMaxSize* (p. 1842) instead.

### See also

*wxTopLevelWindow::SetSizeHints* (p. 1718).

### wxWindow::SetSizer

**void SetSizer(wxSizer\* sizer, bool deleteOld=true)**

Sets the window to have the given layout sizer. 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 if the `deleteOld` parameter is `true`.

Note that this function will also call `SetAutoLayout` (p. 1834) implicitly with `true` parameter if the *sizer* is non-NULL and `false` otherwise.

### Parameters

*sizer*

The sizer to set. Pass NULL to disassociate and conditionally delete the window's sizer. See below.

*deleteOld*

If `true` (the default), this will delete any preexisting sizer. Pass `false` if you wish to handle deleting the old sizer yourself.

### Remarks

`SetSizer` now enables and disables Layout automatically, but prior to wxWidgets 2.3.3 the following applied:

You must call `wxWindow::SetAutoLayout` (p. 1834) to tell a window to use the sizer automatically in `OnSize`; otherwise, you must override `OnSize` and call `Layout()` explicitly. When setting both a `wxSizer` and a `wxLayoutConstraints` (p. 964), only the sizer will have effect.

### **wxWindow::SetSizerAndFit**

**void SetSizerAndFit(wxSizer\* sizer, bool deleteOld=true)**

The same as `SetSizer` (p. 1846), except it also sets the size hints for the window based on the sizer's minimum size.

### **wxWindow::SetThemeEnabled**

**virtual void SetThemeEnabled(bool enable)**

This function tells a window if it should use the system's "theme" code to draw the windows' background instead of its own background drawing code. This does not always have any effect since the underlying platform obviously needs to support the notion of themes in user defined windows. One such platform is GTK+ where windows can have (very colourful) backgrounds defined by a user's selected theme.

Dialogs, notebook pages and the status bar have this flag set to `true` by default so that the default look and feel is simulated best.

### **wxWindow::SetToolTip**

**void SetToolTip(const wxString& tip)**

**void SetToolTip(wxToolTip\* tip)**

Attach a tooltip to the window.

See also: *GetToolTip* (p. 1820), *wxToolTip* (p. 1712)

**wxWindow::SetValidator**

**virtual void SetValidator(const wxValidator& validator)**

Deletes the current validator (if any) and sets the window validator, having called `wxValidator::Clone` to create a new validator of this type.

**wxWindow::SetVirtualSize**

**void SetVirtualSize(int width, int height)**

**void SetVirtualSize(const wxSize& size)**

Sets the virtual size of the window in pixels.

**wxWindow::SetVirtualSizeHints**

**virtual void SetVirtualSizeHints(int minW, int minH, int maxW=-1, int maxH=-1)**

**void SetVirtualSizeHints(const wxSize& minSize=wxDefaultSize, const wxSize& maxSize=wxDefaultSize)**

Allows specification of minimum and maximum virtual window sizes. If a pair of values is not set (or set to -1), the default values will be used.

#### Parameters

*minW*

Specifies the minimum width allowable.

*minH*

Specifies the minimum height allowable.

*maxW*

Specifies the maximum width allowable.

*maxH*

Specifies the maximum height allowable.

*minSize*

Minimum size.

*maxSize*



Maximum size.

### Remarks

If this function is called, the user will not be able to size the virtual area of the window outside the given bounds.

### **wxWindow::SetWindowStyle**

**void SetWindowStyle(long style)**

Identical to *SetWindowStyleFlag* (p. 1849).

### **wxWindow::SetWindowStyleFlag**

**virtual void SetWindowStyleFlag(long style)**

Sets the style of the window. Please note that some styles cannot be changed after the window creation and that *Refresh()* (p. 1830) might need to be called after changing the others for the change to take place immediately.

See *Window styles* (p. 2089) for more information about flags.

### See also

*GetWindowStyleFlag* (p. 1821)

### **wxWindow::SetWindowVariant**

**void SetWindowVariant(wxWindowVariant variant)**

This function can be called under all platforms but only does anything under Mac OS X 10.3+ currently. Under this system, each of the standard control can exist in several sizes which correspond to the elements of `wxWindowVariant` enum:

```
enum wxWindowVariant
{
    wxWINDOW_VARIANT_NORMAL,           // Normal size
    wxWINDOW_VARIANT_SMALL,           // Smaller size (about 25 % smaller
    than normal )
    wxWINDOW_VARIANT_MINI,            // Mini size (about 33 % smaller
    than normal )
    wxWINDOW_VARIANT_LARGE,           // Large size (about 25 % larger
    than normal )
};
```

By default the controls use the normal size, of course, but this function can be used to change this.

### **wxWindow::ShouldInheritColours**

**virtual bool ShouldInheritColours()**

Return `true` from here to allow the colours of this window to be changed

by *InheritAttributes* (p. 1822), returning `false` forbids inheriting them from the parent window.

The base class version returns `false`, but this method is overridden in *wxControl* (p. 285) where it returns `true`.

## **wxWindow::Show**

**virtual bool Show**(bool *show* = `true`)

Shows or hides the window. You may need to call *Raise* (p. 1830) for a top level window if you want to bring it to top, although this is not needed if *Show()* is called immediately after the frame creation.

### **Parameters**

*show*

If `true` displays the window. Otherwise, hides it.

### **Return value**

`true` if the window has been shown or hidden or `false` if nothing was done because it already was in the requested state.

### **See also**

*wxWindow::IsShown* (p. 1824), *wxWindow::Hide* (p. 1822), *wxRadioBox::Show* (p. 1248)

## **wxWindow::Thaw**

**virtual void Thaw**()

Reenables window updating after a previous call to *Freeze* (p. 1808). To really thaw the control, it must be called exactly the same number of times as *Freeze* (p. 1808).

### **See also**

*wxWindowUpdateLocker* (p. 1854)

## **wxWindow::ToggleWindowStyle**

**bool ToggleWindowStyle**(int *flag*)

Turns the given *flag* on if it's currently turned off and vice versa. This function cannot be used if the value of the flag is 0 (which is often the case for default flags).

Also, please notice that not all styles can be changed after the control creation.

### **Return value**

Returns `true` if the style was turned on by this function, `false` if it was switched off.

**See also**

*wxWindow::SetWindowStyleFlag* (p. 1849), *wxWindow::HasFlag* (p. 1822)

**wxWindow::TransferDataFromWindow****virtual bool TransferDataFromWindow()**

Transfers values from child controls to data areas specified by their validators.  
Returns `false` if a transfer failed.

If the window has `wxWS_EX_VALIDATE_RECURSIVELY` extra style flag set, the method will also call *TransferDataFromWindow()* of all child windows.

**See also**

*wxWindow::TransferDataToWindow* (p. 1851), *wxValidator* (p. 1767),  
*wxWindow::Validate* (p. 1853)

**wxWindow::TransferDataToWindow****virtual bool TransferDataToWindow()**

Transfers values to child controls from data areas specified by their validators.

If the window has `wxWS_EX_VALIDATE_RECURSIVELY` extra style flag set, the method will also call *TransferDataToWindow()* of all child windows.

**Return value**

Returns `false` if a transfer failed.

**See also**

*wxWindow::TransferDataFromWindow* (p. 1851), *wxValidator* (p. 1767),  
*wxWindow::Validate* (p. 1853)

**wxWindow::UnregisterHotKey****bool UnregisterHotKey(int hotkeyId)**

Unregisters a system wide hotkey.

**Parameters**

*hotkeyId*

Numeric identifier of the hotkey. Must be the same id that was passed to *RegisterHotKey*.

**Return value**

`true` if the hotkey was unregistered successfully, `false` if the id was invalid.

**Remarks**

This function is currently only implemented under MSW.

**See also**

*wxWindow::RegisterHotKey* (p. 1830)

**wxWindow::Update****virtual void Update()**

Calling this method immediately repaints the invalidated area of the window and all of its children recursively while this would usually only happen when the flow of control returns to the event loop. Notice that this function doesn't invalidate any area of the window so nothing happens if nothing has been invalidated (i.e. marked as requiring a redraw). Use *Refresh* (p. 1830) first if you want to immediately redraw the window unconditionally.

**wxWindow::UpdateWindowUI****virtual void UpdateWindowUI(long flags = wxUPDATE\_UI\_NONE)**

This function sends *wxUpdateUIEvents* (p. 1752) to the window. The particular implementation depends on the window; for example a *wxToolBar* will send an update UI event for each toolbar button, and a *wxFrame* will send an update UI event for each menubar menu item. You can call this function from your application to ensure that your UI is up-to-date at this point (as far as your *wxUpdateUIEvent* handlers are concerned). This may be necessary if you have called *wxUpdateUIEvent::SetMode* (p. 1756) or *wxUpdateUIEvent::SetUpdateInterval* (p. 1757) to limit the overhead that *wxWidgets* incurs by sending update UI events in idle time.

*flags* should be a bitlist of one or more of the following values.

```
enum wxUpdateUI
{
    wxUPDATE_UI_NONE           = 0x0000, // No particular value
    wxUPDATE_UI_RECURSE       = 0x0001, // Call the function for
    wxUPDATE_UI_FROMIDLE      = 0x0002 // Invoked from On(Internal)Idle
};
```

If you are calling this function from an *OnInternalIdle* or *OnIdle* function, make sure you pass the *wxUPDATE\_UI\_FROMIDLE* flag, since this tells the window to only update the UI elements that need to be updated in idle time. Some windows update their elements only when necessary, for example when a menu is about to be shown. The following is an example of how to call *UpdateWindowUI* from an idle function.

```
void MyWindow::OnInternalIdle()
{
    if (wxUpdateUIEvent::CanUpdate(this))
        UpdateWindowUI(wxUPDATE_UI_FROMIDLE);
}
```

**See also**

*wxUpdateUIEvent* (p. 1752), *wxWindow::DoUpdateWindowUI* (p. 1805), *wxWindow::OnInternalIdle* (p. 1827)

**wxWindow::Validate****virtual bool Validate()**

Validates the current values of the child controls using their validators.

If the window has `wxWS_EX_VALIDATE_RECURSIVELY` extra style flag set, the method will also call `Validate()` of all child windows.

**Return value**

Returns `false` if any of the validations failed.

**See also**

*wxWindow::TransferDataFromWindow* (p. 1851), *wxWindow::TransferDataToWindow* (p. 1851), *wxValidator* (p. 1767)

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

Moves the pointer to the given position on the window.

**NB:** This function is not supported under Mac because Apple Human Interface Guidelines forbid moving the mouse cursor programmatically.

**Parameters**

*x*

The new x position for the cursor.

*y*

The new y position for the cursor.

**wxWindow::WindowToClientSize****virtual wxSize WindowToClientSize(const wxSize& size)**

Converts window size *size* to corresponding client area size. In other words, the returned value is what would *GetClientSize* (p. 1811) return if this window had given window size. Components with `wxDefaultCoord` value are left unchanged.

Note that the conversion is not always exact, it assumes that non-client area doesn't change and so doesn't take into account things like menu bar (un)wrapping or (dis)appearance of the scrollbars.

This function is new since wxWidgets version 2.8.8

See also

## ***wxWindow::ClientToWindowSize* (p. 1802) *wxWindowUpdateLocker***

This tiny class prevents redrawing of a *wxWindow* (p. 1795) during its lifetime by using *wxWindow::Freeze* (p. 1808) and *Thaw* (p. 1850) methods. It is typically used for creating automatic objects to temporarily suppress window updates before a batch of operations is performed:

```
void MyFrame::Foo()
{
    m_text = new wxTextCtrl(this, ...);

    wxWindowUpdateLocker noUpdates(m_text);
    m_text->AppendText();
    ... many other operations with m_text...
    m_text->WriteText();
}
```

Using this class is easier and safer than calling *Freeze* (p. 1808) and *Thaw* (p. 1850) because you don't risk to forget calling the latter.

**Derived from**

None.

**Include files**

<wx/wupdlock.h>

### ***wxWindowUpdateLocker::wxWindowUpdateLocker***

***wxWindowUpdateLocker*(*wxWindow \*win*)**

Creates an object preventing the updates of the specified *win*. The parameter must be non-NULL and the window must exist for longer than *wxWindowUpdateLocker* object itself.

### ***wxWindowUpdateLocker::~~wxWindowUpdateLocker***

***~wxWindowUpdateLocker*()**

Destructor reenables updates for the window this object is associated with.

## ***wxWindowCreateEvent***

This event is sent just after the actual window associated with a `wxWindow` object has been created. Since it is derived from `wxCommandEvent`, the event propagates up the window hierarchy.

**Derived from**

`wxCommandEvent` (p. 572)

`wxEvent` (p. 572)

`wxObject` (p. 1148)

**Include files**

<wx/event.h>

**Event table macros**

To process a window creation event, use this event handler macro to direct input to a member function that takes a `wxWindowCreateEvent` argument.

**EVT\_WINDOW\_CREATE(func)**            Process a `wxEVT_CREATE` event.

**See also**

*Event handling overview* (p. 2077), `wxWindowDestroyEvent` (p. 1856)

**wxWindowCreateEvent::wxWindowCreateEvent**

**wxWindowCreateEvent(wxWindow\* win = NULL)**

Constructor.

**wxWindowDC**

A `wxWindowDC` must be constructed if an application wishes to paint on the whole area of a window (client and decorations). This should normally be constructed as a temporary stack object; don't store a `wxWindowDC` object.

To draw on a window from inside **OnPaint**, construct a `wxPaintDC` (p. 1164) object.

To draw on the client area of a window from outside **OnPaint**, construct a `wxClientDC` (p. 193) object.

To draw on the whole window including decorations, construct a `wxWindowDC` (p. 1855) object (Windows only).

**Derived from**

`wxDC` (p. 456)

**Include files**

<wx/dcclient.h>

**See also**

*wxDC* (p. 456), *wxMemoryDC* (p. 1069), *wxPaintDC* (p. 1164), *wxClientDC* (p. 193), *wxScreenDC* (p. 1407)

**wxWindowDC::wxWindowDC**

**wxWindowDC**(*wxWindow\** window)

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

**wxWindowDestroyEvent**

This event is sent from the *wxWindow* destructor *wxWindow::~~wxWindow()* when a window is destroyed.

When a class derived from *wxWindow* is destroyed its destructor will have already run by the time this event is sent. Therefore this event will not usually be received at all.

To receive this event *wxEvtHandler::Connect* (p. 577) must be used (using an event table macro will not work). Since it is received after the destructor has run, an object should not handle its own *wxWindowDestroyEvent*, but it can be used to get notification of the destruction of another window.

**Derived from**

*wxCommandEvent* (p. 572)  
*wxEvent* (p. 572)  
*wxObject* (p. 1148)

**Include files**

<wx/event.h>

**Event table macros**

It is not possible to receive this event using an event table macro.

**See also**

*Event handling overview* (p. 2077), *wxWindowCreateEvent* (p. 1854)

**wxWindowDestroyEvent::wxWindowDestroyEvent**

**wxWindowDestroyEvent**(*wxWindow\** win = *NULL*)

Constructor.



## wxWindowDisabler

This class disables all windows of the application (may be with the exception of one of them) in its constructor and enables them back in its destructor. This comes in handy when you want to indicate to the user that the application is currently busy and cannot respond to user input.

### Derived from

None

### Include files

<wx/utils.h>

### See also

*wxBusyCursor* (p. 162)

## wxWindowDisabler::wxWindowDisabler

**wxWindowDisabler(wxWindow \*winToSkip = NULL)**

Disables all top level windows of the applications with the exception of *winToSkip* if it is not NULL.

## wxWindowDisabler::~~wxWindowDisabler

**~wxWindowDisabler()**

Reenables back the windows disabled by the constructor.

## wxWizard

wxWizard is the central class for implementing 'wizard-like' dialogs. These dialogs are mostly familiar to Windows users and are nothing other than a sequence of 'pages', each displayed inside a dialog which has the buttons to navigate to the next (and previous) pages.

The wizards are typically used to decompose a complex dialog into several simple steps and are mainly useful to the novice users, hence it is important to keep them as simple as possible.

To show a wizard dialog, you must first create an instance of the wxWizard class using either the non-default constructor or a default one followed by call to the *Create* (p. 1859) function. Then you should add all pages you want the wizard to show and call *RunWizard* (p. 1862). Finally, don't forget to call `wizard->Destroy()`, otherwise your application will hang on exit due to an undestroyed window.

**Derived from**

*wxDialog* (p. 496)  
*wxPanel* (p. 1170)  
*wxWindow* (p. 1795)  
*wxEvtHandler* (p. 576)  
*wxObject* (p. 1148)

**Include files**

<wx/wizard.h>

**Event table macros**

To process input from a wizard dialog, use these event handler macros to direct input to member functions that take a *wxWizardEvent* (p. 1862) argument. For some events, *Veto()* (p. 1147) can be called to prevent the event from happening.

**EVT\_WIZARD\_PAGE\_CHANGED(id, func)**      The page has just been changed (this event cannot be vetoed).

**EVT\_WIZARD\_PAGE\_CHANGING(id, func)**      The page is being changed (this event can be vetoed).

**EVT\_WIZARD\_CANCEL(id, func)**      The user attempted to cancel the wizard (this event may also be vetoed).

**EVT\_WIZARD\_HELP(id, func)**      The wizard help button was pressed.

**EVT\_WIZARD\_FINISHED(id, func)**      The wizard finished button was pressed.

**Extended styles**

Use the *wxWindow::SetExtraStyle* (p. 1839) function to set the following style. You will need to use two-step construction (use the default constructor, call **SetExtraStyle**, then call **Create**).

**wxWIZARD\_EX\_HELPBUTTON**      Shows a Help button using *wxID\_HELP*.

See also *wxDialog* (p. 496) for other extended styles.

**See also**

*wxWizardEvent* (p. 1862), *wxWizardPage* (p. 1863), *wxWizard sample* (p. 2040)

**wxWizard::wxWizard****wxWizard()**

Default constructor. Use this if you wish to derive from *wxWizard* and then call *Create* (p. 1859), for example if you wish to set an extra style with *wxWindow::SetExtraStyle* (p. 1839) between the two calls.

**wxWizard**(**wxWindow\*** *parent*, **int** *id* = -1, **const wxString&** *title* = *wxEmptyString*, **const wxBitmap&** *bitmap* = *wxNullBitmap*, **const wxPoint&** *pos* = *wxDefaultPosition*, **long** *style* = *wxDEFAULT\_DIALOG\_STYLE*)

Constructor which really creates the wizard -- if you use this constructor, you shouldn't call *Create* (p. 1859).

Notice that unlike almost all other *wxWidgets* classes, there is no *size* parameter in the *wxWizard* constructor because the wizard will have a predefined default size by default. If you want to change this, you should use the *GetPageAreaSizer* (p. 1860) function.

### Parameters

*parent*

The parent window, may be NULL.

*id*

The id of the dialog, will usually be just -1.

*title*

The title of the dialog.

*bitmap*

The default bitmap used in the left side of the wizard. See also *GetBitmap* (p. 1865).

*pos*

The position of the dialog, it will be centered on the screen by default.

*style*

Window style is passed to *wxDialog*.

### **wxWizard::Create**

**bool** **Create**(**wxWindow\*** *parent*, **int** *id* = -1, **const wxString&** *title* = *wxEmptyString*, **const wxBitmap&** *bitmap* = *wxNullBitmap*, **const wxPoint&** *pos* = *wxDefaultPosition*, **long** *style* = *wxDEFAULT\_DIALOG\_STYLE*)

Creates the wizard dialog. Must be called if the default constructor had been used to create the object.

Notice that unlike almost all other *wxWidgets* classes, there is no *size* parameter in the *wxWizard* constructor because the wizard will have a predefined default size by default. If you want to change this, you should use the *GetPageAreaSizer* (p. 1860) function.

### Parameters

*parent*

The parent window, may be NULL.

*id*

The id of the dialog, will usually be just -1.

*title*

The title of the dialog.

*bitmap*

The default bitmap used in the left side of the wizard. See also *GetBitmap* (p. 1865).

*pos*

The position of the dialog, it will be centered on the screen by default.

*style*

Window style is passed to `wxDialog`.

### **wxWizard::FitToPage**

**void FitToPage(const wxWizardPage\* firstPage)**

This method is obsolete, use *GetPageAreaSizer* (p. 1860) instead.

Sets the page size to be big enough for all the pages accessible via the given *firstPage*, i.e. this page, its next page and so on.

This method may be called more than once and it will only change the page size if the size required by the new page is bigger than the previously set one. This is useful if the decision about which pages to show is taken during run-time, as in this case, the wizard won't be able to get to all pages starting from a single one and you should call *Fit* separately for the others.

### **wxWizard::GetBitmap**

**const wxBitmap& GetBitmap() const**

Returns the bitmap used for the wizard.

### **wxWizard::GetCurrentPage**

**wxWizardPage\* GetCurrentPage() const**

Get the current page while the wizard is running. `NULL` is returned if *RunWizard()* (p. 1862) is not being executed now.

### **wxWizard::GetPageAreaSizer**

**virtual wxSizer\* GetPageAreaSizer() const**

Returns pointer to page area sizer. The wizard is laid out using sizers and the page area

sizer is the place-holder for the pages. All pages are resized before being shown to match the wizard page area.

Page area sizer has a minimal size that is the maximum of several values. First, all pages (or other objects) added to the sizer. Second, all pages reachable by repeatedly applying *wxWizardPage::GetNext* (p. 1865) to any page inserted into the sizer. Third, the minimal size specified using *SetPageSize* (p. 1862) and *FitToPage* (p. 1860). Fourth, the total wizard height may be increased to accommodate the bitmap height. Fifth and finally, wizards are never smaller than some built-in minimal size to avoid wizards that are too small.

The caller can use *wxSizer::SetMinSize* (p. 1454) to enlarge it beyond the minimal size. If *wxRESIZE\_BORDER* was passed to constructor, user can resize wizard and consequently the page area (but not make it smaller than the minimal size).

It is recommended to add the first page to the page area sizer. For simple wizards, this will enlarge the wizard to fit the biggest page. For non-linear wizards, the first page of every separate chain should be added. Caller-specified size can be accomplished using *wxSizer::SetMinSize* (p. 1454).

Adding pages to the page area sizer affects the default border width around page area that can be altered with *SetBorder* (p. 1862).

### **wxWizard::GetPageSize**

**wxSize GetPageSize() const**

Returns the size available for the pages.

### **wxWizard::HasNextPage**

**virtual bool HasNextPage(wxWizardPage \*page)**

Return `true` if this page is not the last one in the wizard. The base class version implements this by calling *page->GetNext* (p. 1865) but this could be undesirable if, for example, the pages are created on demand only.

#### **See also**

*HasPrevPage* (p. 1861)

### **wxWizard::HasPrevPage**

**virtual bool HasPrevPage(wxWizardPage \*page)**

Returns `true` if this page is not the last one in the wizard. The base class version implements this by calling *page->GetPrev* (p. 1865) but this could be undesirable if, for example, the pages are created on demand only.

#### **See also**

*HasNextPage* (p. 1861)

**wxWizard::RunWizard****bool RunWizard(wxWizardPage\* firstPage)**

Executes the wizard starting from the given page, returning `true` if it was successfully finished or `false` if user cancelled it. The *firstPage* can not be `NULL`.

**wxWizard::SetBitmap****void SetBitmap(const wxBitmap& bitmap)**

Sets the bitmap used for the wizard.

**wxWizard::SetPageSize****void SetPageSize(const wxSize& sizePage)**

This method is obsolete, use *GetPageAreaSizer* (p. 1860) instead.

Sets the minimal size to be made available for the wizard pages. The wizard will take into account the size of the bitmap (if any) itself. Also, the wizard will never be smaller than the default size.

The recommended way to use this function is to lay out all wizard pages using the sizers (even though the wizard is not resizeable) and then use *wxSizer::CalcMin* (p. 1448) in a loop to calculate the maximum of minimal sizes of the pages and pass it to *SetPageSize()*.

**wxWizard::SetBorder****void SetBorder(int border)**

Sets width of border around page area. Default is zero. For backward compatibility, if there are no pages in *GetPageAreaSizer* (p. 1860), the default is 5 pixels.

If there is a five point border around all controls in a page and the border around page area is left as zero, a five point white space along all dialog borders will be added to the control border in order to space page controls ten points from the dialog border and non-page controls.

**wxWizardEvent**

*wxWizardEvent* class represents an event generated by the *wizard* (p. 1857): this event is first sent to the page itself and, if not processed there, goes up the window hierarchy as usual.

**Derived from***wxNotifyEvent* (p. 1146)*wxCommandEvent* (p. 250)*wxEvent* (p. 572)*wxObject* (p. 1148)

**Include files**

<wx/wizard.h>

**Event table macros**

To process input from a wizard dialog, use these event handler macros to direct input to member functions that take a `wxWizardEvent` argument.

**EVT\_WIZARD\_PAGE\_CHANGED(id, func)**      The page has been just changed (this event can not be vetoed).

**EVT\_WIZARD\_PAGE\_CHANGING(id, func)**      The page is being changed (this event can be vetoed).

**EVT\_WIZARD\_CANCEL(id, func)**      The user attempted to cancel the wizard (this event may also be vetoed).

**EVT\_WIZARD\_HELP(id, func)**      The wizard help button was pressed.

**EVT\_WIZARD\_FINISHED(id, func)**      The wizard finished button was pressed.

**See also**

*wxWizard* (p. 1857), *wxWizard sample* (p. 2040)

**wxWizardEvent::wxWizardEvent**

**wxWizardEvent(wxEvtType type = wxEVT\_NULL, int id = -1, bool direction = true)**

Constructor. It is not normally used by the user code as the objects of this type are constructed by `wxWizard`.

**wxWizardEvent::GetDirection**

**bool GetDirection() const**

Return the direction in which the page is changing: for `EVT_WIZARD_PAGE_CHANGING`, return `true` if we're going forward or `false` otherwise and for `EVT_WIZARD_PAGE_CHANGED` return `true` if we came from the previous page and `false` if we returned from the next one.

**wxWizardEvent::GetPage**

**wxWizardPage\* GetPage() const**

Returns the *wxWizardPage* (p. 1863) which was active when this event was generated.

**wxWizardPage**

`wxWizardPage` is one of the screens in `wxWizard` (p. 1857): it must know what are the following and preceding pages (which may be `NULL` for the first/last page). Except for this extra knowledge, `wxWizardPage` is just a panel, so the controls may be placed directly on it in the usual way.

This class allows the programmer to decide the order of pages in the wizard dynamically (during run-time) and so provides maximal flexibility. Usually, however, the order of pages is known in advance in which case `wxWizardPageSimple` (p. 1865) class is enough and it is simpler to use.

### Virtual functions to override

To use this class, you must override `GetPrev` (p. 1865) and `GetNext` (p. 1865) pure virtual functions (or you may use `wxWizardPageSimple` (p. 1865) instead).

`GetBitmap` (p. 1865) can also be overridden, but this should be very rarely needed.

### Derived from

`wxPanel` (p. 1170)  
`wxWindow` (p. 1795)  
`wxEvtHandler` (p. 576)  
`wxObject` (p. 1148)

### Include files

<wx/wizard.h>

### See also

`wxWizard` (p. 1857), `wxWizard sample` (p. 2040)

## `wxWizardPage::wxWizardPage`

**`wxWizardPage(wxWizard* parent, const wxBitmap& bitmap = wxNullBitmap, const wxChar *resource = NULL)`**

Constructor accepts an optional bitmap which will be used for this page instead of the default one for this wizard (note that all bitmaps used should be of the same size). Notice that no other parameters are needed because the wizard will resize and reposition the page anyhow.

### Parameters

*parent*

The parent wizard

*bitmap*

The page-specific bitmap if different from the global one



*resource*

Load the page from the specified resource if non-NULL

### **wxWizardPage::GetPrev**

#### **wxWizardPage\* GetPrev() const**

Get the page which should be shown when the user chooses the "Back" button: if `NULL` is returned, this button will be disabled. The first page of the wizard will usually return `NULL` from here, but the others will not.

#### **See also**

*GetNext* (p. 1865)

### **wxWizardPage::GetNext**

#### **wxWizardPage\* GetNext() const**

Get the page which should be shown when the user chooses the "Next" button: if `NULL` is returned, this button will be disabled. The last page of the wizard will usually return `NULL` from here, but the others will not.

#### **See also**

*GetPrev* (p. 1865)

### **wxWizardPage::GetBitmap**

#### **wxBitmap GetBitmap() const**

This method is called by `wxWizard` to get the bitmap to display alongside the page. By default, `m_bitmap` member variable which was set in the *constructor* (p. 1864).

If the bitmap was not explicitly set (i.e. if `wxNullBitmap` is returned), the default bitmap for the wizard should be used.

The only cases when you would want to override this function is if the page bitmap depends dynamically on the user choices, i.e. almost never.

## **wxWizardPageSimple**

`wxWizardPageSimple` is the simplest possible *wxWizardPage* (p. 1863) implementation: it just returns the pointers given to its constructor from `GetNext()` and `GetPrev()` functions.

This makes it very easy to use the objects of this class in the wizards where the pages order is known statically - on the other hand, if this is not the case you must derive your own class from *wxWizardPage* (p. 1863) instead.

#### **Derived from**

*wxWizardPage* (p. 1863)

*wxPanel* (p. 1170)

*wxWindow* (p. 1795)

*wxEvtHandler* (p. 576)

*wxObject* (p. 1148)

### Include files

<wx/wizard.h>

### See also

*wxWizard* (p. 1857), *wxWizard sample* (p. 2040)

## **wxWizardPageSimple::wxWizardPageSimple**

**wxWizardPageSimple**(*wxWizard\** parent = *NULL*, *wxWizardPage\** prev = *NULL*,  
*wxWizardPage\** next = *NULL*, **const** *wxBitmap&* bitmap = *wxNullBitmap*)

Constructor takes the previous and next pages. They may be modified later by *SetPrev()* (p. 1866) or *SetNext()* (p. 1866).

## **wxWizardPageSimple::SetPrev**

**void** **SetPrev**(*wxWizardPage\** prev)

Sets the previous page.

## **wxWizardPageSimple::SetNext**

**void** **SetNext**(*wxWizardPage\** next)

Sets the next page.

## **wxWizardPageSimple::Chain**

**static void** **Chain**(*wxWizardPageSimple\** first, *wxWizardPageSimple\** second)

A convenience function to make the pages follow each other.

Example:

```
wxRadioboxPage *page3 = new wxRadioboxPage(wizard);  
wxValidationPage *page4 = new wxValidationPage(wizard);  
  
wxWizardPageSimple::Chain(page3, page4);
```

## **wxXmlDocument**

This class holds XML data/document as parsed by XML parser in the root node.

`wxXmlDocument` internally uses the `expat` library which comes with `wxWidgets` to parse the given stream.

A simple example of using XML classes is:

```
wxXmlDocument doc;
if (!doc.Load(wxT("myfile.xml")))
    return false;

// start processing the XML file
if (doc.GetRoot()->GetName() != wxT("myroot-node"))
    return false;

wxXmlNode *child = doc.GetRoot()->GetChildren();
while (child) {

    if (child->GetName() == wxT("tag1")) {

        // process text enclosed by <tag1></tag1>
        wxString content = child->GetNodeContent();

        ...

        // process properties of <tag1>
        wxString propvalue1 =
            child->GetPropVal(wxT("prop1"),
                              wxT("default-value"));
        wxString propvalue2 =
            child->GetPropVal(wxT("prop2"),
                              wxT("default-value"));

        ...

    } else if (child->GetName() == wxT("tag2")) {

        // process tag2 ...

    }

    child = child->GetNext();
}
```

**Note:** if you want to preserve the original formatting of the loaded file including whitespaces and indentation, you need to turn off whitespace-only textnode removal and automatic indentation:

```
wxXmlDocument doc;
doc.Load(wxT("myfile.xml"), wxT("UTF-8"),
wxXMLDOC_KEEP_WHITESPACE_NODES);

// myfile2.xml will be indentic to myfile.xml saving it this way:
doc.Save(wxT("myfile2.xml"), wxXML_NO_INDENTATION);
```

Using default parameters, you will get a reformatted document which in general is different

from the original loaded content:

```
wxXmlDocument doc;  
doc.Load(wxT("myfile.xml"));  
doc.Save(wxT("myfile2.xml")); // myfile2.xml != myfile.xml
```

### Derived from

*wxObject* (p. 1148)

### Include files

<wx/xml/xml.h>

### See also

*wxXmlNode* (p. 1871), *wxXmlProperty* (p. 1877)

## wxXmlDocument::wxXmlDocument

**wxXmlDocument()**

**wxXmlDocument(const wxString& filename, const wxString& encoding = wxT("UTF-8"), int flags = wxXMLDOC\_NONE)**

Loads the given *filename* using the given encoding. See *Load* (p. 1869).

**wxXmlDocument(wxInputStream& stream, const wxString& encoding = wxT("UTF-8"), int flags = wxXMLDOC\_NONE)**

Loads the XML document from given stream using the given encoding. See *Load* (p. 1869).

**wxXmlDocument(const wxXmlDocument& doc)**

Copy constructor. Deep copies all the XML tree of the given document.

## wxXmlDocument::~wxXmlDocument

**~wxXmlDocument()**

Virtual destructor. Frees the document root node.

## wxXmlDocument::DetachRoot

**wxXmlNode\* DetachRoot()**

Detaches the document root node and returns it. The document root node will be set to `NULL` and thus *IsOk* (p. 1869) will return `false` after calling this function.

Note that the caller is responsible for deleting the returned node in order to avoid memory

leaks.

### **wxXmlDocument::GetEncoding**

#### **wxString GetEncoding() const**

Returns encoding of in-memory representation of the document (same as passed to *Load* (p. 1869) or constructor, defaults to UTF-8).

NB: this is meaningless in Unicode build where data are stored as `wchar_t*`.

### **wxXmlDocument::GetFileEncoding**

#### **wxString GetFileEncoding() const**

Returns encoding of document (may be empty).

Note: this is the encoding original file was saved in, **not** the encoding of in-memory representation!

### **wxXmlDocument::GetRoot**

#### **wxXmlNode\* GetRoot() const**

Returns the root node of the document.

### **wxXmlDocument::GetVersion**

#### **wxString GetVersion() const**

Returns the version of document. This is the value in the `<?xml version="1.0"?>` header of the XML document. If the version property was not explicitly given in the header, this function returns an empty string.

### **wxXmlDocument::IsOk**

#### **bool IsOk() const**

Returns `true` if the document has been loaded successfully.

### **wxXmlDocument::Load**

**bool Load(const wxString& filename, const wxString& encoding = wxT("UTF-8"), int flags = wxXMLDOC\_NONE)**

Parses *filename* as an xml document and loads its data.

If *flags* does not contain `wxXMLDOC_KEEP_WHITESPACE_NODES`, then, while loading, all nodes of type `wxXML_TEXT_NODE` (see *wxXmlNode* (p. 1871)) are automatically skipped if they contain whitespaces only. The removal of these nodes makes the load

process slightly faster and requires less memory however makes impossible to recreate exactly the loaded text with a `Save` (p. 1870) call later. Read the initial description of this class for more info.

Returns `true` on success, `false` otherwise.

**bool Load(wxInputStream& stream, const wxString& encoding = wxT("UTF-8"), int flags = wxXMLDOC\_NONE)**

Like above but takes the data from given input stream.

### **wxXmlDocument::Save**

**bool Save(const wxString& filename, int indentstep = 1) const**

Saves XML tree creating a file named with given string.

If `indentstep` is greater than or equal to zero, then, while saving, an automatic indentation is added with steps composed by `indentstep` spaces. If `indentstep` is `wxXML_NO_INDENTATION`, then, automatic indentation is turned off.

**bool Save(wxOutputStream& stream, int indentstep = 1) const**

Saves XML tree in the given output stream. See other overload for a description of `indentstep`.

### **wxXmlDocument::SetEncoding**

**void SetEncoding(const wxString& enc)**

Sets the encoding of the document.

### **wxXmlDocument::SetFileEncoding**

**void SetFileEncoding(const wxString& encoding)**

Sets the encoding of the file which will be used to save the document.

### **wxXmlDocument::SetRoot**

**void SetRoot(wxXmlNode\* node)**

Sets the root node of this document. Deletes previous root node. Use `DetachRoot` (p. 1868) and then `SetRoot` (p. 1870) if you want to replace the root node without deleting the old document tree.

### **wxXmlDocument::SetVersion**

**void SetVersion(const wxString& version)**

Sets the version of the XML file which will be used to save the document.

**wxXmlDocument::operator=****wxXmlDocument& operator operator=(const wxXmlDocument& doc)**

Deep copies the given document.

**wxXmlNode**

Represents a node in an XML document. See *wxXmlDocument* (p. 1866).

Node has a name and may have content and properties. Most common node types are `wxXML_TEXT_NODE` (name and properties are irrelevant) and `wxXML_ELEMENT_NODE` (e.g. in `<title>hi</title>` there is an element with name="title", irrelevant content and one child (`wxXML_TEXT_NODE` with content="hi").

If `wxUSE_UNICODE` is 0, all strings are encoded in the encoding given to *wxXmlDocument::Load* (p. 1869) (default is UTF-8).

**Derived from**

No base class

**Include files**

<wx/xml/xml.h>

**Constants**

The following are the node types supported by *wxXmlNode* (p. 1871):

```
enum wxXmlNodeType
{
    wxXML_ELEMENT_NODE,
    wxXML_ATTRIBUTE_NODE,
    wxXML_TEXT_NODE,
    wxXML_CDATA_SECTION_NODE,
    wxXML_ENTITY_REF_NODE,
    wxXML_ENTITY_NODE,
    wxXML_PI_NODE,
    wxXML_COMMENT_NODE,
    wxXML_DOCUMENT_NODE,
    wxXML_DOCUMENT_TYPE_NODE,
    wxXML_DOCUMENT_FRAG_NODE,
    wxXML_NOTATION_NODE,
    wxXML_HTML_DOCUMENT_NODE
}
```

**See also**

*wxXmlDocument* (p. 1866), *wxXmlProperty* (p. 1877)

**wxXmlNode::wxXmlNode**

**wxXmlNode**(**wxXmlNode\*** *parent*, **wxXmlNodeType** *type*, **const wxString&** *name*, **const wxString&** *content* = *wxEmptyString*, **wxXmlProperty\*** *props* = *NULL*, **wxXmlNode\*** *next* = *NULL*)

**Parameters**

*parent*

The parent node. Can be *NULL*.

*type*

One of the *wxXmlNodeType* enumeration value.

*name*

The name of the node. This is the string which appears between angular brackets.

*content*

The content of the node. Only meaningful when *type* is *wxXML\_TEXT\_NODE* or *wxXML\_CDATA\_SECTION\_NODE*.

*props*

If not *NULL*, this *wxXmlProperty* object and its eventual siblings are attached to the node.

*next*

If not *NULL*, this node and its eventual siblings are attached to the node.

**wxXmlNode(const wxXmlNode& node)**

Copy constructor. Note that this does NOT copy syblings and parent pointer, i.e. *GetParent()* (p. 1874) and *GetNext()* (p. 1874) will return *NULL* after using copy ctor and are never unmodified by operator=.

On the other hand, it DOES copy children and properties.

**wxXmlNode**(**wxXmlNodeType** *type*, **const wxString&** *name*, **const wxString&** *content* = *wxEmptyString*)

A simplified version of the first constructor form.

**wxXmlNode::~~wxXmlNode**

**~wxXmlNode()**

The virtual destructor. Deletes attached children and properties.



**wxXmlNode::AddChild****void AddChild(wxXmlNode\* child)**

Adds the given node as child of this node. To attach a second children to this node, use the *SetNext()* (p. 1876) function of the *child* node.

**wxXmlNode::AddProperty****void AddProperty(const wxString& name, const wxString& value)**

Appends a property with given *name* and *value* to the list of properties for this node.

**void AddProperty(wxXmlProperty\* prop)**

Appends the given property to the list of properties for this node.

**wxXmlNode::DeleteProperty****bool DeleteProperty(const wxString& name)**

Removes the first properties which has the given *name* from the list of properties for this node.

**wxXmlNode::GetChildren****wxXmlNode\* GetChildren() const**

Returns the first child of this node. To get a pointer to the second child of this node (if it does exist), use the *GetNext()* (p. 1874) function on the returned value.

**wxXmlNode::GetContent****wxString GetContent() const**

Returns the content of this node. Can be an empty string. Be aware that for nodes of type `wxXML_ELEMENT_NODE` (the most used node type) the content is an empty string. See *GetNodeContent()* (p. 1873) for more details.

**wxXmlNode::GetDepth****int GetDepth(wxXmlNode\* grandparent = NULL) const**

Returns the number of nodes which separate this node from *grandparent*.

This function searches only the parents of this node until it finds *grandparent* or the `NULL` node (which is the parent of non-linked nodes or the parent of *awwxXmlDocument* (p. 1866)'s root node).

**wxXmlNode::GetNodeContent**

**wxString GetNodeContent() const**

Returns the content of the first child node of type `wxXML_TEXT_NODE` or `wxXML_CDATA_SECTION_NODE`. This function is very useful since the XML snippet "`<tagname>tagcontent</tagname>`" is represented by expat with the following tag tree:

```
wxXML_ENTITY_NODE name="tagname", content=""
|-- wxXML_TEXT_NODE name="", content="tagcontent"
```

or eventually:

```
wxXML_ENTITY_NODE name="tagname", content=""
|-- wxXML_CDATA_SECTION_NODE name="", content="tagcontent"
```

An empty string is returned if the node has no children of type `wxXML_TEXT_NODE` or `wxXML_CDATA_SECTION_NODE`, or if the content of the first child of such types is empty.

**wxXmlNode::GetName****wxString GetName() const**

Returns the name of this node. Can be an empty string (e.g. for nodes of type `wxXML_TEXT_NODE` or `wxXML_CDATA_SECTION_NODE`).

**wxXmlNode::GetNext****wxXmlNode\* GetNext() const**

Returns a pointer to the sibling of this node or `NULL` if there are no siblings.

**wxXmlNode::GetParent****wxXmlNode\* GetParent() const**

Returns a pointer to the parent of this node or `NULL` if this node has no parent.

**wxXmlNode::GetPropVal****bool GetPropVal(const wxString& propName, wxString\* value) const**

Returns `true` if a property named *propName* could be found. The value of that property is saved in *value* (which must not be `NULL`).

**wxString GetPropVal(const wxString& propName, const wxString& defaultVal) const**

Returns the value of the property named *propName* if it does exist. If it does not exist, the *defaultVal* is returned.

**wxXmlNode::GetProperties**

**wxXmlProperty \* GetProperties() const**

Return a pointer to the first property of this node.

**wxXmlNode::GetType**

**wxXmlNodeType GetType() const**

Returns the type of this node.

**wxXmlNode::HasProp**

**bool HasProp(const wxString& propName) const**

Returns `true` if this node has a property named *propName*.

**wxXmlNode::InsertChild**

**bool InsertChild(wxXmlNode\* child, wxXmlNode\* followingNode)**

Inserts the *child* node immediately before *followingNode* in the children list. If *followingNode* is `NULL`, then *child* is prepended to the list of children and becomes the first child of this node. Returns `true` if *followingNode* has been found and the *child* node has been inserted.

**wxXmlNode::InsertChildAfter**

**bool InsertChildAfter(wxXmlNode\* child, wxXmlNode\* precedingNode)**

Inserts the *child* node immediately after *precedingNode* in the children list. Returns `true` if *precedingNode* has been found and the *child* node has been inserted.

#### Parameters

*child*

Node to insert.

*precedingNode*

The node to insert *child* after. As a special case, this can be `NULL` if this node has no children yet -- in that case, *child* will become this node's only child node.

This function is new since wxWidgets version 2.8.8

**wxXmlNode::IsWhitespaceOnly**

**bool IsWhitespaceOnly() const**

Returns `true` if the content of this node is a string containing only whitespaces (spaces, tabs, new lines, etc). Note that this function is locale-independent since the parsing of XML

documents must always produce the exact same tree regardless of the locale it runs under.

### **wxXmlNode::RemoveChild**

**bool RemoveChild(wxXmlNode\* child)**

Removes the given node from the children list. Returns `true` if the node was found and removed or `false` if the node could not be found.

Note that the caller is responsible for deleting the removed node in order to avoid memory leaks.

### **wxXmlNode::SetChildren**

**void SetChildren(wxXmlNode\* child)**

Sets as first child the given node. The caller is responsible to delete any previously present children node.

### **wxXmlNode::SetContent**

**void SetContent(const wxString& con)**

Sets the content of this node.

### **wxXmlNode::SetName**

**void SetName(const wxString& name)**

Sets the name of this node.

### **wxXmlNode::SetNext**

**void SetNext(wxXmlNode\* next)**

Sets as sibling the given node. The caller is responsible to delete any previously present sibling node.

### **wxXmlNode::SetParent**

**void SetParent(wxXmlNode\* parent)**

Sets as parent the given node. The caller is responsible to delete any previously present parent node.

### **wxXmlNode::SetProperties**

**void SetProperties(wxXmlProperty\* prop)**

Sets as first property the given `wxXmlProperty` object. The caller is responsible to delete any previously present properties attached to this node.

### **wxXmlNode::SetType**

**void SetType(wxXmlNodeType type)**

Sets the type of this node.

### **wxXmlNode::operator=**

**wxXmlNode& operator=(const wxXmlNode& node)**

See the copy constructor for more info.

## **wxXmlProperty**

Represents a node property.

Example: in ``, `src` is property with value `"hello.gif"` and `id` is a property with value `"3"`.

### **Derived from**

No base class

### **Include files**

`<wx/xml/xml.h>`

### **See also**

*wxXmlDocument* (p. 1866), *wxXmlNode* (p. 1871)

### **wxXmlProperty::wxXmlProperty**

**wxXmlProperty()**

**wxXmlProperty(const wxString& name, const wxString& value, wxXmlProperty\* next = NULL)**

Creates the property with given *name* and *value*. If *next* is not `NULL`, then sets it as sibling of this property.

### **wxXmlProperty::~~wxXmlProperty**

**~wxXmlProperty()**

The virtual destructor.

**wxXmlProperty::GetName****wxString GetName() const**

Returns the name of this property.

**wxXmlProperty::GetNext****wxXmlProperty\* GetNext() const**

Returns the sibling of this property or NULL if there are no siblings.

**wxXmlProperty::GetValue****wxString GetValue() const**

Returns the value of this property.

**wxXmlProperty::SetName****void SetName(const wxString& name)**

Sets the name of this property.

**wxXmlProperty::SetNext****void SetNext(wxXmlProperty\* next)**

Sets the sibling of this property.

**wxXmlProperty::SetValue****void SetValue(const wxString& value)**

Sets the value of this property.

**wxXmlResource**

This is the main class for interacting with the XML-based resource system.

The class holds XML resources from one or more .xml files, binary files or zip archive files.

See *XML-based resource system overview* (p. 2106) for details.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/xrc/xmlres.h>

## Constants

```
enum wxXmlResourceFlags
{
    wxXRC_USE_LOCALE      = 1,
    wxXRC_NO_SUBCLASSING  = 2,
    wxXRC_NO_RELOADING    = 4
};
```

## **wxXmlResource::wxXmlResource**

**wxXmlResource(const wxString& filemask, int flags = wxXRC\_USE\_LOCALE,  
const wxString& domain = wxEmptyString)**

Constructor.

*filemask*

The XRC file, archive file, or wildcard specification that will be used to load all resource files inside a zip archive.

*flags*

wxXRC\_USE\_LOCALE: translatable strings will be translated via \_().  
wxXRC\_NO\_SUBCLASSING: subclass property of object nodes will be ignored  
(useful for previews in XRC editors).

*domain*

The name of the gettext catalog to search for translatable strings. By default all loaded catalogs will be searched. This provides a way to allow the strings to only come from a specific catalog.

**wxXmlResource(int flags = wxXRC\_USE\_LOCALE, const wxString& domain =  
wxEmptyString)**

Constructor.

*flags*

wxXRC\_USE\_LOCALE: translatable strings will be translated via \_().  
wxXRC\_NO\_SUBCLASSING: subclass property of object nodes will be ignored  
(useful for previews in XRC editors). wxXRC\_NO\_RELOADING will prevent the  
XRC files from being reloaded from disk in case they have been modified there since  
being last loaded (may slightly speed up loading them).

*domain*

The name of the gettext catalog to search for translatable strings. By default all loaded catalogs will be searched. This provides a way to allow the strings to only come from a specific catalog.

**wxXmlResource::~wxXmlResource****~wxXmlResource()**

Destructor.

**wxXmlResource::AddHandler****void AddHandler(wxXmlResourceHandler\* handler)**

Initializes only a specific handler (or custom handler). Convention says that the handler name is equal to the control's name plus 'XmlHandler', for example wxTextCtrlXmlHandler, wxHtmlWindowXmlHandler. The XML resource compiler (wxxrc) can create include file that contains initialization code for all controls used within the resource. Note that this handler should be allocated on the heap, since it will be delete by *ClearHandlers* (p. 1880) later.

**wxXmlResource::AttachUnknownControl****bool AttachUnknownControl(const wxString& name, wxWindow\* control, wxWindow\* parent = NULL)**

Attaches an unknown control to the given panel/window/dialog. Unknown controls are used in conjunction with <object class="unknown">.

**wxXmlResource::ClearHandlers****void ClearHandlers()**

Removes all handlers and deletes them (this means that any handlers added using *AddHandler* (p. 1880) must be allocated on the heap).

**wxXmlResource::CompareVersion****int CompareVersion(int major, int minor, int release, int revision) const**

Compares the XRC version to the argument. Returns -1 if the XRC version is less than the argument, +1 if greater, and 0 if they equal.

**wxXmlResource::Get****wxXmlResource\* Get()**

Gets the global resources object or creates one if none exists.

**wxXmlResource::GetFlags****int GetFlags()**

Returns flags, which may be a bitlist of wxXRC\_USE\_LOCALE and



`wxXRC_NO_SUBCLASSING`.

### **wxXmlResource::GetVersion**

**long GetVersion() const**

Returns version information ( $a.b.c.d = d + 256*c + 2562*b + 2563*a$ ).

### **wxXmlResource::GetXRCID**

**int GetXRCID(const wxChar\* str\_id, int value\_if\_not\_found = -2)**

Returns a numeric ID that is equivalent to the string ID used in an XML resource. If an unknown *str\_id* is requested (i.e. other than `wxid_XXX` or integer), a new record is created which associates the given string with a number. If *value\_if\_not\_found* is `wxid_none`, the number is obtained via `wxNewId()` (p. 1953). Otherwise *value\_if\_not\_found* is used. Macro `XRCID(name)` is provided for convenient use in event tables.

### **wxXmlResource::InitAllHandlers**

**void InitAllHandlers()**

Initializes handlers for all supported controls/windows. This will make the executable quite big because it forces linking against most of the `wxWidgets` library.

### **wxXmlResource::Load**

**bool Load(const wxString& filemask)**

Loads resources from XML files that match given filemask. This method understands VFS (see `filesystem.h`).

### **wxXmlResource::LoadBitmap**

**wxBitmap LoadBitmap(const wxString& name)**

Loads a bitmap resource from a file.

### **wxXmlResource::LoadDialog**

**wxDialog\* LoadDialog(wxWindow\* parent, const wxString& name)**

Loads a dialog. *dlg* points to a parent window (if any).

**bool LoadDialog(wxDialog\* dlg, wxWindow\* parent, const wxString& name)**

Loads a dialog. *dlg* points to parent window (if any).

This form is used to finish creation of an already existing instance (the main reason for this is that you may want to use derived class with a new event table).

Example:

```
MyDialog dlg;  
wxTheXmlResource->LoadDialog(&dlg, mainFrame, "my_dialog");  
dlg.ShowModal();
```

### **wxXmlResource::LoadFrame**

**bool LoadFrame(wxFrame\* frame, wxWindow\* parent, const wxString& name)**

Loads a frame.

### **wxXmlResource::LoadIcon**

**wxIcon LoadIcon(const wxString& name)**

Loads an icon resource from a file.

### **wxXmlResource::LoadMenu**

**wxMenu\* LoadMenu(const wxString& name)**

Loads menu from resource. Returns NULL on failure.

### **wxXmlResource::LoadMenuBar**

**wxMenuBar\* LoadMenuBar(wxWindow\* parent, const wxString& name)**

Loads a menubar from resource. Returns NULL on failure.

**wxMenuBar\* LoadMenuBar(const wxString& name)**

Loads a menubar from resource. Returns NULL on failure.

### **wxXmlResource::LoadPanel**

**wxPanel\* LoadPanel(wxWindow\* parent, const wxString& name)**

Loads a panel. *panel* points to parent window (if any).

**bool LoadPanel(wxPanel\* panel, wxWindow\* parent, const wxString& name)**

Loads a panel. *panel* points to parent window (if any). This form is used to finish creation of an already existing instance.

### **wxXmlResource::LoadToolBar**

**wxToolBar\* LoadToolBar(wxWindow\* parent, const wxString& name)**

Loads a toolbar.

**wxXmlResource::Set****wxXmlResource\* Set(wxXmlResource\* res)**

Sets the global resources object and returns a pointer to the previous one (may be NULL).

**wxXmlResource::SetFlags****void SetFlags(int flags)**

Sets flags (bitlist of wxXRC\_USE\_LOCALE and wxXRC\_NO\_SUBCLASSING).

**wxXmlResource::Unload****bool Unload(const wxString& filename)**

This function unloads a resource previously loaded by *Load()* (p. 1881).

Returns `true` if the resource was successfully unloaded and `false` if it hasn't been found in the list of loaded resources.

**wxXmlResource::GetDomain****wxChar\* GetDomain()**

Returns the domain (message catalog) that will be used to load translatable strings in the XRC.

**wxXmlResource::SetDomain****wxChar\* SetDomain(const wxChar\* domain)**

Sets the domain (message catalog) that will be used to load translatable strings in the XRC.

**wxXmlResourceHandler**

wxXmlResourceHandler is an abstract base class for resource handlers capable of creating a control from an XML node.

See *XML-based resource system overview* (p. 2106) for details.

**Derived from**

*wxObject* (p. 1148)

**Include files**

<wx/xrc/xmlres.h>

**wxXmlResourceHandler::wxXmlResourceHandler****wxXmlResourceHandler()**

Default constructor.

**wxXmlResourceHandler::~~wxXmlResourceHandler****~wxXmlResourceHandler()**

Destructor.

**wxXmlResourceHandler::AddStyle****void AddStyle(const wxString& name, int value)**

Add a style flag (e.g. wxMB\_DOCKABLE) to the list of flags understood by this handler.

**wxXmlResourceHandler::AddWindowStyles****void AddWindowStyles()**

Add styles common to all wxWindow-derived classes.

**wxXmlResourceHandler::CanHandle****bool CanHandle(wxXmlNode\* node)**

Returns true if it understands this node and can create a resource from it, false otherwise.

**Note**

You must **not** call any wxXmlResourceHandler methods except *IsOfClass* (p. 1887) from this method! The instance is not yet initialized with node data at the time *CanHandle* is called and it is only safe to operate on *node* directly or to call *IsOfClass*.

**wxXmlResourceHandler::CreateChildren****void CreateChildren(wxObject\* parent, bool this\_hnd\_only = false)**

Creates children.

**wxXmlResourceHandler::CreateChildrenPrivately****void CreateChildrenPrivately(wxObject\* parent, wxXmlNode\* rootnode = NULL)**

Helper function.

**wxXmlResourceHandler::CreateResFromNode**

**wxObject\* CreateResFromNode(wxXmlNode\* node, wxObject\* parent, wxObject\* instance = NULL)**

Creates a resource from a node.

### **wxXmlResourceHandler::CreateResource**

**wxObject\* CreateResource(wxXmlNode\* node, wxObject\* parent, wxObject\* instance)**

Creates an object (menu, dialog, control, ...) from an XML node. Should check for validity. *parent* is a higher-level object (usually window, dialog or panel) that is often necessary to create the resource. If **instance** is non-NULL it should not create a new instance via 'new' but should rather use this one, and call its Create method.

### **wxXmlResourceHandler::DoCreateResource**

**wxObject\* DoCreateResource()**

Called from CreateResource after variables were filled.

### **wxXmlResourceHandler::GetBitmap**

**wxBitmap GetBitmap(const wxString& param = wxT("bitmap"), wxSize size = wxDefaultSize)**

Gets a bitmap.

### **wxXmlResourceHandler::GetBool**

**bool GetBool(const wxString& param, bool defaultv = false)**

Gets a bool flag (1, t, yes, on, true are true, everything else is false).

### **wxXmlResourceHandler::GetColour**

**wxColour GetColour(const wxString& param, const wxColour& default = wxNullColour)**

Gets colour in HTML syntax (#RRGGBB).

### **wxXmlResourceHandler::GetCurFileSystem**

**wxFileSystem& GetCurFileSystem()**

Returns the current file system.

### **wxXmlResourceHandler::GetDimension**

**wxCoord GetDimension(const wxString& param, wxCoord defaultv = 0)**

Gets a dimension (may be in dialog units).

#### **wxXmlResourceHandler::GetFont**

**wxFont** GetFont(const wxString& param = wxT("font"))

Gets a font.

#### **wxXmlResourceHandler::GetID**

**int** GetID()

Returns the XRCID.

#### **wxXmlResourceHandler::GetIcon**

**wxIcon** GetIcon(const wxString& param = wxT("icon"), **wxSize** size = wxDefaultSize)

Returns an icon.

#### **wxXmlResourceHandler::GetLong**

**long** GetLong(const wxString& param, **long** defaultv = 0)

Gets the integer value from the parameter.

#### **wxXmlResourceHandler::GetName**

**wxString** GetName()

Returns the resource name.

#### **wxXmlResourceHandler::GetNodeContent**

**wxString** GetNodeContent(wxXmlNode\* node)

Gets node content from wxXML\_ENTITY\_NODE.

#### **wxXmlResourceHandler::GetParamNode**

**wxXmlNode\*** GetParamNode(const wxString& param)

Finds the node or returns NULL.

#### **wxXmlResourceHandler::GetParamValue**

**wxString** GetParamValue(const wxString& param)

Finds the parameter value or returns the empty string.

**wxXmlResourceHandler::GetPosition****wxPoint** **GetPosition**(const **wxString&** *param* = *wxT("pos")*)

Gets the position (may be in dialog units).

**wxXmlResourceHandler::GetSize****wxSize** **GetSize**(const **wxString&** *param* = *wxT("size")*)

Gets the size (may be in dialog units).

**wxXmlResourceHandler::GetStyle****int** **GetStyle**(const **wxString&** *param* = *wxT("style")*, **int** *defaults* = 0)

Gets style flags from text in form "flag | flag2 | flag3 |..." Only understands flags added with `AddStyle`.

**wxXmlResourceHandler::GetText****wxString** **GetText**(const **wxString&** *param*)

Gets text from *param* and does some conversions:

- replaces `\n`, `\r`, `\t` by respective characters (according to C syntax)
- replaces `$` by `&` and `$$` by `$` (needed for `_File` to `&Filetranslation` because of XML syntax)
- calls `wxGetTranslations` (unless disabled in `wxXmlResource`)

**wxXmlResourceHandler::HasParam****bool** **HasParam**(const **wxString&** *param*)

Check to see if a parameter exists.

**wxXmlResourceHandler::IsOfClass****bool** **IsOfClass**(**wxXmlNode\*** *node*, const **wxString&** *classname*)

Convenience function. Returns true if the node has a property class equal to *classname*, e.g. `<object class="wxDialog">`.

**wxXmlResourceHandler::SetParentResource****void** **SetParentResource**(**wxXmlResource\*** *res*)

Sets the parent resource.

## **wxXmlResourceHandler::SetupWindow**

**void SetupWindow(wxWindow\* wnd)**

Sets common window options.

## **wxZipClassFactory**

Class factory for the zip archive format. See the base class for details.

### **Derived from**

*wxArchiveClassFactory* (p. 58)

### **Include files**

<wx/zipstrm.h>

### **See also**

*Archive formats such as zip* (p. 2223)

*Generic archive programming* (p. 2227)

*wxZipEntry* (p. 1888)

*wxZipInputStream* (p. 1895)

*wxZipOutputStream* (p. 1897)

## **wxZipEntry**

Holds the meta-data for an entry in a zip.

### **Derived from**

*wxArchiveEntry* (p. 62)

### **Include files**

<wx/zipstrm.h>

### **Data structures**

Constants for *Get/SetMethod* (p. 1893):

```
// Compression Method, only 0 (store) and 8 (deflate) are supported
here
//
enum wxZipMethod
{
    wxZIP_METHOD_STORE,
    wxZIP_METHOD_SHRINK,
    wxZIP_METHOD_REDUCE1,
    wxZIP_METHOD_REDUCE2,
    wxZIP_METHOD_REDUCE3,
    wxZIP_METHOD_REDUCE4,
```



```
wxZIP_METHOD_IMPLODE,  
wxZIP_METHOD_TOKENIZE,  
wxZIP_METHOD_DEFLATE,  
wxZIP_METHOD_DEFLATE64,  
wxZIP_METHOD_BZIP2 = 12,  
wxZIP_METHOD_DEFAULT = 0xffff  
};
```

Constants for *Get/SetSystemMadeBy* (p. 1894):

```
// Originating File-System.  
//  
// These are Pkware's values. Note that Info-zip disagree on some of  
// them,  
// most notably NTFS.  
//  
enum wxZipSystem  
{  
    wxZIP_SYSTEM_MSDOS,  
    wxZIP_SYSTEM_AMIGA,  
    wxZIP_SYSTEM_OPENVMS,  
    wxZIP_SYSTEM_UNIX,  
    wxZIP_SYSTEM_VM_CMS,  
    wxZIP_SYSTEM_ATARI_ST,  
    wxZIP_SYSTEM_OS2_HPFS,  
    wxZIP_SYSTEM_MACINTOSH,  
    wxZIP_SYSTEM_Z_SYSTEM,  
    wxZIP_SYSTEM_CPM,  
    wxZIP_SYSTEM_WINDOWS_NTFS,  
    wxZIP_SYSTEM_MVS,  
    wxZIP_SYSTEM_VSE,  
    wxZIP_SYSTEM_ACORN_RISC,  
    wxZIP_SYSTEM_VFAT,  
    wxZIP_SYSTEM_ALTERNATE_MVS,  
    wxZIP_SYSTEM_BEOS,  
    wxZIP_SYSTEM_TANDEM,  
    wxZIP_SYSTEM_OS_400  
};
```

Constants for *Get/SetExternalAttributes* (p. 1892):

```
// Dos/Win file attributes  
//  
enum wxZipAttributes  
{  
    wxZIP_A_RDONLY = 0x01,  
    wxZIP_A_HIDDEN = 0x02,  
    wxZIP_A_SYSTEM = 0x04,  
    wxZIP_A_SUBDIR = 0x10,  
    wxZIP_A_ARCH = 0x20,  
  
    wxZIP_A_MASK = 0x37  
};
```

Constants for *Get/SetFlags* (p. 1892):

```
// Values for the flags field in the zip headers
//
enum wxZipFlags
{
    wxZIP_ENCRYPTED           = 0x0001,
    wxZIP_DEFLATE_NORMAL     = 0x0000,    // normal compression
    wxZIP_DEFLATE_EXTRA      = 0x0002,    // extra compression
    wxZIP_DEFLATE_FAST       = 0x0004,    // fast compression
    wxZIP_DEFLATE_SUPERFAST  = 0x0006,    // superfast compression
    wxZIP_DEFLATE_MASK       = 0x0006,
    wxZIP_SUMS_FOLLOW        = 0x0008,    // crc and sizes come after
the data
    wxZIP_ENHANCED           = 0x0010,
    wxZIP_PATCH              = 0x0020,
    wxZIP_STRONG_ENC         = 0x0040,
    wxZIP_UNUSED             = 0x0F80,
    wxZIP_RESERVED          = 0xF000
};
```

### See also

*Archive formats such as zip* (p. 2223)

*wxZipInputStream* (p. 1895)

*wxZipOutputStream* (p. 1897)

*wxZipNotifier* (p. 1896)

### Field availability

When reading a zip from a stream that is seekable, *GetNextEntry()* (p. 1896) returns a fully populated *wxZipEntry* object except for *wxZipEntry::GetLocalExtra()* (p. 1893).

*GetLocalExtra()* becomes available when the entry is opened, either by calling *wxZipInputStream::OpenEntry* (p. 1896) or by making an attempt to read the entry's data.

For zips on *non-seekable* (p. 2228) streams, the following fields are always available when *GetNextEntry()* returns:

*GetDateTime* (p. 62)

*GetInternalFormat* (p. 63)

*GetInternalName* (p. 1892)

*GetFlags* (p. 1892)

*GetLocalExtra* (p. 1893)

*GetMethod* (p. 1893)

*GetName* (p. 63)

*GetOffset* (p. 63)

*IsDir* (p. 64)

The following fields are also usually available when *GetNextEntry()* returns, however, if the zip was also written to a non-seekable stream the zipper is permitted to store them after the entry's data. In that case they become available when the entry's data has been read to *Eof()*, or *CloseEntry()* (p. 65) has been called. (*GetFlags()* &

`wxZIP_SUMS_FOLLOW) != 0` indicates that one or more of these come after the data:

*GetCompressedSize* (p. 1891)

*GetCrc* (p. 1892)

*GetSize* (p. 63)

The following are stored at the end of the zip, and become available when the end of the zip has been reached, i.e. after *GetNextEntry()* returns `NULL` and *Eof()* is true:

*GetComment* (p. 1891)

*GetExternalAttributes* (p. 1892)

*GetExtra* (p. 1892)

*GetMode* (p. 1893)

*GetSystemMadeBy* (p. 1894)

*IsReadOnly* (p. 64)

*IsMadeByUnix* (p. 1894)

*IsText* (p. 1894)

### **wxZipEntry::wxZipEntry**

**wxZipEntry(const wxString& name = wxEmptyString, const wxDateTime& dt = wxDateTime::Now(), off\_t size = wxInvalidOffset)**

Constructor.

**wxZipEntry(const wxZipEntry& entry)**

Copy constructor.

### **wxZipEntry::Clone**

**wxZipEntry\* Clone() const**

Make a copy of this entry.

### **wxZipEntry::Get/SetComment**

**wxString GetComment() const**

**void SetComment(const wxString& comment)**

A short comment for this entry.

### **wxZipEntry::GetCompressedSize**

**off\_t GetCompressedSize() const**

The compressed size of this entry in bytes.

**wxZipEntry::GetCrc****wxUInt32 GetCrc() const**

CRC32 for this entry's data.

**wxZipEntry::Get/SetExternalAttributes****wxUInt32 GetExternalAttributes() const****void SetExternalAttributes(wxUInt32 attr)**

The low 8 bits are always the DOS/Windows file attributes for this entry. The values of these attributes are given in the enumeration `wxZipAttributes`.

The remaining bits can store platform specific permission bits or attributes, and their meaning depends on the value of `SetSystemMadeBy()` (p. 1894). If `IsMadeByUnix()` (p. 1894) is true then the high 16 bits are unix mode bits.

The following other accessors access these bits:

*IsReadOnly/SetIsReadOnly* (p. 64)

*IsDir/SetIsDir* (p. 64)

*Get/SetMode* (p. 1893)

**wxZipEntry::Get/SetExtra****const char\* GetExtra() const****size\_t GetExtraLen() const****void SetExtra(const char\* extra, size\_t len)**

The extra field from the entry's central directory record.

The extra field is used to store platform or application specific data. See Pkware's document 'appnote.txt' for information on its format.

**wxZipEntry::GetFlags****int GetFlags() const**

Returns a combination of the bits flags in the enumeration `wxZipFlags`.

**wxZipEntry::GetInternalName****wxString GetInternalName() const**

Returns the entry's filename in the internal format used within the archive. The name can include directory components, i.e. it can be a full path.

The names of directory entries are returned without any trailing path separator. This gives

a canonical name that can be used in comparisons.

**wxString GetInternalName(const wxString& name, wxPathFormat format = wxPATH\_NATIVE, bool\* pIsDir = NULL)**

A static member that translates a filename into the internal format used within the archive. If the third parameter is provided, the bool pointed to is set to indicate whether the name looks like a directory name (i.e. has a trailing path separator).

**See also**

*Looking up an archive entry by name* (p. 2225)

**wxZipEntry::Get/SetLocalExtra**

**const char\* GetLocalExtra() const**

**size\_t GetLocalExtraLen() const**

**void SetLocalExtra(const char\* extra, size\_t len)**

The extra field from the entry's local record.

The extra field is used to store platform or application specific data. See Pkware's document 'appnote.txt' for information on its format.

**wxZipEntry::Get/SetMethod**

**int GetMethod() const**

**void SetMethod(int method)**

The compression method. The enumeration `wxZipMethod` lists the possible values.

The default constructor sets this to `wxZIP_METHOD_DEFAULT`, which allows `wxZipOutputStream` (p. 1897) to choose the method when writing the entry.

**wxZipEntry::Get/SetMode**

**int GetMode() const**

If `IsMadeByUnix()` (p. 1894) is true then returns the unix permission bits stored in `GetExternalAttributes()` (p. 1892). Otherwise synthesises them from the DOS attributes.

**void SetMode(int mode)**

Sets the DOS attributes in `GetExternalAttributes()` (p. 1892) to be consistent with the mode given.

If `IsMadeByUnix()` (p. 1894) is true then also stores mode in `GetExternalAttributes()`.

Note that the default constructor sets `GetSystemMadeBy()` (p. 1894) to `wxZIP_SYSTEM_MSDOS` by default. So to be able to store unix permissions when

creating zips, call `SetSystemMadeBy(wxZIP_SYSTEM_UNIX)`.

### **wxZipEntry::SetNotifier**

**void SetNotifier(wxZipNotifier& notifier)**

**void UnsetNotifier()**

Sets the *notifier* (p. 1896) for this entry. Whenever the *wxZipInputStream* (p. 1895) updates this entry, it will then invoke the associated notifier's *OnEntryUpdated* (p. 1897) method.

Setting a notifier is not usually necessary. It is used to handle certain cases when modifying an zip in a pipeline (i.e. between non-seekable streams).

### **See also**

*Archives on non-seekable streams* (p. 2228)  
*wxZipNotifier* (p. 1896)

### **wxZipEntry::Get/SetSystemMadeBy**

**int GetSystemMadeBy() const**

**void SetSystemMadeBy(int system)**

The originating file-system. The default constructor sets this to `wxZIP_SYSTEM_MSDOS`. Set it to `wxZIP_SYSTEM_UNIX` in order to be able to store unix permissions using *SetMode()* (p. 1893).

### **wxZipEntry::IsMadeByUnix**

**bool IsMadeByUnix() const**

Returns true if *GetSystemMadeBy()* (p. 1894) is a flavour of unix.

### **wxZipEntry::IsText/SetIsText**

**bool IsText() const**

**void SetIsText(bool isText = true)**

Indicates that this entry's data is text in an 8-bit encoding.

### **wxZipEntry::operator=**

**wxZipEntry& operator operator=(const wxZipEntry& entry)**

Assignment operator.

## wxZipInputStream

Input stream for reading zip files.

*GetNextEntry()* (p. 1896) returns an *wxZipEntry* (p. 1888) object containing the meta-data for the next entry in the zip (and gives away ownership). Reading from the *wxZipInputStream* then returns the entry's data. *Eof()* becomes true after an attempt has been made to read past the end of the entry's data. When there are no more entries, *GetNextEntry()* returns NULL and sets *Eof()*.

Note that in general zip entries are not seekable, and *wxZipInputStream::SeekI()* always returns *wxInvalidOffset*.

### Derived from

*wxArchiveInputStream* (p. 64)

### Include files

<wx/zipstrm.h>

```
Data structurestypedef wxZipEntry entry_type
```

### See also

*Archive formats such as zip* (p. 2223)

*wxZipEntry* (p. 1888)

*wxZipOutputStream* (p. 1897)

## wxZipInputStream::wxZipInputStream

```
wxZipInputStream(wxInputStream& stream, wxMBConv& conv = wxConvLocal)
```

```
wxZipInputStream(wxInputStream*stream, wxMBConv& conv = wxConvLocal)
```

Constructor. In a Unicode build the second parameter `conv` is used to translate the filename and comment fields into Unicode. It has no effect on the stream's data.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

```
wxZipInputStream(const wxString& archive, const wxString& file)
```

Compatibility constructor (requires `WXWIN_COMPATIBILITY_2_6`).

When this constructor is used, an emulation of seeking is switched on for compatibility with previous versions. Note however, that it is deprecated.

## wxZipInputStream::CloseEntry

```
bool CloseEntry()
```

Closes the current entry. On a non-seekable stream reads to the end of the current entry first.

### **wxZipInputStream::GetComment**

**wxString GetComment()**

Returns the zip comment.

This is stored at the end of the zip, therefore when reading a zip from a non-seekable stream, it returns the empty string until the end of the zip has been reached, i.e. when *GetNextEntry()* returns NULL.

### **wxZipInputStream::GetNextEntry**

**wxZipEntry\* GetNextEntry()**

Closes the current entry if one is open, then reads the meta-data for the next entry and returns it in a *wxZipEntry* (p. 1888) object, giving away ownership. The stream is then open and can be read.

### **wxZipInputStream::GetTotalEntries**

**int GetTotalEntries()**

For a zip on a seekable stream returns the total number of entries in the zip. For zips on non-seekable streams returns the number of entries returned so far by *GetNextEntry()* (p. 1896).

### **wxZipInputStream::OpenEntry**

**bool OpenEntry(wxZipEntry& entry)**

Closes the current entry if one is open, then opens the entry specified by the *entry* object. *entry* should be from the same zip file, and the zip should be on a seekable stream.

#### **See also**

*Looking up an archive entry by name* (p. 2225)

### **wxZipNotifier**

If you need to know when a *wxZipInputStream* (p. 1895) updates a *wxZipEntry* (p. 1888), you can create a notifier by deriving from this abstract base class, overriding *OnEntryUpdated()* (p. 1897). An instance of your notifier class can then be assigned to *wxZipEntry* objects, using *wxZipEntry::SetNotifier()* (p. 1894).

Setting a notifier is not usually necessary. It is used to handle certain cases when modifying a zip in a pipeline (i.e. between non-seekable streams). See '*Archives on*



*non-seekable streams* (p. 2228)'.

#### **Derived from**

No base class

#### **Include files**

<wx/zipstrm.h>

#### **See also**

*Archives on non-seekable streams* (p. 2228)

*wxZipEntry* (p. 1888)

*wxZipInputStream* (p. 1895)

*wxZipOutputStream* (p. 1897)

### **wxZipNotifier::OnEntryUpdated**

**void OnEntryUpdated(wxZipEntry& entry)**

Override this to receive notifications when an *wxZipEntry* (p. 1888) object changes.

## **wxZipOutputStream**

Output stream for writing zip files.

*PutNextEntry()* (p. 1899) is used to create a new entry in the output zip, then the entry's data is written to the *wxZipOutputStream*. Another call to *PutNextEntry()* closes the current entry and begins the next.

#### **Derived from**

*wxArchiveOutputStream* (p. 69)

#### **Include files**

<wx/zipstrm.h>

#### **See also**

*Archive formats such as zip* (p. 2223)

*wxZipEntry* (p. 1888)

*wxZipInputStream* (p. 1895)

### **wxZipOutputStream::wxZipOutputStream**

**wxZipOutputStream(wxOutputStream& stream, int level = -1, wxMBConv& conv =**

*wxConvLocal()*

**wxZipOutputStream(wxOutputStream\* stream, int level = -1, wxMBCConv& conv = wxConvLocal)**

Constructor. *level* is the compression level to use. It can be a value between 0 and 9 or -1 to use the default value which currently is equivalent to 6.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

In a Unicode build the third parameter *conv* is used to translate the filename and comment fields to an 8-bit encoding. It has no effect on the stream's data.

### **wxZipOutputStream::~wxZipOutputStream**

**~wxZipOutputStream()**

The destructor calls *Close()* (p. 1898) to finish writing the zip if it has not been called already.

### **wxZipOutputStream::Close**

**bool Close()**

Finishes writing the zip, returning true if successful. Called by the destructor if not called explicitly.

### **wxZipOutputStream::CloseEntry**

**bool CloseEntry()**

Close the current entry. It is called implicitly whenever another new entry is created with *CopyEntry()* (p. 1898) or *PutNextEntry()* (p. 1899), or when the zip is closed.

### **wxZipOutputStream::CopyArchiveMetaData**

**bool CopyArchiveMetaData(wxZipInputStream& inputStream)**

Transfers the zip comment from the *wxZipInputStream* (p. 1895) to this output stream.

### **wxZipOutputStream::CopyEntry**

**bool CopyEntry(wxZipEntry\* entry, wxZipInputStream& inputStream)**

Takes ownership of *entry* and uses it to create a new entry in the zip. *entry* is then opened in *inputStream* and its contents copied to this stream.

*CopyEntry()* is much more efficient than transferring the data using *Read()* and *Write()* since it will copy them without decompressing and recompressing them.

For zips on seekable streams, `entry` must be from the same zip file as `stream`. For non-seekable streams, `entry` must also be the last thing read from `inputStream`.

### **wxZipOutputStream::Get/SetLevel**

**int GetLevel() const**

**void SetLevel(int level)**

Set the compression level that will be used the next time an entry is created. It can be a value between 0 and 9 or -1 to use the default value which currently is equivalent to 6.

### **wxZipOutputStream::PutNextDirEntry**

**bool PutNextDirEntry(const wxString& name, const wxDateTime& dt = wxDateTime::Now())**

Create a new directory entry (see *wxArchiveEntry::IsDir()* (p. 64)) with the given name and timestamp.

*PutNextEntry()* (p. 1899) can also be used to create directory entries, by supplying a name with a trailing path separator.

### **wxZipOutputStream::PutNextEntry**

**bool PutNextEntry(wxZipEntry\* entry)**

Takes ownership of `entry` and uses it to create a new entry in the zip.

**bool PutNextEntry(const wxString& name, const wxDateTime& dt = wxDateTime::Now(), off\_t size = wxInvalidOffset)**

Create a new entry with the given name, timestamp and size.

### **wxZipOutputStream::SetComment**

**void SetComment(const wxString& comment)**

Sets a comment for the zip as a whole. It is written at the end of the zip.

## **wxZlibInputStream**

This filter stream decompresses a stream that is in zlib or gzip format. Note that reading the gzip format requires zlib version 1.2.1 or greater, (the builtin version does support gzip format).

The stream is not seekable, *Seek()* (p. 943) returns *wxInvalidOffset*. Also *GetSize()* (p. 1545) is not supported, it always returns 0.

**Derived from**

*wxFilterInputStream* (p. 646)

#### **Include files**

<wx/zstream.h>

#### **See also**

*wxInputStream* (p. 941), *wxZlibOutputStream* (p. 1901).

### **wxZlibInputStream::wxZlibInputStream**

**wxZlibInputStream**(wxInputStream& *stream*, int *flags* = wxZLIB\_AUTO)

**wxZlibInputStream**(wxInputStream\* *stream*, int *flags* = wxZLIB\_AUTO)

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

The *flags* wxZLIB\_ZLIB and wxZLIB\_GZIP specify whether the input data is in zlib or gzip format. If wxZLIB\_AUTO is used, then zlib will autodetect the stream type, this is the default.

If *flags* is wxZLIB\_NO\_HEADER, then the data is assumed to be a raw deflate stream without either zlib or gzip headers. This is a lower level mode, which is not usually used directly. It can be used to read a raw deflate stream embedded in a higher level protocol.

This version is not by default compatible with the output produced by the version of *wxZlibOutputStream* in wxWidgets 2.4.x. However, there is a compatibility mode, which is switched on by passing wxZLIB\_24COMPATIBLE for flags. Note that in when operating in compatibility mode error checking is very much reduced. The following symbols can be use for the flags:

```
// Flags
enum {
    #if WXWIN_COMPATIBILITY_2_4
        wxZLIB_24COMPATIBLE = 4, // read v2.4.x data without error
    #endif
    wxZLIB_NO_HEADER = 0,        // raw deflate stream, no header or
    checksum
    wxZLIB_ZLIB = 1,             // zlib header and checksum
    wxZLIB_GZIP = 2,             // gzip header and checksum, requires zlib
    1.2.1+
    wxZLIB_AUTO = 3              // autodetect header zlib or gzip
};
```

### **wxZlibInputStream::CanHandleGZip**

**static bool CanHandleGZip()**

Returns true if zlib library in use can handle gzip compressed data.

## wxZlibOutputStream

This stream compresses all data written to it. The compressed output can be in zlib or gzip format. Note that writing the gzip format requires zlib version 1.2.1 or greater (the builtin version does support gzip format).

The stream is not seekable, `SeekO()` (p. 1157) returns `wxInvalidOffset`.

### Derived from

`wxFilterOutputStream` (p. 647)

### Include files

`<wx/zstream.h>`

### See also

`wxOutputStream` (p. 1156), `wxZlibInputStream` (p. 1899)

## wxZlibOutputStream::wxZlibOutputStream

`wxZlibOutputStream(wxOutputStream& stream, int level = -1, int flags = wxZLIB_ZLIB)`

`wxZlibOutputStream(wxOutputStream* stream, int level = -1, int flags = wxZLIB_ZLIB)`

Creates a new write-only compressed stream. *level* means level of compression. It is number between 0 and 9 (including these values) where 0 means no compression and 9 best but slowest compression. -1 is default value (currently equivalent to 6).

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

The *flags* `wxZLIB_ZLIB` and `wxZLIB_GZIP` specify whether the output data will be in zlib or gzip format. `wxZLIB_ZLIB` is the default.

If *flags* is `wxZLIB_NO_HEADER`, then a raw deflate stream is output without either zlib or gzip headers. This is a lower level mode, which is not usually used directly. It can be used to embed a raw deflate stream in a higher level protocol.

The following symbols can be use for the compression level and flags:

```
// Compression level
enum {
    wxZ_DEFAULT_COMPRESSION = -1,
    wxZ_NO_COMPRESSION = 0,
    wxZ_BEST_SPEED = 1,
    wxZ_BEST_COMPRESSION = 9
};
```

```
// Flags
enum {
    wxZLIB_NO_HEADER = 0, // raw deflate stream, no header or checksum
    wxZLIB_ZLIB = 1,      // zlib header and checksum
    wxZLIB_GZIP = 2       // gzip header and checksum, requires zlib
1.2.1+
};
```

### **wxZlibOutputStream::CanHandleGZip**

#### **static bool CanHandleGZip()**

Returns true if zlib library in use can handle gzip compressed data.

# Functions

The functions and macros defined in wxWidgets are described here: you can either look up a function using the alphabetical listing of them or find it in the corresponding topic.

## Alphabetical functions and macros list

*CLASSINFO* (p. 1964)  
*copystring* (p. 1929)  
*DECLARE\_ABSTRACT\_CLASS* (p. 1965)  
*DECLARE\_APP* (p. 1965)  
*DECLARE\_CLASS* (p. 1965)  
*DECLARE\_DYNAMIC\_CLASS* (p. 1966)  
*IMPLEMENT\_ABSTRACT\_CLASS2* (p. 1966)  
*IMPLEMENT\_ABSTRACT\_CLASS* (p. 1966)  
*IMPLEMENT\_APP* (p. 1967)  
*IMPLEMENT\_CLASS2* (p. 1967)  
*IMPLEMENT\_CLASS* (p. 1967)  
*IMPLEMENT\_DYNAMIC\_CLASS2* (p. 1968)  
*IMPLEMENT\_DYNAMIC\_CLASS* (p. 1967)  
*wxAboutBox* (p. 1934)  
*wxASSERT* (p. 1980)  
*wxASSERT\_MIN\_BITSIZE* (p. 1980)  
*wxASSERT\_MSG* (p. 1980)  
*wxBeginBusyCursor* (p. 1934)  
*wxBell* (p. 1935)  
*wxBITMAP* (p. 1944)  
*wxCHANGE\_UMASK* (p. 1922)  
*wxCHECK* (p. 1982)  
*wxCHECK2\_MSG* (p. 1982)  
*wxCHECK2* (p. 1982)  
*wxCHECK\_GCC\_VERSION* (p. 1908)  
*wxCHECK\_MSG* (p. 1982)  
*wxCHECK\_RET* (p. 1982)  
*wxCHECK\_VERSION* (p. 1909)  
*wxCHECK\_VERSION\_FULL* (p. 1909)  
*wxCHECK\_W32API\_VERSION* (p. 1909)  
*wxClientDisplayRect* (p. 1945)  
*wxClipboardOpen* (p. 1949)  
*wxCloseClipboard* (p. 1949)  
*wxColourDisplay* (p. 1945)  
*wxCOMPILE\_TIME\_ASSERT* (p. 1981)  
*wxCOMPILE\_TIME\_ASSERT2* (p. 1981)  
*wxCONCAT* (p. 1951)  
*wxConcatFiles* (p. 1922)  
*wxConstCast* (p. 1968)

*wxCopyFile* (p. 1923)  
*wxCreateDynamicObject* (p. 1968)  
*wxCreateFileTipProvider* (p. 1935)  
*wxCRT\_SECT\_DECLARE* (p. 1917)  
*wxCRT\_SECT\_DECLARE\_MEMBER* (p. 1917)  
*wxCRT\_SECT\_LOCKER* (p. 1918)  
*wxCRTITICAL\_SECTION* (p. 1918)  
    *wxDDECleanUp* (p. 1954)  
    *wxDDEInitialize* (p. 1954)  
*wxDROP\_ICON* (p. 1945)  
*wxDebugMsg* (p. 1971)  
*WXDEBUG\_NEW* (p. 1968)  
*wxDirExists* (p. 1922)  
*wxDirSelector* (p. 1935)  
*wxDisplayDepth* (p. 1945)  
*wxDisplaySize* (p. 1945)  
*wxDisplaySizeMM* (p. 1945)  
*wxDos2UnixFilename* (p. 1920)  
*wxDynamicCastThis* (p. 1969)  
*wxDynamicCast* (p. 1969)  
*wxDYNLIB\_FUNCTION* (p. 1951)  
*wxEmptyClipboard* (p. 1949)  
*wxEnableTopLevelWindows* (p. 1954)  
*wxEndBusyCursor* (p. 1937)  
*wxENTER\_CRIT\_SECT* (p. 1918)  
*wxEntry* (p. 1909)  
    *wxEntryStart* (p. 1910)  
    *wxEntryCleanup* (p. 1910)  
*wxEnumClipboardFormats* (p. 1949)  
*wxError* (p. 1971)  
*wxExecute* (p. 1913)  
*wxExit* (p. 1915)  
*wxEXPLICIT* (p. 1952)  
*wxFAIL\_MSG* (p. 1981)  
*wxFAIL* (p. 1981)  
*wxFatalError* (p. 1972)  
*wxFileExists* (p. 1920)  
*wxFileModificationTime* (p. 1920)  
*wxFileNameFromPath* (p. 1920)  
*wxFileSelector* (p. 1936)  
*wxFindFirstFile* (p. 1920)  
*wxFindMenuItemId* (p. 1955)  
*wxFindNextFile* (p. 1921)  
*wxFindWindowAtPointer* (p. 1956)  
*wxFindWindowAtPoint* (p. 1955)  
*wxFindWindowByLabel* (p. 1955)  
*wxFindWindowByName* (p. 1955)  
*wxFinite* (p. 1944)  
*wxGenericAboutBox* (p. 1937)  
*wxGetActiveWindow* (p. 1956)  
*wxGetApp* (p. 1911)



*wxBatteryState* (p. 1956)  
*wxClipboardData* (p. 1950)  
*wxClipboardFormatName* (p. 1950)  
*wxColourFromUser* (p. 1937)  
*wxCwd* (p. 1923)  
*wxDiskSpace* (p. 1921)  
*wxDisplayName* (p. 1956)  
*wxDisplaySize* (p. 1945)  
*wxDisplaySizeMM* (p. 1945)  
*wxElapsedTime* (p. 1977)  
*wxEmailAddress* (p. 1926)  
*wxEnv* (p. 1983)  
*wxFileKind* (p. 1921)  
*wxFontFromUser* (p. 1938)  
*wxFreeMemory* (p. 1926)  
*wxFullHostName* (p. 1926)  
*wxHomeDir* (p. 1926)  
*wxHostName* (p. 1927)  
*wxKeyState* (p. 1952)  
*wxLocalTimeMillis* (p. 1978)  
*wxLocalTime* (p. 1977)  
*wxMousePosition* (p. 1957)  
*wxMouseState* (p. 1957)  
*wxMultipleChoices* (p. 1938)  
*wxMultipleChoice* (p. 1940)  
*wxNumberFromUser* (p. 1939)  
*wxOSDirectory* (p. 1922)  
*wxOsDescription* (p. 1927)  
*wxOsVersion* (p. 1927)  
*wxPasswordFromUser* (p. 1939)  
*wxPowerType* (p. 1956)  
*wxPrinterCommand* (p. 1947)  
*wxPrinterFile* (p. 1947)  
*wxPrinterMode* (p. 1947)  
*wxPrinterOptions* (p. 1947)  
*wxPrinterOrientation* (p. 1947)  
*wxPrinterPreviewCommand* (p. 1947)  
*wxPrinterScaling* (p. 1948)  
*wxPrinterTranslation* (p. 1948)  
*wxProcessId* (p. 1916)  
*wxResource* (p. 1958)  
*wxSingleChoiceData* (p. 1942)  
*wxSingleChoiceIndex* (p. 1941)  
*wxSingleChoice* (p. 1940)  
*wxTempFileName* (p. 1923)  
*wxTextFromUser* (p. 1940)  
*wxTopLevelParent* (p. 1959)  
*wxTranslation* (p. 1930)  
*wxUTCTime* (p. 1978)  
*wxUserHome* (p. 1928)  
*wxUserId* (p. 1928)

*wxGetUserName* (p. 1929)  
*wxGetWorkingDirectory* (p. 1923)  
*wxGetenv* (p. 1983)  
*wxHandleFatalExceptions* (p. 1911)  
*wxICON* (p. 1946)  
*wxINTXX\_SWAP\_ALWAYS* (p. 1963)  
*wxINTXX\_SWAP\_ON\_BE* (p. 1963)  
*wxINTXX\_SWAP\_ON\_LE* (p. 1964)  
*wxInitAllImageHandlers* (p. 1911)  
*wxInitialize* (p. 1911)  
*wxIsAbsolutePath* (p. 1922)  
*wxIsBusy* (p. 1942)  
*wxIsClipboardFormatAvailable* (p. 1950)  
*wxIsDebuggerRunning* (p. 1983)  
*wxIsEmpty* (p. 1930)  
*wxIsMainThread* (p. 1918)  
*wxIsNaN* (p. 1944)  
*wxIsPlatformLittleEndian* (p. 1928)  
*wxIsPlatform64Bit* (p. 1928)  
*wxIsWild* (p. 1923)  
*wxKill* (p. 1915)  
*wxLaunchDefaultBrowser* (p. 1959)  
*wxLEAVE\_CRIT\_SECT* (p. 1918)  
*wxLoadUserResource* (p. 1959)  
*wxLogDebug* (p. 1973)  
*wxLogError* (p. 1972)  
*wxLogFatalError* (p. 1972)  
*wxLogMessage* (p. 1972)  
*wxLogStatus* (p. 1973)  
*wxLogSysError* (p. 1973)  
*wxLogTrace* (p. 1974)  
*wxLogVerbose* (p. 1973)  
*wxLogWarning* (p. 1972)  
*wxLL* (p. 1952)  
*wxLongLongFmtSpec* (p. 1952)  
*wxMakeMetafilePlaceable* (p. 1946)  
*wxMatchWild* (p. 1923)  
*wxMessageBox* (p. 1942)  
*wxMilliSleep* (p. 1978)  
*wxMicroSleep* (p. 1978)  
*wxMkdir* (p. 1924)  
*wxMutexGuiEnter* (p. 1919)  
*wxMutexGuiLeave* (p. 1919)  
*wxNewId* (p. 1953)  
*wxNow* (p. 1979)  
*wxOnAssert* (p. 1980)  
*wxON\_BLOCK\_EXIT* (p. 1953)  
*wxON\_BLOCK\_EXIT\_OBJ* (p. 1953)  
*wxOpenClipboard* (p. 1950)  
*wxParseCommonDialogsFilter* (p. 1924)  
*wxPathOnly* (p. 1922)

*wxPLURAL* (p. 1933)  
*wxPostDelete* (p. 1960)  
*wxPostEvent* (p. 1960)  
*wxRegisterClipboardFormat* (p. 1950)  
*wxRegisterId* (p. 1954)  
*wxRemoveFile* (p. 1924)  
*wxRenameFile* (p. 1924)  
*wxRmdir* (p. 1924)  
*wxSafeShowMessage* (p. 1975)  
*wxSafeYield* (p. 1912)  
*wxSetClipboardData* (p. 1951)  
*wxSetCursor* (p. 1946)  
*wxSetDisplayName* (p. 1960)  
*wxSetEnv* (p. 1984)  
*wxSetPrinterCommand* (p. 1948)  
*wxSetPrinterFile* (p. 1948)  
*wxSetPrinterMode* (p. 1948)  
*wxSetPrinterOptions* (p. 1948)  
*wxSetPrinterOrientation* (p. 1948)  
*wxSetPrinterPreviewCommand* (p. 1948)  
*wxSetPrinterScaling* (p. 1949)  
*wxSetPrinterTranslation* (p. 1949)  
*wxSetWorkingDirectory* (p. 1924)  
*wxShell* (p. 1916)  
*wxShowTip* (p. 1943)  
*wxShutdown* (p. 1916)  
*wxSleep* (p. 1979)  
*wxSnprintf* (p. 1932)  
*wxSplitPath* (p. 1925)  
*wxStartTimer* (p. 1979)  
*wxStaticCast* (p. 1970)  
*wxStrcmp* (p. 1930)  
*wxStricmp* (p. 1931)  
*wxStringEq* (p. 1931)  
*wxStringMatch* (p. 1931)  
*wxStringTokenize* (p. 1931)  
*wxStripMenuCodes* (p. 1960)  
*wxStrlen* (p. 1931)  
*wxSTRINGIZE* (p. 1961)  
*wxSTRINGIZE\_T* (p. 1961)  
*wxSUPPRESS\_GCC\_PRIVATE\_DTOR\_WARNING* (p. 1961)  
*wxSysErrorCode* (p. 1975)  
*wxSysErrorMsg* (p. 1976)  
*wxT* (p. 1932)  
*wxTrace* (p. 1976)  
*WXTRACE* (p. 1976)  
*wxTraceLevel* (p. 1977)  
*WXTRACELEVEL* (p. 1976)  
*wxTransferFileToStream* (p. 1925)  
*wxTransferStreamToFile* (p. 1925)  
*wxTrap* (p. 1983)

`wxULL` (p. 1962)  
`wxUninitialize` (p. 1912)  
`wxUnix2DosFilename` (p. 1922)  
`wxUnsetEnv` (p. 1984)  
`wxUsleep` (p. 1979)  
`wxVaCopy` (p. 1962)  
`wxVsnprintf` (p. 1933)  
`wxWakeUpIdle` (p. 1912)  
`wxWriteResource` (p. 1962)  
`wxYield` (p. 1912)  
`wx_const_cast` (p. 1970)  
`wx_reinterpret_cast` (p. 1970)  
`wx_static_cast` (p. 1970)  
`wx_truncate_cast` (p. 1971)  
`_` (p. 1933)  
`_T` (p. 1934) `__WXFUNCTION__` (p. 1963)

## Version macros

The following constants are defined in `wxWidgets`:

- `wxMAJOR_VERSION` is the major version of `wxWidgets`
- `wxMINOR_VERSION` is the minor version of `wxWidgets`
- `wxRELEASE_NUMBER` is the release number
- `wxSUBRELEASE_NUMBER` is the subrelease number which is 0 for all official releases

For example, the values of these constants for `wxWidgets 2.1.15` are 2, 1 and 15.

Additionally, `wxVERSION_STRING` is a user-readable string containing the full `wxWidgets` version and `wxVERSION_NUMBER` is a combination of the three version numbers above: for 2.1.15, it is 2115 and it is 2200 for `wxWidgets 2.2`.

The subrelease number is only used for the sources in between official releases and so normally is not useful.

### Include files

`<wx/version.h>` or `<wx/defs.h>`

### `wxCHECK_GCC_VERSION`

**`bool wxCHECK_GCC_VERSION(major, minor)`**

Returns 1 if the compiler being used to compile the code is GNU C++ compiler (g++) version `major.minor` or greater. Otherwise, and also if the compiler is not GNU C++ at all, returns 0.

**wxCHECK\_VERSION****bool wxCHECK\_VERSION**(*major, minor, release*)

This is a macro which evaluates to true if the current wxWidgets version is at least *major.minor.release*.

For example, to test if the program is compiled with wxWidgets 2.2 or higher, the following can be done:

```
    wxString s;  
    #if wxCHECK_VERSION(2, 2, 0)  
        if ( s.StartsWith("foo") )  
    #else // replacement code for old version  
        if ( strcmp(s, "foo", 3) == 0 )  
    #endif  
    {  
        ...  
    }
```

**wxCHECK\_VERSION\_FULL****bool wxCHECK\_VERSION\_FULL**(*major, minor, release, subrel*)

Same as *wxCHECK\_VERSION* (p. 1909) but also checks that *wxSUBRELEASE\_NUMBER* is at least *subrel*.

**wxCHECK\_W32API\_VERSION****bool wxCHECK\_W32API\_VERSION**(*major, minor, release*)

Returns 1 if the version of w32api headers used is *major.minor.release* or greater. Otherwise, and also if we are not compiling with mingw32/cygwin under Win32 at all, returns 0.

## Application initialization and termination

The functions in this section are used on application startup/shutdown and also to control the behaviour of the main event loop of the GUI programs.

**::wxEntry**

This initializes wxWidgets in a platform-dependent way. Use this if you are not using the default wxWidgets entry code (e.g. *main* or *WinMain*). For example, you can initialize wxWidgets from an Microsoft Foundation Classes application using this function.

The following overload of *wxEntry* is available under all platforms:

**int wxEntry**(*int& argc, wxChar \*\*argv*)

Under MS Windows, an additional overload suitable for calling from *WinMain* is available:

```
int wxEntry(HINSTANCE hInstance, HINSTANCE hPrevInstance = NULL, char
*pCmdLine = NULL, int nCmdShow = SW_SHOWNORMAL)
```

(notice that under Windows CE platform, and only there, the type of *pCmdLine* is `wchar_t *`, otherwise it is `char *`, even in Unicode build).

### See also

*wxEntryStart* (p. 1910)

### Remarks

To clean up `wxWidgets`, call `wxApp::OnExit` followed by the static function `wxApp::CleanUp`. For example, if exiting from an MFC application that also uses `wxWidgets`:

```
int CTheApp::ExitInstance()
{
    // OnExit isn't called by CleanUp so must be called explicitly.
    wxTheApp->OnExit();
    wxApp::CleanUp();

    return CWinApp::ExitInstance();
}
```

### Include files

<wx/app.h>

### ::wxEntryCleanup

```
void wxEntryCleanup()
```

Free resources allocated by a successful call to *wxEntryStart* (p. 1910).

### Include files

<wx/init.h>

### ::wxEntryStart

```
bool wxEntryStart(int& argc, wxChar **argv)
```

This function can be used to perform the initialization of `wxWidgets` if you can't use the default initialization code for any reason.

If the function returns `true`, the initialization was successful and the global *wxApp* (p. 45) object `wxTheApp` has been created. Moreover, *wxEntryCleanup* (p. 1910) must be called afterwards. If the function returns `false`, a catastrophic initialization error occurred and (at least the GUI part of) the library can't be used at all.

Notice that parameters *argc* and *argv* may be modified by this function.

**Include files**

<wx/init.h>

**::wxGetApp****wxAppDerivedClass& wxGetApp()**

This function doesn't exist in `wxWidgets` but it is created by using the `IMPLEMENT_APP` (p. 1967) macro. Thus, before using it anywhere but in the same module where this macro is used, you must make it available using `DECLARE_APP` (p. 1965).

The advantage of using this function compared to directly using the global `wxTheApp` pointer is that the latter is of type `wxApp *` and so wouldn't allow you to access the functions specific to your application class but not present in `wxApp` while `wxGetApp()` returns the object of the right type.

**::wxHandleFatalExceptions****bool wxHandleFatalExceptions(bool *dolt* = true)**

If *dolt* is true, the fatal exceptions (also known as general protection faults under Windows or segmentation violations in the Unix world) will be caught and passed to `wxApp::OnFatalException` (p. 51). By default, i.e. before this function is called, they will be handled in the normal way which usually just means that the application will be terminated. Calling `wxHandleFatalExceptions()` with *dolt* equal to false will restore this default behaviour.

**::wxInitAllImageHandlers****void wxInitAllImageHandlers()**

Initializes all available image handlers. For a list of available handlers, see `wxImage` (p. 906).

**See also**

`wxImage` (p. 906), `wxImageHandler` (p. 930)

**Include files**

<wx/image.h>

**::wxInitialize****bool wxInitialize()**

This function is used in `wxBase` only and only if you don't create `wxApp` (p. 45) object at all. In this case you must call it from your `main()` function before calling any other `wxWidgets` functions.

If the function returns `false` the initialization could not be performed, in this case the

library cannot be used and *wxUninitialize* (p. 1912) shouldn't be called neither.

This function may be called several times but *wxUninitialize* (p. 1912) must be called for each successful call to this function.

**Include files**

<wx/app.h>

**::wxSafeYield**

**bool wxSafeYield(wxWindow\* win = NULL, bool onlyIfNeeded = false)**

This function is similar to *wxYield*, except that it disables the user input to all program windows before calling *wxYield* and re-enables it again afterwards. If *win* is not NULL, this window will remain enabled, allowing the implementation of some limited user interaction.

Returns the result of the call to *::wxYield* (p. 1912).

**Include files**

<wx/utils.h>

**::wxUninitialize**

**void wxUninitialize()**

This function is for use in console (*wxBase*) programs only. It must be called once for each previous successful call to *wxInitialize* (p. 1911).

**Include files**

<wx/app.h>

**::wxYield**

**bool wxYield()**

Calls *wxApp::Yield* (p. 56).

This function is kept only for backwards compatibility. Please use the *wxApp::Yield* (p. 56) method instead in any new code.

**Include files**

<wx/app.h> or <wx/utils.h>

**::wxWakeUpIdle**

**void wxWakeUpIdle()**

This functions wakes up the (internal and platform dependent) idle system, i.e. it will force the system to send an idle event even if the system currently *is* idle and thus would not



send any idle event until after some other event would get sent. This is also useful for sending events between two threads and is used by the corresponding functions `::wxPostEvent` (p. 1960) and `wxEvtHandler::AddPendingEvent` (p. 576).

#### Include files

<wx/app.h>

## Process control functions

The functions in this section are used to launch or terminate the other processes.

### **::wxExecute**

**long wxExecute(const wxString& command, int sync = wxEXEC\_ASYNC, wxProcess \*callback = NULL)**

**wxPerl note:** In wxPerl this function is called `Wx::ExecuteCommand`

**long wxExecute(char \*\*argv, int flags = wxEXEC\_ASYNC, wxProcess \*callback = NULL)**

**wxPerl note:** In wxPerl this function is called `Wx::ExecuteArgs`

**long wxExecute(const wxString& command, wxArrayString& output, int flags = 0)**

**wxPerl note:** In wxPerl this function is called `Wx::ExecuteStdout` and it only takes the `command` argument, and returns a 2-element list ( `status`, `output` ), where `output` is an array reference.

**long wxExecute(const wxString& command, wxArrayString& output, wxArrayString& errors, int flags = 0)**

**wxPerl note:** In wxPerl this function is called `Wx::ExecuteStdoutStderr` and it only takes the `command` argument, and returns a 3-element list ( `status`, `output`, `errors` ), where `output` and `errors` are array references.

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.

The semantics of the third and fourth versions is different from the first two and is described in more details below.

If `flags` parameter contains `wxEXEC_ASYNC` flag (the default), flow of control immediately returns. If it contains `wxEXEC_SYNC`, the current application waits until the other program has terminated.

In the case of synchronous execution, the return value is the exit code of the process

(which terminates by the moment the function returns) and will be -1 if the process couldn't be started and typically 0 if the process terminated successfully. Also, while waiting for the process to terminate, `wxExecute` will call `wxYield` (p. 1912). Because of this, by default this function disables all application windows to avoid unexpected reentrancies which could result from the users interaction with the program while the child process is running. If you are sure that it is safe to not disable the program windows, you may pass `wxEXEC_NODISABLE` flag to prevent this automatic disabling from happening.

For asynchronous execution, however, the return value is the process id and zero value indicates that the command could not be executed. As an added complication, the return value of -1 in this case indicates that we didn't launch a new process, but connected to the running one (this can only happen in case of using DDE under Windows for command execution). In particular, in this, and only this, case the calling code will not get the notification about process termination.

If callback isn't NULL and if execution is asynchronous, `wxProcess::OnTerminate` (p. 1228) will be called when the process finishes. Specifying this parameter also allows you to redirect the standard input and/or output of the process being launched by calling `Redirect` (p. 1229). If the child process IO is redirected, under Windows the process window is not shown by default (this avoids having to flush an unnecessary console for the processes which don't create any windows anyhow) but a `wxEXEC_NOHIDE` flag can be used to prevent this from happening, i.e. with this flag the child process window will be shown normally.

Under Unix the flag `wxEXEC_MAKE_GROUP_LEADER` may be used to ensure that the new process is a group leader (this will create a new session if needed). Calling `wxKill` (p. 1915) passing `wxKILL_CHILDREN` will kill this process as well as all of its children (except those which have started their own session).

Finally, you may use the third overloaded version of this function to execute a process (always synchronously, the contents of *flags* is or'd with `wxEXEC_SYNC`) and capture its output in the array *output*. The fourth version adds the possibility to additionally capture the messages from standard error output in the *errors* array.

**NB:** Currently `wxExecute()` can only be used from the main thread, calling this function from another thread will result in an assert failure in debug build and won't work.

### See also

`wxShell` (p. 1916), `wxProcess` (p. 1224), *Exec sample* (p. 2034).

### Parameters

*command*

The command to execute and any parameters to pass to it as a single string.

*argv*

The command to execute should be the first element of this array, any additional ones are the command parameters and the array must be terminated with a NULL pointer.

*flags*

Combination of bit masks `wxEXEC_ASYNC`, `wxEXEC_SYNC` and `wxEXEC_NOHIDE`

*callback*

An optional pointer to `wxProcess` (p. 1224)

### **Include files**

`<wx/utils.h>`

### **::wxExit**

#### **void wxExit()**

Exits application after calling `wxApp::OnExit` (p. 51). 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 `wxCloseEvent` (p. 200) and `wxApp` (p. 45).

### **Include files**

`<wx/app.h>`

### **::wxKill**

**int wxKill(long pid, int sig = `wxSIGTERM`, wxKillError \*rc = NULL, int flags = 0)**

Equivalent to the Unix kill function: send the given signal `sig` to the process with PID `pid`. The valid signal values are

```
enum wxSignal
{
    wxSIGNONE = 0, // verify if the process exists under Unix
    wxSIGHUP,
    wxSIGINT,
    wxSIGQUIT,
    wxSIGILL,
    wxSIGTRAP,
    wxSIGABRT,
    wxSIGEMT,
    wxSIGFPE,
    wxSIGKILL, // forcefully kill, dangerous!
    wxSIGBUS,
    wxSIGSEGV,
    wxSIGSYS,
    wxSIGPIPE,
    wxSIGALRM,
    wxSIGTERM // terminate the process gently
};
```

`wxSIGNONE`, `wxSIGKILL` and `wxSIGTERM` have the same meaning under both Unix and Windows but all the other signals are equivalent to `wxSIGTERM` under Windows.

Returns 0 on success, -1 on failure. If *rc* parameter is not NULL, it will be filled with an element of `wxKillError` enum:

```
enum wxKillError
{
    wxKILL_OK,                // no error
    wxKILL_BAD_SIGNAL,        // no such signal
    wxKILL_ACCESS_DENIED,     // permission denied
    wxKILL_NO_PROCESS,        // no such process
    wxKILL_ERROR              // another, unspecified error
};
```

The *flags* parameter can be `wxKILL_NOCHILDREN` (the default), or `wxKILL_CHILDREN`, in which case the child processes of this process will be killed too. Note that under Unix, for `wxKILL_CHILDREN` to work you should have created the process by passing `wxEXEC_MAKE_GROUP_LEADER` to `wxExecute`.

### See also

`wxProcess::Kill` (p. 1227), `wxProcess::Exists` (p. 1228), *Exec sample* (p. 2034)

### Include files

<wx/utils.h>

### ::wxGetProcessId

**unsigned long wxGetProcessId()**

Returns the number uniquely identifying the current process in the system.

If an error occurs, 0 is returned.

### Include files

<wx/utils.h>

### ::wxShell

**bool wxShell(const wxString& command = NULL)**

Executes a command in an interactive shell window. If no command is specified, then just the shell is spawned.

See also `wxExecute` (p. 1913), *Exec sample* (p. 2034).

### Include files

<wx/utils.h>

### ::wxShutdown

**bool wxShutdown(wxShutdownFlags flags)**

This function shuts down or reboots the computer depending on the value of the *flags*. Please notice that doing this requires the corresponding access rights (superuser under Unix, `SE_SHUTDOWN` privilege under Windows NT) and that this function is only implemented under Unix and Win32.

**Parameters**

*flags*

Either `wxSHUTDOWN_POWEROFF` or `wxSHUTDOWN_REBOOT`

**Returns**

`true` on success, `false` if an error occurred.

**Include files**

`<wx/utils.h>`

## Thread functions

The functions and macros here mainly exist to make it writing the code which may be compiled in multi thread build (`wxUSE_THREADS = 1`) as well as in single thread configuration (`wxUSE_THREADS = 0`).

For example, a static variable must be protected against simultaneous access by multiple threads in the former configuration but in the latter the extra overhead of using the critical section is not needed. To solve this problem, the `wxCriticalSection` (p. 1918) macro may be used to create and use the critical section only when needed.

**Include files**

`<wx/thread.h>`

**See also**

*wxThread* (p. 1670), *wxMutex* (p. 1131), *Multithreading overview* (p. 2149)

**wxCriticalSection\_DECLARE**

`wxCriticalSection_DECLARE(cs)`

This macro declares a (static) critical section object named `cs` if `wxUSE_THREADS` is 1 and does nothing if it is 0.

**wxCriticalSection\_DECLARE\_MEMBER**

`wxCriticalSection_DECLARE(cs)`

This macro declares a critical section object named `cs` if `wxUSE_THREADS` is 1 and does nothing if it is 0. As it doesn't include the `static` keyword

(unlike `wxCRT_SECT_DECLARE` (p. 1917)), it can be used to declare a class or struct member which explains its name.

### **wxCRT\_SECT\_LOCKER**

**wxCRT\_SECT\_LOCKER**(*name*, *cs*)

This macro creates a *critical section lock* (p. 295) object named *name* and associated with the critical section *cs* if `wxUSE_THREADS` is 1 and does nothing if it is 0.

### **wxCRTITICAL\_SECTION**

**wxCRTITICAL\_SECTION**(*name*)

This macro combines `wxCRT_SECT_DECLARE` (p. 1917) and `wxCRT_SECT_LOCKER` (p. 1918): it creates a static critical section object and also the lock object associated with it. Because of this, it can be only used inside a function, not at global scope. For example:

```
int IncCount()
{
    static int s_counter = 0;

    wxCRTITICAL_SECTION(counter);

    return ++s_counter;
}
```

(note that we suppose that the function is called the first time from the main thread so that the critical section object is initialized correctly by the time other threads start calling it, if this is not the case this approach **cannot** be used and the critical section must be made a global instead).

### **wxENTER\_CRIT\_SECT**

**wxENTER\_CRIT\_SECT**(`wxCriticalSection&` *cs*)

This macro is equivalent to *cs.Enter()* (p. 294) if `wxUSE_THREADS` is 1 and does nothing if it is 0.

### **::wxIsMainThread**

**bool wxIsMainThread()**

Returns `true` if this thread is the main one. Always returns `true` if `wxUSE_THREADS` is 0.

### **wxLEAVE\_CRIT\_SECT**

**wxLEAVE\_CRIT\_SECT**(`wxCriticalSection&` *cs*)

This macro is equivalent to *cs.Leave()* (p. 295) if `wxUSE_THREADS` is 1 and does nothing if it is 0.

**::wxMutexGuiEnter****void wxMutexGuiEnter()**

This function must be called when any thread other than the main GUI thread wants to get access to the GUI library. This function will block the execution of the calling thread until the main thread (or any other thread holding the main GUI lock) leaves the GUI library and no other thread will enter the GUI library until the calling thread calls `::wxMutexGuiLeave()` (p. 1919).

Typically, these functions are used like this:

```
void MyThread::Foo(void)
{
    // before doing any GUI calls we must ensure that this thread is
    the only
    // one doing it!

    wxMutexGuiEnter();

    // Call GUI here:
    my_window->DrawSomething();

    wxMutexGuiLeave();
}
```

Note that under GTK, no creation of top-level windows is allowed in any thread but the main one.

This function is only defined on platforms which support preemptive threads.

**::wxMutexGuiLeave****void wxMutexGuiLeave()**

See `::wxMutexGuiEnter()` (p. 1919).

This function is only defined on platforms which support preemptive threads.

## File functions

**Include files**

<wx/filefn.h>

**See also**

`wxPathList` (p. 1174)

`wxDir` (p. 509)

`wxFile` (p. 591)

`wxFileName` (p. 609)

**::wxDos2UnixFilename****void wxDos2UnixFilename(wxChar \*s)**

Converts a DOS to a Unix filename by replacing backslashes with forward slashes.

**::wxFileExists****bool wxFileExists(const wxString& filename)**

Returns true if the file exists and is a plain file.

**::wxFileModificationTime****time\_t wxFileModificationTime(const wxString& filename)**

Returns time of last modification of given file.

The function returns (time\_t)-1 if an error occurred (e.g. file not found).

**::wxFileNameFromPath****wxString wxFileNameFromPath(const wxString& path)****char \* wxFileNameFromPath(char \*path)**

**NB:** This function is obsolete, please use *wxFileName::SplitPath* (p. 627) instead.

Returns the filename for a full path. The second form returns a pointer to temporary storage that should not be deallocated.

**::wxFindFirstFile****wxString wxFindFirstFile(const char \*spec, int flags = 0)**

This function does directory searching; returns the first file that matches the path *spec*, or the empty string. Use *wxFindNextFile* (p. 1921) to get the next matching file. Neither will report the current directory "." or the parent directory "..".

**Warning**

As of wx 2.5.2, these functions are not thread-safe! (they use static variables). You probably want to use *wxDir::GetFirst* (p. 511) or *wxDirTraverser* (p. 518) instead.

*spec* may contain wildcards.

*flags* may be *wxDIR* for restricting the query to directories, *wxFILE* for files or zero for either.

For example:

```
wxString f = wxFindFirstFile("/home/project/*.*");
while ( !f.empty() )
```



```
{
    ...
    f = wxFindNextFile();
}
```

### **::wxFindNextFile**

#### **wxString wxFindNextFile()**

Returns the next file that matches the path passed to *wxFindFirstFile* (p. 1920).

See *wxFindFirstFile* (p. 1920) for an example.

### **::wxGetDiskSpace**

#### **bool wxGetDiskSpace(const wxString& path, wxLongLong \*total = NULL, wxLongLong \*free = NULL)**

This function returns the total number of bytes and number of free bytes on the disk containing the directory *path* (it should exist). Both *total* and *free* parameters may be `NULL` if the corresponding information is not needed.

#### **Returns**

`true` on success, `false` if an error occurred (for example, the directory doesn't exist).

#### **Portability**

This function is implemented for Win32, Mac OS and generic Unix provided the system has `statfs()` function.

This function first appeared in wxWidgets 2.3.2.

### **::wxGetFileKind**

#### **wxFileKind wxGetFileKind(int fd)**

#### **wxFileKind wxGetFileKind(FILE \*fp)**

Returns the type of an open file. Possible return values are:

```
enum wxFileKind
{
    wxFILE_KIND_UNKNOWN,
    wxFILE_KIND_DISK,      // a file supporting seeking to arbitrary
offsets
    wxFILE_KIND_TERMINAL,  // a tty
    wxFILE_KIND_PIPE       // a pipe
};
```

#### **Include files**

<wx/filefn.h>

### **::wxGetOSDirectory**

**wxString wxGetOSDirectory()**

Returns the Windows directory under Windows; on other platforms returns the empty string.

### **::wxIsAbsolutePath**

**bool wxIsAbsolutePath(const wxString& filename)**

Returns true if the argument is an absolute filename, i.e. with a slash or drive name at the beginning.

### **::wxDirExists**

**bool wxDirExists(const wxChar \*dirname)**

Returns true if *dirname* exists and is a directory.

### **::wxPathOnly**

**wxString wxPathOnly(const wxString& path)**

Returns the directory part of the filename.

### **::wxUnix2DosFilename**

**void wxUnix2DosFilename(wxChar \*s)**

This function is deprecated, use *wxFileName* (p. 609) instead.

Converts a Unix to a DOS filename by replacing forward slashes with backslashes.

### **wxCHANGE\_UMASK**

**wxCHANGE\_UMASK(int mask)**

Under Unix this macro changes the current process umask to the given value, unless it is equal to -1 in which case nothing is done, and restores it to the original value on scope exit. It works by declaring a variable which sets umask to *mask* in its constructor and restores it in its destructor.

Under other platforms this macro expands to nothing.

### **::wxConcatFiles**

**bool wxConcatFiles(const wxString& file1, const wxString& file2, const wxString&**

*file3*)

Concatenates *file1* and *file2* to *file3*, returning true if successful.

### **::wxCopyFile**

**bool wxCopyFile(const wxString& *file1*, const wxString& *file2*, bool *overwrite* = true)**

Copies *file1* to *file2*, returning true if successful. If *overwrite* parameter is true (default), the destination file is overwritten if it exists, but if *overwrite* is false, the functions fails in this case.

### **::wxGetCwd**

**wxString wxGetCwd()**

Returns a string containing the current (or working) directory.

### **::wxGetWorkingDirectory**

**wxString wxGetWorkingDirectory(char \*buf=NULL, int sz=1000)**

**NB:** This function is deprecated: use *wxCwd* (p. 1923) instead.

Copies the current working directory into the buffer if supplied, or copies the working directory into new storage (which you *must* delete yourself) if the buffer is NULL.

sz is the size of the buffer if supplied.

### **::wxGetTempFileName**

**char \* wxGetTempFileName(const wxString& *prefix*, char \*buf=NULL)**

**bool wxGetTempFileName(const wxString& *prefix*, wxString& *buf*)**

**NB:** These functions are obsolete, please use *wxFileName::CreateTempFileName* (p. 615) instead.

### **::wxIsWild**

**bool wxIsWild(const wxString& *pattern*)**

Returns true if the pattern contains wildcards. See *wxMatchWild* (p. 1923).

### **::wxMatchWild**

**bool wxMatchWild(const wxString& *pattern*, const wxString& *text*, bool *dot\_special*)**

Returns true if the *pattern* matches the *text*; if *dot\_special* is true, filenames beginning with a dot are not matched with wildcard characters. See *wxIsWild* (p. 1923).

**::wxMkdir****bool wxMkdir(const wxString& dir, int perm = 0777)**

Makes the directory *dir*, returning true if successful.

*perm* is the access mask for the directory for the systems on which it is supported (Unix) and doesn't have any effect on the other ones.

**::wxParseCommonDialogsFilter****int wxParseCommonDialogsFilter(const wxString& wildCard, wxArrayString& descriptions, wxArrayString& filters)**

Parses the *wildCard*, returning the number of filters. Returns 0 if none or if there's a problem. The arrays will contain an equal number of items found before the error. On platforms where native dialogs handle only one filter per entry, entries in arrays are automatically adjusted. *wildCard* is in the form: "All files (\*) | \* | Image Files (\*.jpeg \*.png) | \*.jpg;\*.png"

**::wxRemoveFile****bool wxRemoveFile(const wxString& file)**

Removes *file*, returning true if successful.

**::wxRenameFile****bool wxRenameFile(const wxString& file1, const wxString& file2, bool overwrite = true)**

Renames *file1* to *file2*, returning true if successful.

If *overwrite* parameter is true (default), the destination file is overwritten if it exists, but if *overwrite* is false, the functions fails in this case.

**::wxRmdir****bool wxRmdir(const wxString& dir, int flags=0)**

Removes the directory *dir*, returning true if successful. Does not work under VMS.

The *flags* parameter is reserved for future use.

Please notice that there is also a `wxRmdir()` function which simply wraps the standard POSIX `rmdir()` function and so return an integer error code instead of a boolean value (but otherwise is currently identical to `wxRmdir`), don't confuse these two functions.

**::wxSetWorkingDirectory****bool wxSetWorkingDirectory(const wxString& dir)**

Sets the current working directory, returning true if the operation succeeded. Under MS Windows, the current drive is also changed if *dir* contains a drive specification.

### **::wxSplitPath**

**void wxSplitPath(const char \* fullname, wxString \* path, wxString \* name, wxString \* ext)**

**NB:** This function is obsolete, please use `wxFileName::SplitPath` (p. 627) instead.

This function splits a full file name into components: the path (including possible disk/drive specification under Windows), the base name and the extension. Any of the output parameters (*path*, *name* or *ext*) may be NULL if you are not interested in the value of a particular component.

`wxSplitPath()` will correctly handle filenames with both DOS and Unix path separators under Windows, however it will not consider backslashes as path separators under Unix (where backslash is a valid character in a filename).

On entry, *fullname* should be non-NULL (it may be empty though).

On return, *path* contains the file path (without the trailing separator), *name* contains the file name and *ext* contains the file extension without leading dot. All three of them may be empty if the corresponding component is. The old contents of the strings pointed to by these parameters will be overwritten in any case (if the pointers are not NULL).

### **::wxTransferFileToStream**

**bool wxTransferFileToStream(const wxString& filename, ostream& stream)**

Copies the given file to *stream*. Useful when converting an old application to use streams (within the document/view framework, for example).

#### **Include files**

<wx/docview.h>

### **::wxTransferStreamToFile**

**bool wxTransferStreamToFile(istream& stream const wxString& filename)**

Copies the given stream to the file *filename*. Useful when converting an old application to use streams (within the document/view framework, for example).

#### **Include files**

<wx/docview.h>

## **Network, user and OS functions**

The functions in this section are used to retrieve information about the current computer

and/or user characteristics.

### **::wxGetEmailAddress**

**wxString wxGetEmailAddress()**

**bool wxGetEmailAddress(char \* buf, int sz)**

Copies the user's email address into the supplied buffer, by concatenating the values returned by *wxGetFullHostName* (p. 1926) and *wxGetUserId* (p. 1928).

Returns true if successful, false otherwise.

#### **Include files**

<wx/utils.h>

### **::wxGetFreeMemory**

**wxMemorySize wxGetFreeMemory()**

Returns the amount of free memory in bytes under environments which support it, and -1 if not supported or failed to perform measurement.

#### **Include files**

<wx/utils.h>

### **::wxGetFullHostName**

**wxString wxGetFullHostName()**

Returns the FQDN (fully qualified domain host name) or an empty string on error.

#### **See also**

*wxGetHostName* (p. 1927)

#### **Include files**

<wx/utils.h>

### **::wxGetHomeDir**

**wxString wxGetHomeDir()**

Return the (current) user's home directory.

#### **See also**

*wxGetUserHome* (p. 1928)

*wxStandardPaths* (p. 1522)

**Include files**

<wx/utils.h>

**::wxGetHostName**

**wxString wxGetHostName()**

**bool wxGetHostName(char \* buf, int sz)**

Copies the current host machine's name into the supplied buffer. Please note that the returned name is *not* fully qualified, i.e. it does not include the domain name.

Under Windows or NT, this function first looks in the environment variable `SYSTEM_NAME`; if this is not found, the entry **HostName** in the **wxWidgets** section of the `WIN.INI` file is tried.

The first variant of this function returns the hostname if successful or an empty string otherwise. The second (deprecated) function returns true if successful, false otherwise.

**See also**

*wxGetFullHostName* (p. 1926)

**Include files**

<wx/utils.h>

**::wxGetOsDescription**

**wxString wxGetOsDescription()**

Returns the string containing the description of the current platform in a user-readable form. For example, this function may return strings like `Windows NT Version 4.0` or `Linux 2.2.2 i386`.

**See also**

*::wxGetOsVersion* (p. 1927)

**Include files**

<wx/utils.h>

**::wxGetOsVersion**

**wxOperatingSystemId wxGetOsVersion(int \*major = NULL, int \*minor = NULL)**

Gets the version and the operating system ID for currently running OS. See *wxPlatformInfo* (p. 1185) for more details about `wxOperatingSystemId`.

**See also**

*::wxGetOsDescription* (p. 1927), *wxPlatformInfo* (p. 1185)

**Include files**

<wx/utils.h>

**::wxIsPlatformLittleEndian**

**bool wxIsPlatformLittleEndian()**

Returns `true` if the current platform is little endian (instead of big endian). The check is performed at run-time.

**See also**

*Byte order macros* (p. 1963)

**Include files**

<wx/utils.h>

**::wxIsPlatform64Bit**

**bool wxIsPlatform64Bit()**

Returns `true` if the operating system the program is running under is 64 bit. The check is performed at run-time and may differ from the value available at compile-time (at compile-time you can just check if `sizeof(void*)==8`) since the program could be running in emulation mode or in a mixed 32/64 bit system (bi-architecture operating system).

Very important: this function is not 100% reliable on some systems given the fact that there isn't always a standard way to do a reliable check on the OS architecture.

**Include files**

<wx/utils.h>

**::wxGetUserHome**

**const wxChar \* wxGetUserHome(const wxString& user = "")**

Returns the home directory for the given user. If the username is empty (default value), this function behaves like *wxGetHomeDir* (p. 1926).

**Include files**

<wx/utils.h>

**::wxGetUserId**

**wxString wxGetUserId()**



**bool wxGetUserId(char \* buf, int sz)**

This function returns the "user id" also known as "login name" under Unix i.e. something like "jsmith". It uniquely identifies the current user (on this system).

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 **wxWidgets** section of the WIN.INI file is tried.

The first variant of this function returns the login name if successful or an empty string otherwise. The second (deprecated) function returns true if successful, false otherwise.

**See also**

*wxGetUserName* (p. 1929)

**Include files**

<wx/utils.h>

**::wxGetUserName**

**wxString wxGetUserName()**

**bool wxGetUserName(char \* buf, int sz)**

This function returns the full user name (something like "Mr. John Smith").

Under Windows or NT, this function looks for the entry **UserName** in the **wxWidgets** section of the WIN.INI file. If PenWindows is running, the entry **Current** in the section **User** of the PENWIN.INI file is used.

The first variant of this function returns the user name if successful or an empty string otherwise. The second (deprecated) function returns `true` if successful, `false` otherwise.

**See also**

*wxGetUserId* (p. 1928)

**Include files**

<wx/utils.h>

## String functions

**::copystring**

**char \* copystring(const char \*s)**

Makes a copy of the string *s* using the C++ new operator, so it can be deleted with the *delete* operator.

This function is deprecated, use *wxString* (p. 1553) class instead.

### **::wxGetTranslation**

```
const wxChar * wxGetTranslation(const wxChar* str, const wxChar* domain =  
NULL)
```

```
const wxChar * wxGetTranslation(const wxChar* str, const wxChar* strPlural, size_t  
n, const wxChar* domain = NULL)
```

This function returns the translation of string *str* in the current *locale* (p. 1011). If the string is not found in any of the loaded message catalogs (see *internationalization overview* (p. 2062)), the original string is returned. In debug build, an error message is logged -- this should help to find the strings which were not yet translated. If *domain* is specified then only that domain/catalog is searched for a matching string. As this function is used very often, an alternative (and also common in Unix world) syntax is provided: the *\_()* (p. 1933) macro is defined to do the same thing as *wxGetTranslation*.

The second form is used when retrieving translation of string that has different singular and plural form in English or different plural forms in some other language. It takes two extra arguments: as above, *str* parameter must contain the singular form of the string to be converted and is used as the key for the search in the catalog. The *strPlural* parameter is the plural form (in English). The parameter *n* is used to determine the plural form. If no message catalog is found *str* is returned if 'n == 1', otherwise *strPlural*.

See GNU gettext manual

([http://www.gnu.org/manual/gettext/html\\_chapter/gettext\\_10.html#SEC150](http://www.gnu.org/manual/gettext/html_chapter/gettext_10.html#SEC150)) for additional information on plural forms handling. For a shorter alternative see the *wxPLURAL()* (p. 1933) macro.

Both versions call *wxLocale::GetString* (p. 1015).

Note that this function is not suitable for literal strings in Unicode builds, since the literal strings must be enclosed into *\_T()* (p. 1934) or *wxT* (p. 1932) macro which makes them unrecognised by *xgettext*, and so they are not extracted to the message catalog. Instead, use the *\_()* (p. 1933) and *wxPLURAL* (p. 1933) macro for all literal strings.

### **::wxIsEmpty**

```
bool wxIsEmpty(const char * p)
```

Returns *true* if the pointer is either *NULL* or points to an empty string, *false* otherwise.

### **::wxStrcmp**

```
int wxStrcmp(const char *p1, const char *p2)
```

Returns a negative value, 0, or positive value if *p1* is less than, equal to or greater than *p2*. The comparison is case-sensitive.

This function complements the standard C function *strcmp()* which performs

case-insensitive comparison.

### **::wxStricmp**

**int wxStricmp(const char \*p1, const char \*p2)**

Returns a negative value, 0, or positive value if *p1* is less than, equal to or greater than *p2*. The comparison is case-insensitive.

This function complements the standard C function *strcmp()* which performs case-sensitive comparison.

### **::wxStringEq**

**bool wxStringEq(const wxString& s1, const wxString& s2)**

**NB:** This function is obsolete, use *wxString* (p. 1553) instead.

A macro defined as:

```
#define wxStringEq(s1, s2) (s1 && s2 && (strcmp(s1, s2) == 0))
```

### **::wxStringMatch**

**bool wxStringMatch(const wxString& s1, const wxString& s2,  
 bool subString = true, bool exact = false)**

**NB:** This function is obsolete, use *wxString::Find* (p. 1565) instead.

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.

### **::wxStringTokenize**

**wxArrayString wxStringTokenize(const wxString& str,  
 const wxString& delims = wxDEFAULT\_DELIMITERS,  
 wxStringTokenizerMode mode = wxTOKEN\_DEFAULT)**

This is a convenience function wrapping *wxStringTokenizer* (p. 1583) which simply returns all tokens found in the given *str* in an array.

Please see *wxStringTokenizer::wxStringTokenizer* (p. 1584) for the description of the other parameters.

### **::wxStrlen**

**size\_t wxStrlen(const char \*p)**

This is a safe version of standard function *strlen()*: it does exactly the same thing (i.e. returns the length of the string) except that it returns 0 if *p* is the *NULL* pointer.

## **::wxSnprintf**

**int wxSnprintf(wxChar \*buf, size\_t len, const wxChar \*format, ...)**

This function replaces the dangerous standard function `sprintf()` and is like `snprintf()` available on some platforms. The only difference with `sprintf()` is that an additional argument - buffer size - is taken and the buffer is never overflowed.

Returns the number of characters copied to the buffer or -1 if there is not enough space.

### **See also**

*wxVsnprintf* (p. 1933), *wxString::Printf* (p. 1570)

## **wxT**

**wxChar wxT(char ch)**

**const wxChar \* wxT(const char \*s)**

`wxT()` is a macro which can be used with character and string literals (in other words, `'x'` or `"foo"`) to automatically convert them to Unicode in Unicode build configuration. Please see the *Unicode overview* (p. 2056) for more information.

```
This macro is simply returns the value passed to it without changes in ASCII build. In fact,
its definition is:#ifdef UNICODE
#define wxT(x) L ## x
#else // !Unicode
#define wxT(x) x
#endif
```

## **wxTRANSLATE**

**const wxChar \* wxTRANSLATE(const char \*s)**

This macro doesn't do anything in the program code -- it simply expands to the value of its argument (except in Unicode build where it is equivalent to `wxT` (p. 1932) which makes it unnecessary to use both `wxTRANSLATE` and `wxT` with the same string which would be really unreadable).

However it does have a purpose and it is to mark the literal strings for the extraction into the message catalog created by `xgettext` program. Usually this is achieved using `_( )` (p. 1933) but that macro not only marks the string for extraction but also expands into *awxGetTranslation* (p. 1930) function call which means that it cannot be used in some situations, notably for static array initialization.

Here is an example which should make it more clear: suppose that you have a static array of strings containing the weekday names and which have to be translated (note that it is a bad example, really, as *wxDateTime* (p. 348) already can be used to get the localized week day names already). If you write

```
static const wxChar * const weekdays[] = { _("Mon"), ..., _("Sun") };
```

```
...  
// use weekdays[n] as usual
```

the code wouldn't compile because the function calls are forbidden in the array initializer. So instead you should do

```
static const wxChar * const weekdays[] = { wxTRANSLATE("Mon"), ...,  
wxTRANSLATE("Sun") };  
...  
// use wxGetTranslation weekdays[n])
```

here.

Note that although the code **would** compile if you simply omit `wxTRANSLATE()` in the above, it wouldn't work as expected because there would be no translations for the weekday names in the program message catalog and `wxGetTranslation` wouldn't find them.

### **::wxVsnprintf**

**int wxVsnprintf(wxChar \*buf, size\_t len, const wxChar \*format, va\_list argPtr)**

The same as `wxSnprintf` (p. 1932) but takes a `va_list` argument instead of arbitrary number of parameters.

Note that if `wxUSE_PRINTF_POS_PARAMS` is set to 1, then this function supports positional arguments (see `wxString::Printf` (p. 1570) for more information). However other functions of the same family (`wxPrintf`, `wxSprintf`, `wxFprintf`, `wxVfprintf`, `wxVprintf`, `wxVsprintf`) currently do not to support positional parameters even when `wxUSE_PRINTF_POS_PARAMS` is 1.

### **See also**

`wxSnprintf` (p. 1932), `wxString::PrintfV` (p. 1571)

—

**const wxChar \* \_(const char \*s)**

This macro expands into a call to `wxGetTranslation` (p. 1930) function, so it marks the message for the extraction by `xgettext` just as `wxTRANSLATE` (p. 1932) does, but also returns the translation of the string for the current locale during execution.

Don't confuse this macro with `_T()` (p. 1934)!

### **wxPLURAL**

**const wxChar \* wxPLURAL(const char \*sing, const char \*plur, size\_t n)**

This macro is identical to `_(/)` (p. 1933) but for the plural variant of `wxGetTranslation` (p. 1930).

## **\_T**

**wxChar \_T(char ch)**

**const wxChar \* \_T(const wxChar ch)**

This macro is exactly the same as *wxT* (p. 1932) and is defined in *wxWidgets* simply because it may be more intuitive for Windows programmers as the standard Win32 headers also define it (as well as yet another name for the same macro which is *\_TEXT()*).

Don't confuse this macro with *\_()* (p. 1933)!

## **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.

### **::wxAboutBox**

**void wxAboutBox(const wxAboutDialogInfo& info)**

This function shows the standard about dialog containing the information specified in *info*. If the current platform has a native about dialog which is capable of showing all the fields in *info*, the native dialog is used, otherwise the function falls back to the generic *wxWidgets* version of the dialog, i.e. does the same thing as *wxGenericAboutBox()* (p. 1937).

```
Here is an example of how this function may be used:
void
MyFrame::ShowSimpleAboutDialog(wxCommandEvent& WXUNUSED(event))
{
    wxAboutDialogInfo info;
    info.SetName(_("My Program"));
    info.SetVersion(_("1.2.3 Beta"));
    info.SetDescription(_("This program does something great."));
    info.SetCopyright(_T("(C) 2007 Me <my@email.address>"));

    wxAboutBox(info);
}
```

Please see the *dialogs sample* (p. 2033) for more examples of using this function and *wxAboutDialogInfo* (p. 18) for the description of the information which can be shown in the about dialog.

### **Include files**

<wx/aboutdlg.h>

### **::wxBeginBusyCursor**

**void wxBeginBusyCursor(wxCursor \*cursor = wxHOURLASS\_CURSOR)**

Changes the cursor to the given cursor for all windows in the application. Use *wxEndBusyCursor* (p. 1937) to revert the cursor back to its previous state. These two calls can be nested, and a counter ensures that only the outer calls take effect.

See also *wxIsBusy* (p. 1942), *wxBusyCursor* (p. 162).

#### **Include files**

<wx/utils.h>

#### **::wxBell**

**void wxBell()**

Ring the system bell.

#### **Include files**

<wx/utils.h>

#### **::wxCreateFileTipProvider**

**wxTipProvider \* wxCreateFileTipProvider(const wxString& filename, size\_t currentTip)**

This function creates a *wxTipProvider* (p. 1689) which may be used with *wxShowTip* (p. 1943).

*filename*

The name of the file containing the tips, one per line

*currentTip*

The index of the first tip to show - normally this index is remembered between the 2 program runs.

#### **See also**

*Tips overview* (p. 2144)

#### **Include files**

<wx/tipdlg.h>

#### **::wxDirSelector**

**wxString wxDirSelector(const wxString& message = wxDirSelectorPromptStr, const wxString& default\_path = "", long style = 0, const wxPoint& pos = wxDefaultPosition, wxWindow \*parent = NULL)**

Pops up a directory selector dialog. The arguments have the same meaning as those of

`wxDirDialog::wxDirDialog()`. The message is displayed at the top, and the `default_path`, if specified, is set as the initial selection.

The application must check for an empty return value (if the user pressed Cancel). For example:

```
const wxString& dir = wxDirSelector("Choose a folder");
if ( !dir.empty() )
{
    ...
}
```

### Include files

<wx/dirdlg.h>

### ::wxFileSelector

**wxString wxFileSelector(const wxString& message, const wxString& default\_path = "", const wxString& default\_filename = "", const wxString& default\_extension = "", const wxString& wildcard = "\*", int flags = 0, wxWindow \*parent = NULL, int x = -1, int y = -1)**

Pops up a file selector box. In Windows, this is the common file selector dialog. In X, this is a file selector box with the same functionality. The path and filename are distinct elements of a full file pathname. If path is empty, the current directory will be used. If filename is empty, 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 `wxFD_OPEN`, `wxFD_SAVE`, `wxFD_OVERWRITE_PROMPT` or `wxFD_FILE_MUST_EXIST`. Note that `wxFD_MULTIPLE` can only be used with *wxFileDialog* (p. 600) and not here as this function only returns a single file name.

Both the Unix 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.

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 an empty return value (the user pressed Cancel). For example:

```
wxString filename = wxFileSelector("Choose a file to open");
if ( !filename.empty() )
{
    // work with the file
    ...
}
//else: cancelled by user
```



**Include files**

<wx/filedlg.h>

**::wxEndBusyCursor****void wxEndBusyCursor()**

Changes the cursor back to the original cursor, for all windows in the application. Use with *wxBeginBusyCursor* (p. 1934).

See also *wxIsBusy* (p. 1942), *wxBusyCursor* (p. 162).

**Include files**

<wx/utils.h>

**::wxGenericAboutBox****void wxGenericAboutBox(const wxAboutDialogInfo& info)**

This function does the same thing as *wxAboutBox* (p. 1934) except that it always uses the generic wxWidgets version of the dialog instead of the native one. This is mainly useful if you need to customize the dialog by e.g. adding custom controls to it (customizing the native dialog is not currently supported).

See the *dialogs sample* (p. 2033) for an example of about dialog customization.

**See also**

*wxAboutDialogInfo* (p. 18)

**Include files**

<wx/aboutdlg.h>

<wx/generic/aboutdlgg.h>

**::wxGetColourFromUser****wxColour wxGetColourFromUser(wxWindow \*parent, const wxColour& collnit, const wxString& caption = wxEmptyString)**

Shows the colour selection dialog and returns the colour selected by user or invalid colour (use *wxColour::IsOk* (p. 217) to test whether a colour is valid) if the dialog was cancelled.

**Parameters**

*parent*

The parent window for the colour selection dialog

*collnit*

If given, this will be the colour initially selected in the dialog.

*caption*

If given, this will be used for the dialog caption.

#### **Include files**

<wx/colordlg.h>

#### **::wxGetFontFromUser**

**wxFont wxGetFontFromUser(wxWindow \*parent, const wxFont& fontInit, const wxString& caption = wxEmptyString)**

Shows the font selection dialog and returns the font selected by user or invalid font (use *wxFont::IsOk* (p. 663) to test whether a font is valid) if the dialog was cancelled.

#### **Parameters**

*parent*

The parent window for the font selection dialog

*fontInit*

If given, this will be the font initially selected in the dialog.

*caption*

If given, this will be used for the dialog caption.

#### **Include files**

<wx/fontdlg.h>

#### **::wxGetMultipleChoices**

**size\_t wxGetMultipleChoices(  
wxArrayInt& selections,  
const wxString& message,  
const wxString& caption,  
const wxArrayString& aChoices,  
wxWindow \*parent = NULL,  
int x = -1, int y = -1,  
bool centre = true,  
int width=150, int height=200)**

**size\_t wxGetMultipleChoices(  
wxArrayInt& selections,  
const wxString& message,  
const wxString& caption,  
int n, const wxString& choices[],**

```
wxWindow *parent = NULL,  
int x = -1, int y = -1,  
bool centre = true,  
int width=150, int height=200)
```

Pops up a dialog box containing a message, OK/Cancel buttons and a multiple-selection listbox. The user may choose an arbitrary (including 0) number of items in the listbox whose indices will be returned in *selection* array. The initial contents of this array will be used to select the items when the dialog is shown.

You may pass the list of strings to choose from either using *choices* which is an array of *n* strings for the listbox or by using a single *aChoices* parameter of type *wxArrayString* (p. 83).

If *centre* is true, the message text (which may include new line characters) is centred; if false, the message is left-justified.

#### Include files

```
<wx/choicdlg.h>
```

**wxPerl note:** In wxPerl there is just an array reference in place of *selections* and *choices*, and no *selections* parameter; the function returns an array containing the user selections.

#### ::wxGetNumberFromUser

```
long wxGetNumberFromUser( const wxString& message, const wxString& prompt,  
const wxString& caption, long value, long min = 0, long max = 100, wxWindow *parent  
= NULL, const wxPoint& pos = wxDefaultPosition)
```

Shows a dialog asking the user for numeric input. The dialog's title is set to *caption*, it contains a (possibly) multiline *message* above the single line *prompt* and the zone for entering the number.

The number entered must be in the range *min*..*max* (both of which should be positive) and *value* is the initial value of it. If the user enters an invalid value or cancels the dialog, the function will return -1.

Dialog is centered on its *parent* unless an explicit position is given in *pos*.

#### Include files

```
<wx/numdlg.h>
```

#### ::wxGetPasswordFromUser

```
wxString wxGetPasswordFromUser(const wxString& message, const wxString&  
caption = "Input text",  
const wxString& default_value = "", wxWindow *parent = NULL,  
int x = wxDefaultCoord, int y = wxDefaultCoord, bool centre = true)
```

Similar to *wxGetTextFromUser* (p. 1940) but the text entered in the dialog is not shown on

screen but replaced with stars. This is intended to be used for entering passwords as the function name implies.

**Include files**

<wx/textdlg.h>

**::wxGetTextFromUser**

```
wxString wxGetTextFromUser(const wxString& message, const wxString& caption
= "Input text",
const wxString& default_value = "", wxWindow *parent = NULL,
int x = wxDefaultCoord, int y = wxDefaultCoord, bool centre = true)
```

Pop up a dialog box with title set to *caption*, *message*, and a *default\_value*. The user may type in text and press OK to return this text, or press Cancel to return the empty string.

If *centre* is true, the message text (which may include new line characters) is centred; if false, the message is left-justified.

**Include files**

<wx/textdlg.h>

**::wxGetMultipleChoice**

```
int wxGetMultipleChoice(const wxString& message, const wxString& caption, int n,
const wxString& choices[],
int n_sel, int *selection, wxWindow *parent = NULL, int x = -1, int y = -1,
bool centre = true, int width=150, int height=200)
```

Pops up a dialog box containing a message, OK/Cancel buttons and a multiple-selection listbox. The user may choose one or more item(s) and press OK or Cancel.

The number of initially selected choices, and array of the selected indices, are passed in; this array will contain the user selections on exit, with the function returning the number of selections. *selection* must be as big as the number of choices, in case all are selected.

If Cancel is pressed, -1 is returned.

*choices* is an array of *n* strings for the listbox.

If *centre* is true, the message text (which may include new line characters) is centred; if false, the message is left-justified.

**Include files**

<wx/choicdlg.h>

**::wxGetSingleChoice**

```
wxString wxGetSingleChoice(const wxString& message,
const wxString& caption,
```

```
const wxArrayString& aChoices,  
wxWindow *parent = NULL,  
int x = -1, int y = -1,  
bool centre = true,  
int width=150, int height=200)
```

```
wxString wxGetSingleChoice(const wxString& message,  
const wxString& caption,  
int n, const wxString& choices[],  
wxWindow *parent = NULL,  
int x = -1, int y = -1,  
bool centre = true,  
int width=150, int height=200)
```

Pops up a dialog box containing a message, OK/Cancel buttons and a single-selection listbox. The user may choose an item and press OK to return a string or Cancel to return the empty string. Use `wxGetSingleChoiceIndex` (p. 1941) if empty string is a valid choice and if you want to be able to detect pressing Cancel reliably.

You may pass the list of strings to choose from either using `choices` which is an array of *n* strings for the listbox or by using a single `aChoices` parameter of type `wxArrayString` (p. 83).

If `centre` is true, the message text (which may include new line characters) is centred; if false, the message is left-justified.

#### Include files

<wx/choicdlg.h>

**wxPerl note:** In wxPerl there is just an array reference in place of `n` and `choices`.

#### ::wxGetSingleChoiceIndex

```
int wxGetSingleChoiceIndex(const wxString& message,  
const wxString& caption,  
const wxArrayString& aChoices,  
wxWindow *parent = NULL, int x = -1, int y = -1,  
bool centre = true, int width=150, int height=200)
```

```
int wxGetSingleChoiceIndex(const wxString& message,  
const wxString& caption,  
int n, const wxString& choices[],  
wxWindow *parent = NULL, int x = -1, int y = -1,  
bool centre = true, int width=150, int height=200)
```

As `wxGetSingleChoice` but returns the index representing the selected string. If the user pressed cancel, -1 is returned.

#### Include files

<wx/choicdlg.h>

**wxPerl note:** In wxPerl there is just an array reference in place of `nand choices`.

### **::wxGetSingleChoiceData**

```
wxString wxGetSingleChoiceData(const wxString& message,  
const wxString& caption,  
const wxArrayString& aChoices,  
const wxString& client_data[],  
wxWindow *parent = NULL,  
int x = -1, int y = -1,  
bool centre = true, int width=150, int height=200)
```

```
wxString wxGetSingleChoiceData(const wxString& message,  
const wxString& caption,  
int n, const wxString& choices[],  
const wxString& client_data[],  
wxWindow *parent = NULL,  
int x = -1, int y = -1,  
bool centre = true, int width=150, int height=200)
```

As **wxGetSingleChoice** but takes an array of client data pointers corresponding to the strings, and returns one of these pointers or **NULL** if Cancel was pressed. The *client\_data* array must have the same number of elements as *choices* or *aChoices*!

### **Include files**

<wx/choicdlg.h>

**wxPerl note:** In wxPerl there is just an array reference in place of `nand choices`, and the client data array must have the same length as the choices array.

### **::wxIsBusy**

```
bool wxIsBusy()
```

Returns true if between two *wxBeginBusyCursor* (p. 1934) and *wxEndBusyCursor* (p. 1937) calls.

See also *wxBusyCursor* (p. 162).

### **Include files**

<wx/utils.h>

### **::wxMessageBox**

```
int wxMessageBox(const wxString& message, const wxString& caption = "Message",  
int style = wxOK,  
wxWindow *parent = NULL, int x = -1, int y = -1)
```

General purpose message dialog. *style* may be a bit list of the following identifiers:

<code>wxYES_NO</code>	Puts Yes and No buttons on the message box. May be combined with <code>wxCANCEL</code> .
<code>wxCANCEL</code>	Puts a Cancel button on the message box. May only be combined with <code>wxYES_NO</code> or <code>wxOK</code> .
<code>wxOK</code>	Puts an Ok button on the message box. May be combined with <code>wxCANCEL</code> .
<code>wxICON_EXCLAMATION</code>	Displays an exclamation mark symbol.
<code>wxICON_HAND</code>	Displays an error symbol.
<code>wxICON_ERROR</code>	Displays an error symbol - the same as <code>wxICON_HAND</code> .
<code>wxICON_QUESTION</code>	Displays a question mark symbol.
<code>wxICON_INFORMATION</code>	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)
    main_frame->Close();
...
```

*message* may contain newline characters, in which case the message will be split into separate lines, to cater for large messages.

### Include files

`<wx/msgdlg.h>`

### **::wxShowTip**

**bool wxShowTip(wxWindow \*parent, wxTipProvider \*tipProvider, bool showAtStartup = true)**

This function shows a "startup tip" to the user. The return value is the state of the 'Show tips at startup' checkbox.

*parent*

The parent window for the modal dialog

*tipProvider*

An object which is used to get the text of the tips. It may be created with the `wxCreateFileTipProvider` (p. 1935) function.

### *showAtStartup*

Should be true if startup tips are shown, false otherwise. This is used as the initial value for "Show tips at startup" checkbox which is shown in the tips dialog.

#### **See also**

*Tips overview* (p. 2144)

#### **Include files**

<wx/tipdlg.h>

## **Math functions**

#### **Include files**

<wx/math.h>

#### **wxFinite**

**int wxFinite(double x)**

Returns a non-zero value if x is neither infinite or NaN (not a number), returns 0 otherwise.

#### **wxIsNaN**

**bool wxIsNaN(double x)**

Returns a non-zero value if x is NaN (not a number), returns 0 otherwise.

## **GDI functions**

The following are relevant to the GDI (Graphics Device Interface).

#### **Include files**

<wx/gdicmn.h>

#### **wxBITMAP**

**wxBITMAP(bitmapName)**

This macro loads a bitmap from either application resources (on the platforms for which they exist, i.e. Windows and OS2) or from an XPM file. It allows to avoid using `#ifdefs` when creating bitmaps.

#### **See also**

*Bitmaps and icons overview* (p. 2117), *wxICON* (p. 1946)



**Include files**

<wx/gdicmn.h>

**::wxClientDisplayRect**

**void wxClientDisplayRect(int \*x, int \*y, int \*width, int \*height)**

**wxRect wxGetClientDisplayRect()**

Returns the dimensions of the work area on the display. On Windows this means the area not covered by the taskbar, etc. Other platforms are currently defaulting to the whole display until a way is found to provide this info for all window managers, etc.

**::wxColourDisplay**

**bool wxColourDisplay()**

Returns true if the display is colour, false otherwise.

**::wxDisplayDepth**

**int wxDisplayDepth()**

Returns the depth of the display (a value of 1 denotes a monochrome display).

**::wxDisplaySize**

**void wxDisplaySize(int \*width, int \*height)**

**wxSize wxGetDisplaySize()**

Returns the display size in pixels.

**::wxDisplaySizeMM**

**void wxDisplaySizeMM(int \*width, int \*height)**

**wxSize wxGetDisplaySizeMM()**

Returns the display size in millimeters.

**::wxDROP\_ICON**

**wxIconOrCursor wxDROP\_ICON(const char \*name)**

This macro creates either a cursor (MSW) or an icon (elsewhere) with the given name. Under MSW, the cursor is loaded from the resource file and the icon is loaded from XPM file under other platforms.

This macro should be used with *wxDropSource constructor* (p. 559).

**Include files**

<wx/dnd.h>

**wxICON**

**wxICON**(iconName)

This macro loads an icon from either application resources (on the platforms for which they exist, i.e. Windows and OS2) or from an XPM file. It allows to avoid using `#ifdefs` when creating icons.

**See also**

*Bitmaps and icons overview* (p. 2117), *wxBITMAP* (p. 1944)

**Include files**

<wx/gdicmn.h>

**::wxMakeMetafilePlaceable**

**bool wxMakeMetafilePlaceable**(const wxString& filename, int minX, int minY, int maxX, int maxY, float scale=1.0)

Given a filename for an existing, valid metafile (as constructed using *wxMetafileDC* (p. 1109)) makes it into a placeable metafile by prepending a header containing the given bounding box. The bounding box may be obtained from a device context after drawing into it, using the functions *wxDC::MinX*, *wxDC::MinY*, *wxDC::MaxX* and *wxDC::MaxY*.

In addition to adding the placeable metafile header, this function adds the equivalent of the following code to the start of the metafile data:

```
SetMapMode(dc, MM_ANISOTROPIC);
SetWindowOrg(dc, minX, minY);
SetWindowExt(dc, maxX - minX, maxY - minY);
```

This simulates the *wxMM\_TEXT* mapping mode, which *wxWidgets* 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**(const wxCursor& cursor)

Globally sets the cursor; only has an effect on Windows, Mac and GTK+. You should call this function with *wxNullCursor* to restore the system cursor.

See also *wxCursor* (p. 297), *wxWindow::SetCursor* (p. 1837).

## Printer settings

**NB:** These routines are obsolete and should no longer be used!

The following functions are used to control PostScript printing. Under Windows, PostScript output can only be sent to a file.

### Include files

<wx/dcps.h>

### ::wxGetPrinterCommand

**wxString wxGetPrinterCommand()**

Gets the printer command used to print a file. The default is `lpr`.

### ::wxGetPrinterFile

**wxString wxGetPrinterFile()**

Gets the PostScript output filename.

### ::wxGetPrinterMode

**int wxGetPrinterMode()**

Gets the printing mode controlling where output is sent (`PS_PREVIEW`, `PS_FILE` or `PS_PRINTER`). The default is `PS_PREVIEW`.

### ::wxGetPrinterOptions

**wxString wxGetPrinterOptions()**

Gets the additional options for the print command (e.g. specific printer). The default is nothing.

### ::wxGetPrinterOrientation

**int wxGetPrinterOrientation()**

Gets the orientation (`PS_PORTRAIT` or `PS_LANDSCAPE`). The default is `PS_PORTRAIT`.

### ::wxGetPrinterPreviewCommand

**wxString wxGetPrinterPreviewCommand()**

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(const wxString& command)**

Sets the printer command used to print a file. The default is `lpr`.

**::wxSetPrinterFile**

**void wxSetPrinterFile(const wxString& 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(const wxString& 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(const wxString& 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.

## **Clipboard functions**

These clipboard functions are implemented for Windows only. The use of these functions is deprecated and the code is no longer maintained. Use the *wxClipboard* (p. 195) class instead.

### **Include files**

<wx/clipbrd.h>

### **::wxClipboardOpen**

**bool wxClipboardOpen()**

Returns true if this application has already opened the clipboard.

### **::wxCloseClipboard**

**bool wxCloseClipboard()**

Closes the clipboard to allow other applications to use it.

### **::wxEmptyClipboard**

**bool wxEmptyClipboard()**

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, const wxString& formatName, int maxCount)**

Gets the name of a registered clipboard format, and puts it into the buffer *formatName* which is of maximum length *maxCount*. *dataFormat* must not specify a predefined clipboard format.

### **::wxIsClipboardFormatAvailable**

**bool wxIsClipboardFormatAvailable(int dataFormat)**

Returns true if the given data format is available on the clipboard.

### **::wxOpenClipboard**

**bool wxOpenClipboard()**

Opens the clipboard for passing data to it or getting data from it.

### **::wxRegisterClipboardFormat**

**int wxRegisterClipboardFormat(const wxString& 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.

## Miscellaneous functions

**wxCONCAT****wxCONCAT(x, y)**

This macro returns the concatenation of two tokens *x* and *y*.

**wxDYNLIB\_FUNCTION****wxDYNLIB\_FUNCTION(type, name, dynlib)**

When loading a function from a DLL you always have to cast the returned `void *` pointer to the correct type and, even more annoyingly, you have to repeat this type twice if you want to declare and define a function pointer all in one line

This macro makes this slightly less painful by allowing you to specify the type only once, as the first parameter, and creating a variable of this type named after the function but with `pfn` prefix and initialized with the function *name* from the *wxDynamicLibrary* (p. 564) *dynlib*.

**Parameters***type*

the type of the function

*name*

the name of the function to load, not a string (without quotes, it is quoted automatically by the macro)

*dynlib*

the library to load the function from

## **wxEXPLICIT**

`wxEXPLICIT` is a macro which expands to the C++ `explicit` keyword if the compiler supports it or nothing otherwise. Thus, it can be used even in the code which might have to be compiled with an old compiler without support for this language feature but still take advantage of it when it is available.

## **::wxGetKeyState**

**bool wxGetKeyState(wxKeyCode key)**

For normal keys, returns `true` if the specified key is currently down.

For toggleable keys (Caps Lock, Num Lock and Scroll Lock), return `true` if the key is toggled such that its LED indicator is lit. There is currently no way to test whether toggleable keys are up or down.

Even though there are virtual key codes defined for mouse buttons, they cannot be used with this function currently.

### **Include files**

<wx/utils.h>

## **wxLL**

**wxLongLong\_t wxLL(number)**

This macro is defined for the platforms with a native 64 bit integer type and allows to define 64 bit compile time constants:

```
#ifdef wxLongLong_t
    wxLongLong_t ll = wxLL(0x1234567890abcdef);
#endif
```

### **Include files**

<wx/longlong.h>

### **See also**

`wxULL` (p. 1962), `wxLongLong` (p. 1032)

## **wxLongLongFmtSpec**

This macro is defined to contain the `printf()` format specifier using which 64 bit integer numbers (i.e. those of type `wxLongLong_t`) can be printed. Example of using it:



```
#ifndef wxLongLong_t
    wxLongLong_t ll = wxLL(0x1234567890abcdef);
    printf("Long long = %" wxLongLongFmtSpec "x\n", ll);
#endif
```

**See also**

`wxLL` (p. 1952)

**Include files**

<wx/longlong.h>

**::wxNewId****long wxNewId()**

Generates an integer identifier unique to this run of the program.

**Include files**

<wx/utils.h>

**wxON\_BLOCK\_EXIT**

**wxON\_BLOCK\_EXIT0**(*func*) **wxON\_BLOCK\_EXIT1**(*func*, *p1*)  
**wxON\_BLOCK\_EXIT2**(*func*, *p1*, *p2*)

This family of macros allows to ensure that the global function *func* with 0, 1, 2 or more parameters (up to some implementation-defined limit) is executed on scope exit, whether due to a normal function return or because an exception has been thrown. A typical example of its usage:

```
void *buf = malloc(size);
wxON_BLOCK_EXIT1(free, buf);
```

Please see the original article by Andrei Alexandrescu and Petru Marginean published in December 2000 issue of *C/C++ Users Journal* for more details.

**Include files**

<wx/scopeguard.h>

**See also**

`wxON_BLOCK_EXIT_OBJ` (p. 1953)

**wxON\_BLOCK\_EXIT\_OBJ**

**wxON\_BLOCK\_EXIT\_OBJ0**(*obj*, *method*) **wxON\_BLOCK\_EXIT\_OBJ1**(*obj*, *method*,  
*p1*) **wxON\_BLOCK\_EXIT\_OBJ2**(*obj*, *method*, *p1*, *p2*)

This family of macros is similar to `wxON_BLOCK_EXIT` (p. 1953) but calls a method of the given object instead of a free function.

**Include files**

<wx/scopeguard.h>

**::wxRegisterId**

**void wxRegisterId(long id)**

Ensures that ids subsequently generated by **NewId** do not clash with the given **id**.

**Include files**

<wx/utils.h>

**::wxDDECleanUp**

**void wxDDECleanUp()**

Called when wxWidgets exits, to clean up the DDE system. This no longer needs to be called by the application.

See also *wxDDEInitialize* (p. 1954).

**Include files**

<wx/dde.h>

**::wxDDEInitialize**

**void wxDDEInitialize()**

Initializes the DDE system. May be called multiple times without harm.

This no longer needs to be called by the application: it will be called by wxWidgets if necessary.

See also *wxDDEServer* (p. 482), *wxDDEClient* (p. 477), *wxDDEConnection* (p. 478), *wxDDECleanUp* (p. 1954).

**Include files**

<wx/dde.h>

**::wxEnableTopLevelWindows**

**void wxEnableTopLevelWindows(bool enable = true)**

This function enables or disables all top level windows. It is used by *::wxSafeYield* (p. 1912).

**Include files**

<wx/utils.h>

**::wxFindMenuItemId**

```
int wxFindMenuItemId(wxFrame *frame, const wxString& menuString, const
wxString& itemString)
```

Find a menu item identifier associated with the given frame's menu bar.

**Include files**

<wx/utils.h>

**::wxFindWindowByLabel**

```
wxWindow * wxFindWindowByLabel(const wxString& label, wxWindow
*parent=NULL)
```

**NB:** This function is obsolete, please use `wxWindow::FindWindowByLabel` (p. 1807) instead.

Find a window by its label. Depending on the type of window, the label may be a window title or panel item label. If *parent* is NULL, the search will start from all top-level frames and dialog boxes; if non-NULL, the search will be limited to the given window hierarchy. The search is recursive in both cases.

**Include files**

<wx/utils.h>

**::wxFindWindowByName**

```
wxWindow * wxFindWindowByName(const wxString& name, wxWindow
*parent=NULL)
```

**NB:** This function is obsolete, please use `wxWindow::FindWindowByName` (p. 1808) instead.

Find a window by its name (as given in a window constructor or **Create** function call). If *parent* is NULL, the search will start from all top-level frames and dialog boxes; if non-NULL, the search will be limited to the given window hierarchy. The search is recursive in both cases.

If no such named window is found, **wxFindWindowByLabel** is called.

**Include files**

<wx/utils.h>

**::wxFindWindowAtPoint**

```
wxWindow * wxFindWindowAtPoint(const wxPoint& pt)
```

Find the deepest window at the given mouse position in screen coordinates, returning the

window if found, or NULL if not.

### **::wxFindWindowAtPointer**

**wxWindow \* wxFindWindowAtPointer(wxPoint& pt)**

Find the deepest window at the mouse pointer position, returning the window and current pointer position in screen coordinates.

### **::wxGetActiveWindow**

**wxWindow \* wxGetActiveWindow()**

Gets the currently active window (implemented for MSW and GTK only currently, always returns NULL in the other ports).

#### **Include files**

<wx/window.h>

### **::wxGetBatteryState**

**wxBatteryState wxGetBatteryState()**

Returns battery state as one of `wxBATTERY_NORMAL_STATE`, `wxBATTERY_LOW_STATE`, `wxBATTERY_CRITICAL_STATE`, `wxBATTERY_SHUTDOWN_STATE` or `wxBATTERY_UNKNOWN_STATE`. `wxBATTERY_UNKNOWN_STATE` is also the default on platforms where this feature is not implemented (currently everywhere but MS Windows).

#### **Include files**

<wx/utils.h>

### **::wxGetDisplayName**

**wxString wxGetDisplayName()**

Under X only, returns the current display name. See also *wxSetDisplayName* (p. 1960).

#### **Include files**

<wx/utils.h>

### **::wxGetPowerType**

**wxPowerType wxGetPowerType()**

Returns the type of power source as one of `wxPOWER_SOCKET`, `wxPOWER_BATTERY` or `wxPOWER_UNKNOWN`. `wxPOWER_UNKNOWN` is also the default on platforms where this feature is not implemented (currently everywhere but MS Windows).

**Include files**

<wx/utils.h>

**::wxGetMousePosition****wxPoint wxGetMousePosition()**

Returns the mouse position in screen coordinates.

**Include files**

<wx/utils.h>

**::wxGetMouseState****wxMouseState wxGetMouseState()**

Returns the current state of the mouse. Returns a wxMouseState instance that contains the current position of the mouse pointer in screen coordinates, as well as boolean values indicating the up/down status of the mouse buttons and the modifier keys.

**Include files**

<wx/utils.h>

wxMouseState has the following interface:

```
class wxMouseState
{
public:
    wxMouseState();

    wxCoord    GetX();
    wxCoord    GetY();

    bool        LeftDown();
    bool        MiddleDown();
    bool        RightDown();

    bool        ControlDown();
    bool        ShiftDown();
    bool        AltDown();
    bool        MetaDown();
    bool        CmdDown();

    void        SetX(wxCoord x);
    void        SetY(wxCoord y);

    void        SetLeftDown(bool down);
    void        SetMiddleDown(bool down);
    void        SetRightDown(bool down);

    void        SetControlDown(bool down);
    void        SetShiftDown(bool down);
```

```
        void          SetAltDown(bool down);  
        void          SetMetaDown(bool down);  
};
```

### **::wxGetResource**

**bool wxGetResource(const wxString& section, const wxString& entry, const wxString& \*value, const wxString& file = NULL)**

**bool wxGetResource(const wxString& section, const wxString& entry, float \*value, const wxString& file = NULL)**

**bool wxGetResource(const wxString& section, const wxString& entry, long \*value, const wxString& file = NULL)**

**bool wxGetResource(const wxString& section, const wxString& entry, int \*value, const wxString& file = NULL)**

Gets a resource value from the resource database (for example, WIN.INI, or .Xdefaults). If *file* is NULL, WIN.INI or .Xdefaults is used, otherwise the specified file is used.

Under X, if an application class (wxApp::GetClassName) has been defined, it is appended to the string /usr/lib/X11/app-defaults/ to try to find an applications default file when merging all resource databases.

The reason for passing the result in an argument is that it can be convenient to define a default value, which gets overridden if the value exists in the resource file. It saves a separate test for that resource's existence, and it also allows the overloading of the function for different types.

See also *wxWriteResource* (p. 1962), *wxConfigBase* (p. 262).

### **Include files**

<wx/utils.h>

### **::wxGetStockLabel**

**wxString wxGetStockLabel(wxWindowID id, bool withCodes = true, const wxString& accelerator = wxEmptyString)**

Returns label that should be used for given *id* element.

### **Parameters**

*id*

given id of the *wxMenuItem* (p. 1099), *wxButton* (p. 164), *wxToolBar* (p. 1694) tool, etc.

*withCodes*

if false then strip accelerator code from the label; usefull for getting labels without

accelerator char code like for toolbar tooltip or under platforms without traditional keyboard like smartphones

*accelerator*

optional accelerator string automatically added to label; useful for building labels for *wxMenuItem* (p. 1099)

#### **Include files**

<wx/stockitem.h>

### **::wxGetTopLevelParent**

**wxWindow \* wxGetTopLevelParent(wxWindow \*win)**

Returns the first top level parent of the given window, or in other words, the frame or dialog containing it, or `NULL`.

#### **Include files**

<wx/window.h>

### **::wxLaunchDefaultBrowser**

**bool wxLaunchDefaultBrowser(const wxString& url, int flags = 0)**

Open the *url* in user's default browser. If *flags* parameter contains `wxBROWSER_NEW_WINDOW` flag, a new window is opened for the URL (currently this is only supported under Windows).

Returns `true` if the application was successfully launched.

Note that for some configurations of the running user, the application which is launched to open the given URL may be URL-dependent (e.g. a browser may be used for local URLs while another one may be used for remote URLs).

#### **Include files**

<wx/utils.h>

### **::wxLoadUserResource**

**wxString wxLoadUserResource(const wxString& resourceName, const wxString& resourceType="TEXT")**

Loads a user-defined Windows resource as a string. If the resource is found, the function creates a new character array and copies the data into it. A pointer to this data is returned. If unsuccessful, `NULL` is returned.

The resource must be defined in the `.rc` file using the following syntax:

```
myResource TEXT file.ext
```

where `file.ext` is a file that the resource compiler can find.

This function is available under Windows only.

#### **Include files**

<wx/utils.h>

#### **::wxPostDelete**

**void wxPostDelete(wxObject \*object)**

Tells the system to delete the specified object when all other events have been processed. In some environments, it is necessary to use this instead of deleting a frame directly with the delete operator, because some GUIs will still send events to a deleted window.

Now obsolete: use `wxWindow::Close` (p. 1802) instead.

#### **Include files**

<wx/utils.h>

#### **::wxPostEvent**

**void wxPostEvent(wxEvtHandler \*dest, wxEvent& event)**

In a GUI application, this function posts *event* to the specified *dest* object using `wxEvtHandler::AddPendingEvent` (p. 576). Otherwise, it dispatches *event* immediately using `wxEvtHandler::ProcessEvent` (p. 580). See the respective documentation for details (and caveats).

#### **Include files**

<wx/app.h>

#### **::wxSetDisplayName**

**void wxSetDisplayName(const wxString& displayName)**

Under X only, sets the current display name. This is the X host and display name such as "colonsay:0.0", and the function indicates which display should be used for creating windows from this point on. Setting the display within an application allows multiple displays to be used.

See also `wxGetDisplayName` (p. 1956).

#### **Include files**

<wx/utils.h>

#### **::wxStripMenuCodes**



**wxString wxStripMenuCodes(const wxString& str, int flags = wxStrip\_All)**

Strips any menu codes from *str* and returns the result.

By default, the function strips both the mnemonics character ( ' & ' ) which is used to indicate a keyboard shortcut, and the accelerators, which are used only in the menu items and are separated from the main text by the `\t` (TAB) character. By using *flags* of `wxStrip_Mnemonics` or `wxStrip_Accel` to strip only the former or the latter part, respectively.

Notice that in most cases `wxMenuItem::GetLabelFromText` (p. 1103) or `wxControl::GetLabelText` (p. 286) can be used instead.

#### Include files

<wx/utils.h>

### wxSTRINGIZE

**wxSTRINGIZE(x)**

Returns the string representation of the given symbol which can be either a literal or a macro (hence the advantage of using this macro instead of the standard preprocessor `#` operator which doesn't work with macros).

Notice that this macro always produces a `char` string, use `wxSTRINGIZE_T` (p. 1961) to build a wide string Unicode build.

#### See also

`wxCONCAT` (p. 1951)

### wxSTRINGIZE\_T

**wxSTRINGIZE\_T(x)**

Returns the string representation of the given symbol as either an ASCII or Unicode string, depending on the current build. This is the Unicode-friendly equivalent of `wxSTRINGIZE` (p. 1961).

### wxSUPPRESS\_GCC\_PRIVATE\_DTOR\_WARNING

**wxSUPPRESS\_GCC\_PRIVATE\_DTOR\_WARNING(name)**

GNU C++ compiler gives a warning for any class whose destructor is private unless it has a friend. This warning may sometimes be useful but it doesn't make sense for reference counted class which always delete themselves (hence destructor should be private) but don't necessarily have any friends, so this macro is provided to disable the warning in such case. The *name* parameter should be the name of the class but is only used to construct a unique friend class name internally. Example of using the macro:

```
class RefCounted
```

```
{
public:
    RefCounted() { m_nRef = 1; }
    void IncRef() { m_nRef++ ; }
    void DecRef() { if ( !--m_nRef ) delete this; }

private:
    ~RefCounted() { }

    wxSUPPRESS_GCC_PRIVATE_DTOR(RefCounted)
};
```

Notice that there should be no semicolon after this macro.

## **wxULL**

**wxLongLong\_t wxULL(*number*)**

This macro is defined for the platforms with a native 64 bit integer type and allows to define unsigned 64 bit compile time constants:

```
#ifdef wxLongLong_t
    unsigned wxLongLong_t ll = wxULL(0x1234567890abcdef);
#endif
```

## **Include files**

<wx/longlong.h>

## **See also**

*wxLL* (p. 1952), *wxLongLong* (p. 1032)

## **wxVaCopy**

**void wxVaCopy(va\_list *argptrDst*, va\_list *argptrSrc*)**

This macro is the same as the standard C99 `va_copy` for the compilers which support it or its replacement for those that don't. It must be used to preserve the value of a `va_list` object if you need to use it after passing it to another function because it can be modified by the latter.

As with `va_start`, each call to `wxVaCopy` must have a matching `va_end`.

## **::wxWriteResource**

**bool wxWriteResource(const wxString& *section*, const wxString& *entry*, const wxString& *value*, const wxString& *file* = NULL)**

**bool wxWriteResource(const wxString& *section*, const wxString& *entry*, float *value*, const wxString& *file* = NULL)**

**bool wxWriteResource(const wxString& *section*, const wxString& *entry*, long *value*,**

**const wxString& file = NULL)**

**bool wxWriteResource(const wxString& section, const wxString& entry, int value, const wxString& file = NULL)**

Writes a resource value into the resource database (for example, WIN.INI, or .Xdefaults). If *file* is NULL, WIN.INI or .Xdefaults is used, otherwise the specified file is used.

Under X, the resource databases are cached until the internal function **wxFlushResources** is called automatically on exit, when all updated resource databases are written to their files.

Note that it is considered bad manners to write to the .Xdefaults file under Unix, although the WIN.INI file is fair game under Windows.

See also *wxGetResource* (p. 1958), *wxConfigBase* (p. 262).

#### **Include files**

<wx/utils.h>

**\_\_WXFUNCTION\_\_**

**\_\_WXFUNCTION\_\_()**

This macro expands to the name of the current function if the compiler supports any of **\_\_FUNCTION\_\_**, **\_\_func\_\_** or equivalent variables or macros or to NULL if none of them is available.

## **Byte order macros**

The endian-ness issues (that is the difference between big-endian and little-endian architectures) are important for the portable programs working with the external binary data (for example, data files or data coming from network) which is usually in some fixed, platform-independent format. The macros are helpful for transforming the data to the correct format.

### **wxINTXX\_SWAP\_ALWAYS**

**wxInt32 wxINT32\_SWAP\_ALWAYS(wxInt32 value)**

**wxUInt32 wxUINT32\_SWAP\_ALWAYS(wxUInt32 value)**

**wxInt16 wxINT16\_SWAP\_ALWAYS(wxInt16 value)**

**wxUInt16 wxUINT16\_SWAP\_ALWAYS(wxUInt16 value)**

These macros will swap the bytes of the *value* variable from little endian to big endian or vice versa unconditionally, i.e. independently of the current platform.

### **wxINTXX\_SWAP\_ON\_BE**

**wxInt32 wxINT32\_SWAP\_ON\_BE(wxInt32 value)**

**wxUInt32 wxUINT32\_SWAP\_ON\_BE(wxUInt32 value)**

**wxInt16 wxINT16\_SWAP\_ON\_BE(wxInt16 value)**

**wxUInt16 wxUINT16\_SWAP\_ON\_BE(wxUInt16 value)**

This macro will swap the bytes of the *value* variable from little endian to big endian or vice versa if the program is compiled on a big-endian architecture (such as Sun work stations). If the program has been compiled on a little-endian architecture, the value will be unchanged.

Use these macros to read data from and write data to a file that stores data in little-endian (for example Intel i386) format.

**wxINTXX\_SWAP\_ON\_LE**

**wxInt32 wxINT32\_SWAP\_ON\_LE(wxInt32 value)**

**wxUInt32 wxUINT32\_SWAP\_ON\_LE(wxUInt32 value)**

**wxInt16 wxINT16\_SWAP\_ON\_LE(wxInt16 value)**

**wxUInt16 wxUINT16\_SWAP\_ON\_LE(wxUInt16 value)**

This macro will swap the bytes of the *value* variable from little endian to big endian or vice versa if the program is compiled on a little-endian architecture (such as Intel PCs). If the program has been compiled on a big-endian architecture, the value will be unchanged.

Use these macros to read data from and write data to a file that stores data in big-endian format.

## RTTI functions

wxWidgets uses its own RTTI ("run-time type identification") system which predates the current standard C++ RTTI and so is kept for backwards compatibility reasons but also because it allows some things which the standard RTTI doesn't directly support (such as creating a class from its name).

The standard C++ RTTI can be used in the user code without any problems and in general you shouldn't need to use the functions and the macros in this section unless you are thinking of modifying or adding any wxWidgets classes.

**See also**

*RTTI overview* (p. 2044)

## CLASSINFO

**wxClassInfo \* CLASSINFO(className)**

Returns a pointer to the `wxClassInfo` object associated with this class.

**Include files**

<wx/object.h>

**DECLARE\_ABSTRACT\_CLASS**

**DECLARE\_ABSTRACT\_CLASS**(className)

Used inside a class declaration to declare that the class should be made known to the class hierarchy, but objects of this class cannot be created dynamically. The same as `DECLARE_CLASS`.

Example:

```
class wxCommand: public wxObject
{
    DECLARE_ABSTRACT_CLASS(wxCommand)

    private:
        ...
    public:
        ...
};
```

**Include files**

<wx/object.h>

**DECLARE\_APP**

**DECLARE\_APP**(className)

This is used in headers to create a forward declaration of the `wxGetApp` (p. 1911) function implemented by `IMPLEMENT_APP` (p. 1967). It creates the declaration `className& wxGetApp(void)`.

Example:

```
DECLARE_APP(MyApp)
```

**Include files**

<wx/app.h>

**DECLARE\_CLASS**

**DECLARE\_CLASS**(className)

Used inside a class declaration to declare that the class should be made known to the class hierarchy, but objects of this class cannot be created dynamically. The same as `DECLARE_ABSTRACT_CLASS`.

**Include files**

<wx/object.h>

**DECLARE\_DYNAMIC\_CLASS**

**DECLARE\_DYNAMIC\_CLASS**(className)

Used inside a class declaration to make the class known to wxWidgets RTTI system and also declare that the objects of this class should be dynamically creatable from run-time type information. Notice that this implies that the class should have a default constructor, if this is not the case consider using *DECLARE\_CLASS* (p. 1965).

Example:

```
class wxFrame: public wxWindow
{
    DECLARE_DYNAMIC_CLASS(wxFrame)

private:
    const wxString& frameTitle;
public:
    ...
};
```

**Include files**

<wx/object.h>

**IMPLEMENT\_ABSTRACT\_CLASS**

**IMPLEMENT\_ABSTRACT\_CLASS**(className, baseClassName)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information. The same as *IMPLEMENT\_CLASS*.

Example:

```
IMPLEMENT_ABSTRACT_CLASS(wxCommand, wxObject)

wxCommand::wxCommand(void)
{
    ...
}
```

**Include files**

<wx/object.h>

**IMPLEMENT\_ABSTRACT\_CLASS2**

**IMPLEMENT\_ABSTRACT\_CLASS2**(className, baseClassName1, baseClassName2)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information and two base classes. The same as `IMPLEMENT_CLASS2`.

**Include files**

<wx/object.h>

**IMPLEMENT\_APP**

**IMPLEMENT\_APP**(className)

This is used in the application class implementation file to make the application class known to wxWidgets for dynamic construction. You use this instead of

Old form:

```
MyApp myApp;
```

New form:

```
IMPLEMENT_APP(MyApp)
```

See also *DECLARE\_APP* (p. 1965).

**Include files**

<wx/app.h>

**IMPLEMENT\_CLASS**

**IMPLEMENT\_CLASS**(className, baseClassName)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information. The same as `IMPLEMENT_ABSTRACT_CLASS`.

**Include files**

<wx/object.h>

**IMPLEMENT\_CLASS2**

**IMPLEMENT\_CLASS2**(className, baseClassName1, baseClassName2)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information and two base classes. The same as `IMPLEMENT_ABSTRACT_CLASS2`.

**Include files**

<wx/object.h>

**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(wxFFrame, wxWindow)

wxFFrame::wxFFrame(void)
{
    ...
}
```

**Include files**

<wx/object.h>

**IMPLEMENT\_DYNAMIC\_CLASS2****IMPLEMENT\_DYNAMIC\_CLASS2(className, baseClassName1, baseClassName2)**

Used in a C++ implementation file to complete the declaration of a class that has run-time type information, and whose instances can be created dynamically. Use this for classes derived from two base classes.

**Include files**

<wx/object.h>

**wxConstCast****classname \* wxConstCast(ptr, classname)**

This macro expands into `const_cast<classname *>(ptr)` if the compiler supports `const_cast` or into an old, C-style cast, otherwise.

**See also**

*wx\_const\_cast* (p. 1970)  
*wxDynamicCast* (p. 1969)  
*wxStaticCast* (p. 1970)

**::wxCreateDynamicObject****wxObject \* wxCreateDynamicObject(const wxString& className)**

Creates and returns an object of the given class, if the class has been registered with the dynamic class system using `DECLARE...` and `IMPLEMENT...` macros.

**WXDEBUG\_NEW**



**WXDEBUG\_NEW(arg)**

This is defined in debug mode to be call the redefined new operator with filename and line number arguments. The definition is:

```
#define WXDEBUG_NEW new(__FILE__, __LINE__)
```

In non-debug mode, this is defined as the normal new operator.

**Include files**

<wx/object.h>

**wxDynamicCast**

**classname \* wxDynamicCast(ptr, classname)**

This macro returns the pointer *ptr* cast to the type *classname* \* if the pointer is of this type (the check is done during the run-time) or `NULL` otherwise. Usage of this macro is preferred over obsoleted `wxObject::IsKindOf()` function.

The *ptr* argument may be `NULL`, in which case `NULL` will be returned.

Example:

```
wxWindow *win = wxWindow::FindFocus();
wxTextCtrl *text = wxDynamicCast(win, wxTextCtrl);
if ( text )
{
    // a text control has the focus...
}
else
{
    // no window has the focus or it is not a text control
}
```

**See also**

*RTTI overview* (p. 2044)

*wxDynamicCastThis* (p. 1969)

*wxConstCast* (p. 1968)

*wxStaticCast* (p. 1970)

**wxDynamicCastThis**

**classname \* wxDynamicCastThis(classname)**

This macro is equivalent to `wxDynamicCast(this, classname)` but the latter provokes spurious compilation warnings from some compilers (because it tests whether *this* pointer is non-`NULL` which is always true), so this macro should be used to avoid them.

**See also**

*wxDynamicCast* (p. 1969)

### **wxStaticCast**

**classname \* wxStaticCast**(ptr, classname)

This macro checks that the cast is valid in debug mode (an assert failure will result if `wxDynamicCast(ptr, classname) == NULL`) and then returns the result of executing an equivalent of `static_cast<classname *>(ptr)`.

#### **See also**

*wx\_static\_cast* (p. 1970)  
*wxDynamicCast* (p. 1969)  
*wxConstCast* (p. 1968)

### **wx\_const\_cast**

**T wx\_const\_cast**(T, x)

Same as `const_cast<T>(x)` if the compiler supports const cast or `(T)x` for old compilers. Unlike *wxConstCast* (p. 1968), the cast is to the type *T* and not to *T \** and also the order of arguments is the same as for the standard cast.

#### **See also**

*wx\_reinterpret\_cast* (p. 1970),  
*wx\_static\_cast* (p. 1970)

### **wx\_reinterpret\_cast**

**T wx\_reinterpret\_cast**(T, x)

Same as `reinterpret_cast<T>(x)` if the compiler supports reinterpret cast or `(T)x` for old compilers.

#### **See also**

*wx\_const\_cast* (p. 1970),  
*wx\_static\_cast* (p. 1970)

### **wx\_static\_cast**

**T wx\_static\_cast**(T, x)

Same as `static_cast<T>(x)` if the compiler supports static cast or `(T)x` for old compilers. Unlike *wxStaticCast* (p. 1970), there are no checks being done and the meaning of the macro arguments is exactly the same as for the standard static cast, i.e. *T* is the full type name and star is not appended to it.

#### **See also**

*wx\_const\_cast* (p. 1970),  
*wx\_reinterpret\_cast* (p. 1970),  
*wx\_truncate\_cast* (p. 1971)

### **wx\_truncate\_cast**

**T wx\_truncate\_cast**(T, x)

This case doesn't correspond to any standard cast but exists solely to make casts which possibly result in a truncation of an integer value more readable.

#### **See also**

*wx\_static\_cast* (p. 1970)

## **Log functions**

These functions provide a variety of logging functions: see *Log classes overview* (p. 2069) for further information. The functions use (implicitly) the currently active log target, so their descriptions here may not apply if the log target is not the standard one (installed by wxWidgets in the beginning of the program).

#### **Include files**

<wx/log.h>

### **::wxDebugMsg**

**void wxDebugMsg**(const wxString& *fmt*, ...)

**NB:** This function is now obsolete, replaced by *Log functions* (p. 1971) and *wxLogDebug* (p. 1973) in particular.

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.

**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.

#### **Include files**

<wx/utils.h>

### **::wxError**

**void wxError**(const wxString& *msg*, const wxString& *title* = "wxWidgets Internal Error")

**NB:** This function is now obsolete, please use *wxLogError* (p. 1972) instead.

Displays *msg* and continues. This writes to standard error under Unix, and pops up a message box under Windows. Used for internal wxWidgets errors. See also *wxFatalError* (p. 1972).

**Include files**

<wx/utils.h>

**::wxFatalError**

**void wxFatalError(const wxString& msg, const wxString& title = "wxWidgets Fatal Error")**

**NB:** This function is now obsolete, please use *wxLogFatalError* (p. 1972) instead.

Displays *msg* and exits. This writes to standard error under Unix, and pops up a message box under Windows. Used for fatal internal wxWidgets errors. See also *wxError* (p. 1971).

**Include files**

<wx/utils.h>

**::wxLogError**

**void wxLogError(const char \*formatString, ...)**

**void wxVLogError(const char \*formatString, va\_list argPtr)**

The functions to use for error messages, i.e. the messages that must be shown to the user. The default processing is to pop up a message box to inform the user about it.

**::wxLogFatalError**

**void wxLogFatalError(const char \*formatString, ...)**

**void wxVLogFatalError(const char \*formatString, va\_list argPtr)**

Like *wxLogError* (p. 1972), but also terminates the program with the exit code 3. Using *abort()* standard function also terminates the program with this exit code.

**::wxLogWarning**

**void wxLogWarning(const char \*formatString, ...)**

**void wxVLogWarning(const char \*formatString, va\_list argPtr)**

For warnings - they are also normally shown to the user, but don't interrupt the program work.

**::wxLogMessage**

**void wxLogMessage(const char \*formatString, ...)**

**void wxVLogMessage(const char \*formatString, va\_list argPtr)**

For all normal, informational messages. They also appear in a message box by default (but it can be changed).

### **::wxLogVerbose**

**void wxLogVerbose(const char \*formatString, ...)**

**void wxVLogVerbose(const char \*formatString, va\_list argPtr)**

For verbose output. Normally, it is suppressed, but might be activated if the user wishes to know more details about the program progress (another, but possibly confusing name for the same function is **wxLogInfo**).

### **::wxLogStatus**

**void wxLogStatus(wxFrame \*frame, const char \*formatString, ...)**

**void wxVLogStatus(wxFrame \*frame, const char \*formatString, va\_list argPtr)**

**void wxLogStatus(const char \*formatString, ...)**

**void wxVLogStatus(const char \*formatString, va\_list argPtr)**

Messages logged by these functions will appear in the statusbar of the *frame* or of the top level application window by default (i.e. when using the second version of the functions).

If the target frame doesn't have a statusbar, the message will be lost.

### **::wxLogSysError**

**void wxLogSysError(const char \*formatString, ...)**

**void wxVLogSysError(const char \*formatString, va\_list argPtr)**

Mostly used by wxWidgets itself, but might be handy for logging errors after system call (API function) failure. It logs the specified message text as well as the last system error code (*errno* or *::GetLastError()* depending on the platform) and the corresponding error message. The second form of this function takes the error code explicitly as the first argument.

### **See also**

*wxSysErrorCode* (p. 1975), *wxSysErrorMsg* (p. 1976)

### **::wxLogDebug**

**void wxLogDebug(const char \*formatString, ...)**

**void wxVLogDebug(const char \*formatString, va\_list argPtr)**

The right functions for debug output. They only do something in debug mode (when the preprocessor symbol `__WXDEBUG__` is defined) and expand to nothing in release mode (otherwise).

### **::wxLogTrace**

**void wxLogTrace(const char \*formatString, ...)**

**void wxVLogTrace(const char \*formatString, va\_list argPtr)**

**void wxLogTrace(const char \*mask, const char \*formatString, ...)**

**void wxVLogTrace(const char \*mask, const char \*formatString, va\_list argPtr)**

**void wxLogTrace(wxTraceMask mask, const char \*formatString, ...)**

**void wxVLogTrace(wxTraceMask mask, const char \*formatString, va\_list argPtr)**

As **wxLogDebug**, trace functions only do something in debug build and expand to nothing in the release one. The reason for making it a separate function from it is that usually there are a lot of trace messages, so it might make sense to separate them from other debug messages.

The trace messages also usually can be separated into different categories and the second and third versions of this function only log the message if the *mask* which it has is currently enabled in *wxLog* (p. 1018). This allows to selectively trace only some operations and not others by changing the value of the trace mask (possible during the run-time).

For the second function (taking a string mask), the message is logged only if the mask has been previously enabled by the call to *AddTraceMask* (p. 1021) or by setting *WXTRACE environment variable* (p. 2198). The predefined string trace masks used by *wxWidgets* are:

- `wxTRACE_MemAlloc`: trace memory allocation (new/delete)
- `wxTRACE_Messages`: trace window messages/X callbacks
- `wxTRACE_ResAlloc`: trace GDI resource allocation
- `wxTRACE_RefCount`: trace various ref counting operations
- `wxTRACE_OleCalls`: trace OLE method calls (Win32 only)

**Caveats:** since both the mask and the format string are strings, this might lead to function signature confusion in some cases: if you intend to call the format string only version of *wxLogTrace*, then add a `%s` format string parameter and then supply a second string parameter for that `%s`, the string mask version of *wxLogTrace* will erroneously get called instead, since you are supplying two string parameters to the function. In this case you'll unfortunately have to avoid having two leading string parameters, e.g. by adding a bogus integer (with its `%d` format string).

The third version of the function only logs the message if all the bits corresponding to the *mask* are set in the `wxLog` trace mask which can be set by `SetTraceMask` (p. 1025). This version is less flexible than the previous one because it doesn't allow defining the user trace masks easily - this is why it is deprecated in favour of using string trace masks.

- `wxTraceMemAlloc`: trace memory allocation (new/delete)
- `wxTraceMessages`: trace window messages/X callbacks
- `wxTraceResAlloc`: trace GDI resource allocation
- `wxTraceRefCount`: trace various ref counting operations
- `wxTraceOleCalls`: trace OLE method calls (Win32 only)

### **::wxSafeShowMessage**

**void wxSafeShowMessage(const wxString& title, const wxString& text)**

This function shows a message to the user in a safe way and should be safe to call even before the application has been initialized or if it is currently in some other strange state (for example, about to crash). Under Windows this function shows a message box using a native dialog instead of `wxMessageBox` (p. 1942) (which might be unsafe to call), elsewhere it simply prints the message to the standard output using the title as prefix.

#### **Parameters**

*title*

The title of the message box shown to the user or the prefix of the message string

*text*

The text to show to the user

#### **See also**

`wxLogFatalError` (p. 1972)

#### **Include files**

<wx/log.h>

### **::wxSysErrorCode**

**unsigned long wxSysErrorCode()**

Returns the error code from the last system call. This function uses `errno` on Unix platforms and `GetLastError` under Win32.

#### **See also**

`wxSysErrorMsg` (p. 1976), `wxLogSysError` (p. 1973)

**::wxSysErrorMsg****const wxChar \* wxSysErrorMsg(unsigned long *errCode* = 0)**

Returns the error message corresponding to the given system error code. If *errCode* is 0 (default), the last error code (as returned by `wxSysErrorCode` (p. 1975)) is used.

**See also**

`wxSysErrorCode` (p. 1975), `wxLogSysError` (p. 1973)

**WXTRACE****Include files**

<wx/object.h>

**WXTRACE**(formatString, ...)

**NB:** This macro is now obsolete, replaced by *Log functions* (p. 1971).

Calls `wxTrace` with printf-style variable argument syntax. Output is directed to the current output stream (see `wxDebugContext` (p. 2074)).

**Include files**

<wx/memory.h>

**WXTRACELEVEL**

**WXTRACELEVEL**(level, formatString, ...)

**NB:** This function is now obsolete, replaced by *Log functions* (p. 1971).

Calls `wxTraceLevel` with printf-style variable argument syntax. Output is directed to the current output stream (see `wxDebugContext` (p. 2074)). 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.

**Include files**

<wx/memory.h>

**::wxTrace**

**void wxTrace(const wxString& *fmt*, ...)**

**NB:** This function is now obsolete, replaced by *Log functions* (p. 1971).

Takes printf-style variable argument syntax. Output is directed to the current output stream (see `wxDebugContext` (p. 2074)).

**Include files**



<wx/memory.h>

### **::wxTraceLevel**

**void wxTraceLevel(int level, const wxString& fmt, ...)**

**NB:** This function is now obsolete, replaced by *Log functions* (p. 1971).

Takes printf-style variable argument syntax. Output is directed to the current output stream (see *wxDebugContext* (p. 2074)). 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.

#### **Include files**

<wx/memory.h>

## **Time functions**

The functions in this section deal with getting the current time and starting/stopping the global timers. Please note that the timer functions are deprecated because they work with one global timer only and *wxTimer* (p. 1680) and/or *wxStopWatch* (p. 1543) classes should be used instead. For retrieving the current time, you may also use *wxDateTime::Now* (p. 359) or *wxDateTime::UNow* (p. 360) methods.

### **::wxGetElapsedTime**

**long wxGetElapsedTime(bool resetTimer = true)**

Gets the time in milliseconds since the last *::wxStartTimer* (p. 1979).

If *resetTimer* is true (the default), the timer is reset to zero by this call.

See also *wxTimer* (p. 1680).

#### **Include files**

<wx/timer.h>

### **::wxGetLocalTime**

**long wxGetLocalTime()**

Returns the number of seconds since local time 00:00:00 Jan 1st 1970.

#### **See also**

*wxDateTime::Now* (p. 359)

#### **Include files**

<wx/timer.h>

**::wxGetLocalTimeMillis****wxLongLong wxGetLocalTimeMillis()**

Returns the number of milliseconds since local time 00:00:00 Jan 1st 1970.

**See also**

*wxDateTime::Now* (p. 359),  
*wxLongLong* (p. 1032)

**Include files**

<wx/timer.h>

**::wxGetUTCTime****long wxGetUTCTime()**

Returns the number of seconds since GMT 00:00:00 Jan 1st 1970.

**See also**

*wxDateTime::Now* (p. 359)

**Include files**

<wx/timer.h>

**::wxMicroSleep****void wxMicroSleep(unsigned long *microseconds*)**

Sleeps for the specified number of microseconds. The microsecond resolution may not, in fact, be available on all platforms (currently only Unix platforms with `nanosleep(2)` may provide it) in which case this is the same as *wxMilliSleep* (p. 1978)(*microseconds*/1000).

**Include files**

<wx/utils.h>

**::wxMilliSleep****void wxMilliSleep(unsigned long *milliseconds*)**

Sleeps for the specified number of milliseconds. Notice that usage of this function is encouraged instead of calling `usleep(3)` directly because the standard `usleep()` function is not MT safe.

**Include files**

<wx/utils.h>

**::wxNow****wxString wxNow()**

Returns a string representing the current date and time.

**Include files**

<wx/utils.h>

**::wxSleep****void wxSleep(int secs)**

Sleeps for the specified number of seconds.

**Include files**

<wx/utils.h>

**::wxStartTimer****void wxStartTimer()**

Starts a stopwatch; use `::wxGetElapsedTime` (p. 1977) to get the elapsed time.

See also `wxTimer` (p. 1680).

**Include files**

<wx/timer.h>

**::wxUsleep****void wxUsleep(unsigned long milliseconds)**

This function is deprecated because its name is misleading: notice that the argument is in milliseconds, not microseconds. Please use either `wxMilliSleep` (p. 1978) or `wxMicroSleep` (p. 1978) depending on the resolution you need.

## Debugging macros and functions

Useful macros and functions for error checking and defensive programming. `wxWidgets` defines three families of the assert-like macros: the `wxASSERT` and `wxFail` macros only do anything if `__WXDEBUG__` is defined (in other words, in the debug build) but disappear completely in the release build. On the other hand, the `wxCHECK` macros stay event in release builds but a check failure doesn't generate any user-visible effects then. Finally, the compile time assertions don't happen during the run-time but result in the compilation error messages if the condition they check fail.

**Include files**

<wx/debug.h>

### **::wxOnAssert**

**void wxOnAssert(const char \*fileName, int lineNumber, const char \*func, const char \*cond, const char \*msg = NULL)**

This function is called whenever one of debugging macros fails (i.e. condition is false in an assertion). It is only defined in the debug mode, in release builds the *wxCHECK* (p. 1982) failures don't result in anything.

To override the default behaviour in the debug builds which is to show the user a dialog asking whether he wants to abort the program, continue or continue ignoring any subsequent assert failures, you may override *wxApp::OnAssertFailure* (p. 49) which is called by this function if the global application object exists.

### **wxASSERT**

**wxASSERT(condition)**

Assert macro. An error message will be generated if the condition is false in debug mode, but nothing will be done in the release build.

Please note that the condition in *wxASSERT()* should have no side effects because it will not be executed in release mode at all.

#### **See also**

*wxASSERT\_MSG* (p. 1980),  
*wxCOMPILE\_TIME\_ASSERT* (p. 1981)

### **wxASSERT\_MIN\_BITSIZE**

**wxASSERT\_MIN\_BITSIZE(type, size)**

This macro results in a *compile time assertion failure* (p. 1981) if the size of the given type *type* is less than *size* bits.

You may use it like this, for example:

```
// we rely on the int being able to hold values up to 2^32
wxASSERT_MIN_BITSIZE(int, 32);

// can't work with the platforms using UTF-8 for wchar_t
wxASSERT_MIN_BITSIZE(wchar_t, 16);
```

### **wxASSERT\_MSG**

**wxASSERT\_MSG(condition, msg)**

Assert macro with message. An error message will be generated if the condition is false.

**See also**

`wxASSERT` (p. 1980),  
`wxCOMPILE_TIME_ASSERT` (p. 1981)

**wxCOMPILE\_TIME\_ASSERT**

**wxCOMPILE\_TIME\_ASSERT**(*condition*, *msg*)

Using `wxCOMPILE_TIME_ASSERT` results in a compilation error if the specified *condition* is false. The compiler error message should include the *msg* identifier - please note that it must be a valid C++ identifier and not a string unlike in the other cases.

This macro is mostly useful for testing the expressions involving the `sizeof` operator as they can't be tested by the preprocessor but it is sometimes desirable to test them at the compile time.

Note that this macro internally declares a struct whose name it tries to make unique by using the `__LINE__` in it but it may still not work if you use it on the same line in two different source files. In this case you may either change the line in which either of them appears on or use the `wxCOMPILE_TIME_ASSERT2` (p. 1981) macro.

Also note that Microsoft Visual C++ has a bug which results in compiler errors if you use this macro with 'Program Database For Edit And Continue' (`/ZI`) option, so you shouldn't use it ('Program Database' (`/Zi`) is ok though) for the code making use of this macro.

**See also**

`wxASSERT_MSG` (p. 1980),  
`wxASSERT_MIN_BITSIZE` (p. 1980)

**wxCOMPILE\_TIME\_ASSERT2**

**wxCOMPILE\_TIME\_ASSERT**(*condition*, *msg*, *name*)

This macro is identical to `wxCOMPILE_TIME_ASSERT2` (p. 1981) except that it allows you to specify a unique *name* for the struct internally defined by this macro to avoid getting the compilation errors described *above* (p. 1981).

**wxFAIL**

**wxFAIL**()

Will always generate an assert error if this code is reached (in debug mode).

See also: `wxFAIL_MSG` (p. 1981)

**wxFAIL\_MSG**

**wxFAIL\_MSG**(*msg*)

Will always generate an assert error with specified message if this code is reached (in debug mode).

This macro is useful for marking unreachable" code areas, for example it may be used in the "default:" branch of a switch statement if all possible cases are processed above.

#### **See also**

*wxFail* (p. 1981)

### **wxCHECK**

**wxCHECK**(*condition*, *retValue*)

Checks that the condition is true, returns with the given return value if not (FAILs in debug mode). This check is done even in release mode.

### **wxCHECK\_MSG**

**wxCHECK\_MSG**(*condition*, *retValue*, *msg*)

Checks that the condition is true, returns with the given return value if not (FAILs in debug mode). This check is done even in release mode.

This macro may be only used in non-void functions, see also *wxCHECK\_RET* (p. 1982).

### **wxCHECK\_RET**

**wxCHECK\_RET**(*condition*, *msg*)

Checks that the condition is true, and returns if not (FAILs with given error message in debug mode). This check is done even in release mode.

This macro should be used in void functions instead of *wxCHECK\_MSG* (p. 1982).

### **wxCHECK2**

**wxCHECK2**(*condition*, *operation*)

Checks that the condition is true and *wxFail* (p. 1981) and execute *operation* if it is not. This is a generalisation of *wxCHECK* (p. 1982) and may be used when something else than just returning from the function must be done when the *condition* is false.

This check is done even in release mode.

### **wxCHECK2\_MSG**

**wxCHECK2**(*condition*, *operation*, *msg*)

This is the same as *wxCHECK2* (p. 1982), but *wxFail\_MSG* (p. 1981) with the specified *msg* is called instead of *wxFail*() if the *condition* is false.

**::wxTrap****void wxTrap()**

In debug mode (when `__WXDEBUG__` is defined) this function generates a debugger exception meaning that the control is passed to the debugger if one is attached to the process. Otherwise the program just terminates abnormally.

In release mode this function does nothing.

**Include files**

<wx/debug.h>

**::wxIsDebuggerRunning****bool wxIsDebuggerRunning()**

Returns `true` if the program is running under debugger, `false` otherwise.

Please note that this function is currently only implemented for Win32 and Mac builds using CodeWarrior and always returns `false` elsewhere.

## Environment access functions

The functions in this section allow to access (get) or change value of environment variables in a portable way. They are currently implemented under Win32 and POSIX-like systems (Unix).

**Include files**

<wx/utils.h>

**wxGetenv****wxChar \* wxGetEnv(const wxString& var)**

This is a macro defined as `getenv( )` or its wide char version in Unicode mode.

Note that under Win32 it may not return correct value for the variables set with `wxSetEnv` (p. 1984), use `wxGetEnv` (p. 1983) function instead.

**wxGetEnv****bool wxGetEnv(const wxString& var, wxString \*value)**

Returns the current value of the environment variable `var` in `value`. `value` may be `NULL` if you just want to know if the variable exists and are not interested in its value.

Returns `true` if the variable exists, `false` otherwise.

**wxSetEnv****bool wxSetEnv(const wxString& var, const wxChar \*value)**

Sets the value of the environment variable *var* (adding it if necessary) to *value*.

Returns `true` on success.

**wxUnsetEnv****bool wxUnsetEnv(const wxString& var)**

Removes the variable *var* from the environment. `wxGetEnv` (p. 1983) will return `NULL` after the call to this function.

Returns `true` on success.



# Constants

This chapter describes the constants defined by wxWidgets.

## Preprocessor symbols defined by wxWidgets

These are preprocessor symbols used in the wxWidgets source, grouped by category (and sorted by alphabetical order inside each category). All of these macros except for the `wxUSE_XXX` variety is defined if the corresponding condition is true and undefined if it isn't, so they should be always tested using `ifdef` and not `if`.

### GUI system

<code>__WINDOWS__</code>	any Windows, you may also use <code>__WXMSW__</code>
<code>__WIN16__</code>	Win16 API (not supported since wxWidgets 2.6)
<code>__WIN32__</code>	Win32 API
<code>__WXBASE__</code>	Only wxBase, no GUI features (same as <code>wxUSE_GUI == 0</code> )
<code>__WXCOCOA__</code>	OS X using Cocoa API
<code>__WXWINCE__</code>	Windows CE
<code>__WXGTK__</code>	GTK+
<code>__WXGTK12__</code>	GTK+ 1.2 or higher
<code>__WXGTK20__</code>	GTK+ 2.0 or higher
<code>__WXMOTIF__</code>	Motif
<code>__WXMOTIF20__</code>	Motif 2.0 or higher
<code>__WXMAC__</code>	Mac OS all targets
<code>__WXMAC_CLASSIC__</code>	MacOS for Classic
<code>__WXMAC_CARBON__</code>	MacOS for Carbon CFM (running under Classic or OSX) or true OS X Mach-O Builds
<code>__WXMAC_OSX__</code>	MacOS X Carbon Mach-O Builds
<code>__WXMGL__</code>	SciTech Soft MGL ( <code>__WXUNIVERSAL__</code> will be also defined)
<code>__WXMSW__</code>	Any Windows

<code>__WXOSX__</code>	Any Mac OS X port (either Carbon or Cocoa)
<code>__WXPALMOS__</code>	PalmOS
<code>__WXPM__</code>	OS/2 native Presentation Manager
<code>__WXSTUBS__</code>	Stubbed version ('template' wxWin implementation)
<code>__WXXT__</code>	Xt; mutually exclusive with <code>WX_MOTIF</code> , not implemented in wxWidgets 2.x
<code>__WXX11__</code>	wxX11 ( <code>__WXUNIVERSAL__</code> will be also defined)
<code>__WXWINE__</code>	WINE (i.e. WIN32 on Unix)
<code>__WXUNIVERSAL__</code>	wxUniversal port, always defined in addition to one of the symbols above so this should be tested first.
<code>__X__</code>	any X11-based GUI toolkit except GTK+

There are two wxWidgets ports to Mac OS. One of them, wxMac, exists in two versions: Classic and Carbon. The Classic version is the only one to work on Mac OS version 8. The Carbon version may be built either as CFM or Mach-O (binary format, like ELF) and the former may run under OS 9 while the latter only runs under OS X. Finally, there is a new Cocoa port which can only be used under OS X. To summarize:

- If you want to test for all Mac platforms, classic and OS X, you should test both `__WXMAC__` and `__WXCOCOA__`.
- If you want to test for any GUI Mac port under OS X, use `__WXOSX__`.
- If you want to test for any port under Mac OS X, including, for example, wxGTK and also wxBase, use `__DARWIN__` (see below).

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

## Operating systems

<code>__APPLE__</code>	any Mac OS version
<code>__AIX__</code>	AIX
<code>__BSD__</code>	Any *BSD system
<code>__CYGWIN__</code>	Cygwin: Unix on Win32
<code>__DARWIN__</code>	Mac OS X using the BSD Unix C library (as opposed to using the Metrowerks MSL C/C++ library)

__DATA_GENERAL__	DG-UX
__DOS_GENERAL__	DOS (used with wxMGL only)
__FREEBSD__	FreeBSD
__HPUX__	HP-UX (Unix)
__GNU__	GNU Hurd
__LINUX__	Linux
__MACH__	Mach-O Architecture (Mac OS X only builds)
__OSF__	OSF/1
__PALMOS__	PalmOS
__SGI__	IRIX
__SOLARIS__	Solaris
__SUN__	Any Sun
__SUNOS__	Sun OS
__SVR4__	SystemV R4
__SYSV__	SystemV generic
__ULTRIX__	Ultrix
__UNIX__	any Unix
__UNIX_LIKE__	Unix, BeOS or VMS
__VMS__	VMS
__WINDOWS__	any Windows
__WINE__	Wine

### Hardware architectures (CPU)

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

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

### Hardware type

<code>__SMARTPHONE__</code>	Generic mobile devices with phone buttons and a small display
<code>__PDA__</code>	Personal digital assistant, usually with touch screen
<code>__HANDHELD__</code>	Small but powerful computer, usually with a keyboard
<code>__POCKETPC__</code>	Microsoft-powered PocketPC devices with touch-screen
<code>__WINCE_STANDARDSDK__</code>	Microsoft-powered Windows CE devices, for generic Windows CE applications
<code>__WINCE_NET__</code>	Microsoft-powered Windows CE .NET devices ( <code>__WIN32_WCE</code> is 400 or greater)
<code>WIN32_PLATFORM_WFSP</code>	Microsoft-powered smartphone

### Compilers

<code>__BORLANDC__</code>	Borland C++. The value of the macro corresponds to the compiler version: 500 is 5.0.
<code>__DJGPP__</code>	DJGPP
<code>__DIGITALMARS__</code>	Digital Mars
<code>__GNUG__</code>	Gnu C++ on any platform, see also <code>wxCHECK_GCC_VERSION</code> (p. 1908)
<code>__GNUWIN32__</code>	Gnu-Win32 compiler, see also <code>wxCHECK_W32API_VERSION</code> (p. 1909)
<code>__MINGW32__</code>	MinGW
<code>__MWERKS__</code>	CodeWarrior MetroWerks compiler
<code>__SUNCC__</code>	Sun CC
<code>__SYMANTECC__</code>	Symantec C++
<code>__VISAGECPP__</code>	IBM Visual Age (OS/2)
<code>__VISUALC__</code>	Microsoft Visual C++. The value of this macro corresponds to the compiler version: 1020 for 4.2 (the first supported version), 1100 for 5.0, 1200 for 6.0 and so on
<code>__XLC__</code>	AIX compiler
<code>__WATCOMC__</code>	Watcom C++. The value of this macro corresponds to the compiler version, 1100 is

11.0 and 1200 is OpenWatcom.

`_WIN32_WCE`

Windows CE version

## Feature tests

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

Currently the following symbols exist: `wxHAS_LARGE_FILES` Defined if *wxFile* (p. 591) supports files more than 4GB in size.

`wxHAS_LARGE_FFILES` Defined if *wxFFFile* (p. 584) supports files more than 4GB in size.

`wxHAS_POWER_EVENTS` Defined if *wxPowerEvent* (p. 1195) are ever generated on the current platform.

`wxHAS_RADIO_MENU_ITEMS` Defined if the current port supports *radio menu items* (p. 1079).

`wxHAS_RAW_KEY_CODES` Defined if *raw key codes* (p. 960) are supported.

`wxHAS_REGEX_ADVANCED` Defined if advanced syntax is available in *wxRegEx* (p. 1260).

`wxHAS_TASK_BAR_ICON` Defined if *wxTaskBarIcon* (p. 1607) is available on the current platform.

## Miscellaneous

`__WXWINDOWS__` always defined in *wxWidgets* applications, see also *wxCHECK\_VERSION* (p. 1909)

`__WXDEBUG__` defined in debug mode, undefined in release mode

`wxUSE_XXX` if defined as 1, feature XXX is active (the symbols of this form are always defined, use `#if` and not `#ifdef` to test for them)

`WX_PRECOMP` is defined if precompiled headers (PCH) are in use. In this case, `wx/wxprec.h` includes `wx/wx.h` which, in turn, includes a number of *wxWidgets* headers thus making it unnecessary to include them explicitly. However if this is not defined, you do need to include them and so the usual idiom which allows to support both cases

	is to first include <code>wx/wxprec.h</code> and then, inside <code>ifndef WX_PRECOMP</code> , individual headers you need.
<code>_UNICODE</code> and <code>UNICODE</code>	both are defined if <code>wxUSE_UNICODE</code> is set to 1
<code>wxUSE_GUI</code>	this particular feature test macro is defined to 1 when compiling or using the library with the GUI features activated, if it is defined as 0, only <code>wxBase</code> is available.
<code>wxUSE_BASE</code>	only used by <code>wxWidgets</code> internally (defined as 1 when building <code>wxBase</code> code, either as a standalone library or as part of the monolithic <code>wxWidgets</code> library, defined as 0 when building GUI library only)
<code>wxNO_RTTI</code>	is defined if the compiler RTTI support has been switched off
<code>wxNO_EXCEPTIONS</code>	is defined if the compiler support for C++ exceptions has been switched off
<code>wxNO_THREADS</code>	if this macro is defined, the compilation options don't include compiler flags needed for multithreaded code generation. This implies that <code>wxUSE_THREADS</code> is 0 and also that other (non-wx-based) threading packages cannot be used neither.

## Standard event identifiers

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

- when creating a new window you may specify `wxID_ANY` to let `wxWidgets` assign an unused identifier to it automatically
- when installing an event handler using either the event table macros or `wxEvtHandler::Connect` (p. 577), you may use it to indicate that you want to handle the events coming from any control, regardless of its identifier

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

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

```
wxID_LOWEST = 4999,
```

```
wxID_OPEN,
wxID_CLOSE,
wxID_NEW,
wxID_SAVE,
wxID_SAVEAS,
wxID_REVERT,
wxID_EXIT,
wxID_UNDO,
wxID_REDO,
wxID_HELP,
wxID_PRINT,
wxID_PRINT_SETUP,
wxID_PAGE_SETUP,
wxID_PREVIEW,
wxID_ABOUT,
wxID_HELP_CONTENTS,
wxID_HELP_INDEX,
wxID_HELP_SEARCH,
wxID_HELP_COMMANDS,
wxID_HELP_PROCEDURES,
wxID_HELP_CONTEXT,
wxID_CLOSE_ALL,

wxID_EDIT = 5030,
wxID_CUT,
wxID_COPY,
wxID_PASTE,
wxID_CLEAR,
wxID_FIND,
wxID_DUPLICATE,
wxID_SELECTALL,
wxID_DELETE,
wxID_REPLACE,
wxID_REPLACE_ALL,
wxID_PROPERTIES,

wxID_VIEW_DETAILS,
wxID_VIEW_LARGEICONS,
wxID_VIEW_SMALLICONS,
wxID_VIEW_LIST,
wxID_VIEW_SORTDATE,
wxID_VIEW_SORTNAME,
wxID_VIEW_SORTSIZE,
wxID_VIEW_SORTTYPE,

wxID_FILE = 5050,
wxID_FILE1,
wxID_FILE2,
wxID_FILE3,
wxID_FILE4,
wxID_FILE5,
wxID_FILE6,
wxID_FILE7,
wxID_FILE8,
wxID_FILE9,

// Standard button IDs
```

```
wxID_OK = 5100,
wxID_CANCEL,
wxID_APPLY,
wxID_YES,
wxID_NO,
wxID_STATIC,
wxID_FORWARD,
wxID_BACKWARD,
wxID_DEFAULT,
wxID_MORE,
wxID_SETUP,
wxID_RESET,
wxID_CONTEXT_HELP,
wxID_YESTOALL,
wxID_NOTOALL,
wxID_ABORT,
wxID_RETRY,
wxID_IGNORE,

wxID_UP,
wxID_DOWN,
wxID_HOME,
wxID_REFRESH,
wxID_STOP,
wxID_INDEX,

wxID_BOLD,
wxID_ITALIC,
wxID_JUSTIFY_CENTER,
wxID_JUSTIFY_FILL,
wxID_JUSTIFY_RIGHT,
wxID_JUSTIFY_LEFT,
wxID_UNDERLINE,
wxID_INDENT,
wxID_UNINDENT,
wxID_ZOOM_100,
wxID_ZOOM_FIT,
wxID_ZOOM_IN,
wxID_ZOOM_OUT,
wxID_UNDELETE,
wxID_REVERT_TO_SAVED,

// System menu IDs (used by wxUniv):
wxID_SYSTEM_MENU = 5200,
wxID_CLOSE_FRAME,
wxID_MOVE_FRAME,
wxID_RESIZE_FRAME,
wxID_MAXIMIZE_FRAME,
wxID_ICONIZE_FRAME,
wxID_RESTORE_FRAME,

// IDs used by generic file dialog (13 consecutive starting from
this value)
wxID_FILEDLGG = 5900,

wxID_HIGHEST = 5999
```

---



## Keycodes

### Include files

<wx/defs.h>

Keypresses are represented by an enumerated type, `wxKeyCode`. The possible values are the ASCII character codes, plus the following:

```
WXX_BACK      = 8
WXX_TAB       = 9
WXX_RETURN    = 13
WXX_ESCAPE    = 27
WXX_SPACE     = 32
WXX_DELETE    = 127

// These are by design not compatible with unicode characters.
// If you want to get a unicode character from a key event use
// wxKeyEvent::GetUnicodeKey instead.
WXX_START     = 300
WXX_LBUTTON
WXX_RBUTTON
WXX_CANCEL
WXX_MBUTTON
WXX_CLEAR
WXX_SHIFT
WXX_ALT
WXX_CONTROL
WXX_MENU
WXX_PAUSE
WXX_CAPITAL
WXX_END
WXX_HOME
WXX_LEFT
WXX_UP
WXX_RIGHT
WXX_DOWN
WXX_SELECT
WXX_PRINT
WXX_EXECUTE
WXX_SNAPSHOT
WXX_INSERT
WXX_HELP
WXX_NUMPAD0
WXX_NUMPAD1
WXX_NUMPAD2
WXX_NUMPAD3
WXX_NUMPAD4
WXX_NUMPAD5
WXX_NUMPAD6
WXX_NUMPAD7
WXX_NUMPAD8
WXX_NUMPAD9
WXX_MULTIPLY
```

```
WXX_ADD
WXX_SEPARATOR
WXX_SUBTRACT
WXX_DECIMAL
WXX_DIVIDE
WXX_F1
WXX_F2
WXX_F3
WXX_F4
WXX_F5
WXX_F6
WXX_F7
WXX_F8
WXX_F9
WXX_F10
WXX_F11
WXX_F12
WXX_F13
WXX_F14
WXX_F15
WXX_F16
WXX_F17
WXX_F18
WXX_F19
WXX_F20
WXX_F21
WXX_F22
WXX_F23
WXX_F24
WXX_NUMLOCK
WXX_SCROLL
WXX_PAGEUP,
WXX_PAGEDOWN,

WXX_NUMPAD_SPACE,
WXX_NUMPAD_TAB,
WXX_NUMPAD_ENTER,
WXX_NUMPAD_F1,
WXX_NUMPAD_F2,
WXX_NUMPAD_F3,
WXX_NUMPAD_F4,
WXX_NUMPAD_HOME,
WXX_NUMPAD_LEFT,
WXX_NUMPAD_UP,
WXX_NUMPAD_RIGHT,
WXX_NUMPAD_DOWN,
WXX_NUMPAD_PAGEUP,
WXX_NUMPAD_PAGEDOWN,
WXX_NUMPAD_END,
WXX_NUMPAD_BEGIN,
WXX_NUMPAD_INSERT,
WXX_NUMPAD_DELETE,
WXX_NUMPAD_EQUAL,
WXX_NUMPAD_MULTIPLY,
WXX_NUMPAD_ADD,
WXX_NUMPAD_SEPARATOR,
WXX_NUMPAD_SUBTRACT,
```

```
WXX_NUMPAD_DECIMAL,
WXX_NUMPAD_DIVIDE,

// the following key codes are only generated under Windows
currently
WXX_WINDOWS_LEFT,
WXX_WINDOWS_RIGHT,
WXX_WINDOWS_MENU,
WXX_COMMAND,

// Hardware-specific buttons
WXX_SPECIAL1 = 193,
WXX_SPECIAL2,
WXX_SPECIAL3,
WXX_SPECIAL4,
WXX_SPECIAL5,
WXX_SPECIAL6,
WXX_SPECIAL7,
WXX_SPECIAL8,
WXX_SPECIAL9,
WXX_SPECIAL10,
WXX_SPECIAL11,
WXX_SPECIAL12,
WXX_SPECIAL13,
WXX_SPECIAL14,
WXX_SPECIAL15,
WXX_SPECIAL16,
WXX_SPECIAL17,
WXX_SPECIAL18,
WXX_SPECIAL19,
WXX_SPECIAL20
```

## Key Modifiers

### Include files

<wx/defs.h>

The following key modifier constants are defined:

```
enum wxKeyModifier
{
    wxMOD_NONE          = 0x0000,
    wxMOD_ALT           = 0x0001,
    wxMOD_CONTROL       = 0x0002,
    wxMOD_ALTGR         = wxMOD_ALT | wxMOD_CONTROL,
    wxMOD_SHIFT         = 0x0004,
    wxMOD_META          = 0x0008,
#ifdef __WXMAC__ || defined(__WXCOCOA__)
    wxMOD_CMD           = wxMOD_META,
#else
    wxMOD_CMD           = wxMOD_CONTROL,
#endif
    wxMOD_ALL           = 0xffff
};
```

Notice that `wxMOD_CMD` should be used instead of `wxMOD_CONTROL` in portable code to account for the fact that although `CONTROL` modifier exists under Mac OS, it is not used for the same purpose as under Windows or Unix there while the special Mac-specific `COMMAND` modifier is used in exactly the same way.

## Language identifiers

The following `wxLanguage` constants may be used to specify the language in `wxLocale::Init` (p. 1017) and are returned by `wxLocale::GetSystemLanguage` (p. 1016):

<code>wxLANGUAGE_DEFAULT</code>	user's default language as obtained from the operating system
<code>wxLANGUAGE_UNKNOWN</code>	returned by <code>GetSystemLanguage</code> (p. 1016) if it fails to detect the default language
<code>wxLANGUAGE_USER_DEFINED</code>	user defined languages' integer identifiers should start from this
<code>wxLANGUAGE_ABKHAZIAN</code>	
<code>wxLANGUAGE_AFAR</code>	
<code>wxLANGUAGE_AFRIKAANS</code>	
<code>wxLANGUAGE_ALBANIAN</code>	
<code>wxLANGUAGE_AMHARIC</code>	
<code>wxLANGUAGE_ARABIC</code>	
<code>wxLANGUAGE_ARABIC_ALGERIA</code>	
<code>wxLANGUAGE_ARABIC_BAHRAIN</code>	
<code>wxLANGUAGE_ARABIC_EGYPT</code>	
<code>wxLANGUAGE_ARABIC_IRAQ</code>	
<code>wxLANGUAGE_ARABIC_JORDAN</code>	
<code>wxLANGUAGE_ARABIC_KUWAIT</code>	
<code>wxLANGUAGE_ARABIC_LEBANON</code>	
<code>wxLANGUAGE_ARABIC_LIBYA</code>	
<code>wxLANGUAGE_ARABIC_MOROCCO</code>	
<code>wxLANGUAGE_ARABIC_OMAN</code>	

wxLANGUAGE\_ARABIC\_QATAR  
wxLANGUAGE\_ARABIC\_SAUDI\_ARABIA  
wxLANGUAGE\_ARABIC\_SUDAN  
wxLANGUAGE\_ARABIC\_SYRIA  
wxLANGUAGE\_ARABIC\_TUNISIA  
wxLANGUAGE\_ARABIC\_UAE  
wxLANGUAGE\_ARABIC\_YEMEN  
wxLANGUAGE\_ARMENIAN  
wxLANGUAGE\_ASSAMESE  
wxLANGUAGE\_AYMARA  
wxLANGUAGE\_AZERI  
wxLANGUAGE\_AZERI\_CYRILLIC  
wxLANGUAGE\_AZERI\_LATIN  
wxLANGUAGE\_BASHKIR  
wxLANGUAGE\_BASQUE  
wxLANGUAGE\_BELARUSIAN  
wxLANGUAGE\_BENGALI  
wxLANGUAGE\_BHUTANI  
wxLANGUAGE\_BIHARI  
wxLANGUAGE\_BISLAMA  
wxLANGUAGE\_BRETON  
wxLANGUAGE\_BULGARIAN  
wxLANGUAGE\_BURMESE  
wxLANGUAGE\_CAMBODIAN  
wxLANGUAGE\_CATALAN  
wxLANGUAGE\_CHINESE  
wxLANGUAGE\_CHINESE\_SIMPLIFIED  
wxLANGUAGE\_CHINESE\_TRADITIONAL  
wxLANGUAGE\_CHINESE\_HONGKONG

wxLANGUAGE\_CHINESE\_MACAU  
wxLANGUAGE\_CHINESE\_SINGAPORE  
wxLANGUAGE\_CHINESE\_TAIWAN  
wxLANGUAGE\_CORSICAN  
wxLANGUAGE\_CROATIAN  
wxLANGUAGE\_CZECH  
wxLANGUAGE\_DANISH  
wxLANGUAGE\_DUTCH  
wxLANGUAGE\_DUTCH\_BELGIAN  
wxLANGUAGE\_ENGLISH  
wxLANGUAGE\_ENGLISH\_UK  
wxLANGUAGE\_ENGLISH\_US  
wxLANGUAGE\_ENGLISH\_AUSTRALIA  
wxLANGUAGE\_ENGLISH\_BELIZE  
wxLANGUAGE\_ENGLISH\_BOTSWANA  
wxLANGUAGE\_ENGLISH\_CANADA  
wxLANGUAGE\_ENGLISH\_CARIBBEAN  
wxLANGUAGE\_ENGLISH\_DENMARK  
wxLANGUAGE\_ENGLISH\_EIRE  
wxLANGUAGE\_ENGLISH\_JAMAICA  
wxLANGUAGE\_ENGLISH\_NEW\_ZEALAND  
wxLANGUAGE\_ENGLISH\_PHILIPPINES  
wxLANGUAGE\_ENGLISH\_SOUTH\_AFRICA  
wxLANGUAGE\_ENGLISH\_TRINIDAD  
wxLANGUAGE\_ENGLISH\_ZIMBABWE  
wxLANGUAGE\_ESPERANTO  
wxLANGUAGE\_ESTONIAN  
wxLANGUAGE\_FAEROESE  
wxLANGUAGE\_FARSI

wxLANGUAGE\_FIJI  
wxLANGUAGE\_FINNISH  
wxLANGUAGE\_FRENCH  
wxLANGUAGE\_FRENCH\_BELGIAN  
wxLANGUAGE\_FRENCH\_CANADIAN  
wxLANGUAGE\_FRENCH\_LUXEMBOURG  
wxLANGUAGE\_FRENCH\_MONACO  
wxLANGUAGE\_FRENCH\_SWISS  
wxLANGUAGE\_FRISIAN  
wxLANGUAGE\_GALICIAN  
wxLANGUAGE\_GEORGIAN  
wxLANGUAGE\_GERMAN  
wxLANGUAGE\_GERMAN\_AUSTRIAN  
wxLANGUAGE\_GERMAN\_BELGIUM  
wxLANGUAGE\_GERMAN\_LIECHTENSTEIN  
wxLANGUAGE\_GERMAN\_LUXEMBOURG  
wxLANGUAGE\_GERMAN\_SWISS  
wxLANGUAGE\_GREEK  
wxLANGUAGE\_GREENLANDIC  
wxLANGUAGE\_GUARANI  
wxLANGUAGE\_GUJARATI  
wxLANGUAGE\_HAUSA  
wxLANGUAGE\_HEBREW  
wxLANGUAGE\_HINDI  
wxLANGUAGE\_HUNGARIAN  
wxLANGUAGE\_ICELANDIC  
wxLANGUAGE\_INDONESIAN  
wxLANGUAGE\_INTERLINGUA  
wxLANGUAGE\_INTERLINGUE

wxLANGUAGE\_INUKTITUT  
wxLANGUAGE\_INUPIAK  
wxLANGUAGE\_IRISH  
wxLANGUAGE\_ITALIAN  
wxLANGUAGE\_ITALIAN\_SWISS  
wxLANGUAGE\_JAPANESE  
wxLANGUAGE\_JAVANESE  
wxLANGUAGE\_KANNADA  
wxLANGUAGE\_KASHMIRI  
wxLANGUAGE\_KASHMIRI\_INDIA  
wxLANGUAGE\_KAZAKH  
wxLANGUAGE\_KERNEWEK  
wxLANGUAGE\_KINYARWANDA  
wxLANGUAGE\_KIRGHIZ  
wxLANGUAGE\_KIRUNDI  
wxLANGUAGE\_KONKANI  
wxLANGUAGE\_KOREAN  
wxLANGUAGE\_KURDISH  
wxLANGUAGE\_LAOTHIAN  
wxLANGUAGE\_LATIN  
wxLANGUAGE\_LATVIAN  
wxLANGUAGE\_LINGALA  
wxLANGUAGE\_LITHUANIAN  
wxLANGUAGE\_MACEDONIAN  
wxLANGUAGE\_MALAGASY  
wxLANGUAGE\_MALAY  
wxLANGUAGE\_MALAYALAM  
wxLANGUAGE\_MALAY\_BRUNEI\_DARUSSALAM  
wxLANGUAGE\_MALAY\_MALAYSIA



wxLANGUAGE\_MALTESE  
wxLANGUAGE\_MANIPURI  
wxLANGUAGE\_MAORI  
wxLANGUAGE\_MARATHI  
wxLANGUAGE\_MOLDAVIAN  
wxLANGUAGE\_MONGOLIAN  
wxLANGUAGE\_NAURU  
wxLANGUAGE\_NEPALI  
wxLANGUAGE\_NEPALI\_INDIA  
wxLANGUAGE\_NORWEGIAN\_BOKMAL  
wxLANGUAGE\_NORWEGIAN\_NYNORSK  
wxLANGUAGE\_OCCITAN  
wxLANGUAGE\_ORIYA  
wxLANGUAGE\_OROMO  
wxLANGUAGE\_PASHTO  
wxLANGUAGE\_POLISH  
wxLANGUAGE\_PORTUGUESE  
wxLANGUAGE\_PORTUGUESE\_BRAZILIAN  
wxLANGUAGE\_PUNJABI  
wxLANGUAGE\_QUECHUA  
wxLANGUAGE\_RHAETO\_ROMANCE  
wxLANGUAGE\_ROMANIAN  
wxLANGUAGE\_RUSSIAN  
wxLANGUAGE\_RUSSIAN\_UKRAINE  
wxLANGUAGE\_SAMI  
wxLANGUAGE\_SAMOAN  
wxLANGUAGE\_SANGHO  
wxLANGUAGE\_SANSKRIT  
wxLANGUAGE\_SCOTS\_GAELIC

wxLANGUAGE\_SERBIAN  
wxLANGUAGE\_SERBIAN\_CYRILLIC  
wxLANGUAGE\_SERBIAN\_LATIN  
wxLANGUAGE\_SERBO\_CROATIAN  
wxLANGUAGE\_SESOTHO  
wxLANGUAGE\_SETSWANA  
wxLANGUAGE\_SHONA  
wxLANGUAGE\_SINDHI  
wxLANGUAGE\_SINHALESE  
wxLANGUAGE\_SISWATI  
wxLANGUAGE\_SLOVAK  
wxLANGUAGE\_SLOVENIAN  
wxLANGUAGE\_SOMALI  
wxLANGUAGE\_SPANISH  
wxLANGUAGE\_SPANISH\_ARGENTINA  
wxLANGUAGE\_SPANISH\_BOLIVIA  
wxLANGUAGE\_SPANISH\_CHILE  
wxLANGUAGE\_SPANISH\_COLOMBIA  
wxLANGUAGE\_SPANISH\_COSTA\_RICA  
wxLANGUAGE\_SPANISH\_DOMINICAN\_REPUBLIC  
wxLANGUAGE\_SPANISH\_ECUADOR  
wxLANGUAGE\_SPANISH\_EL\_SALVADOR  
wxLANGUAGE\_SPANISH\_GUATEMALA  
wxLANGUAGE\_SPANISH\_HONDURAS  
wxLANGUAGE\_SPANISH\_MEXICAN  
wxLANGUAGE\_SPANISH\_MODERN  
wxLANGUAGE\_SPANISH\_NICARAGUA  
wxLANGUAGE\_SPANISH\_PANAMA  
wxLANGUAGE\_SPANISH\_PARAGUAY

wxLANGUAGE\_SPANISH\_PERU  
wxLANGUAGE\_SPANISH\_PUERTO\_RICO  
wxLANGUAGE\_SPANISH\_URUGUAY  
wxLANGUAGE\_SPANISH\_US  
wxLANGUAGE\_SPANISH\_VENEZUELA  
wxLANGUAGE\_SUNDANESE  
wxLANGUAGE\_SWAHILI  
wxLANGUAGE\_SWEDISH  
wxLANGUAGE\_SWEDISH\_FINLAND  
wxLANGUAGE\_TAGALOG  
wxLANGUAGE\_TAJIK  
wxLANGUAGE\_TAMIL  
wxLANGUAGE\_TATAR  
wxLANGUAGE\_TELUGU  
wxLANGUAGE\_THAI  
wxLANGUAGE\_TIBETAN  
wxLANGUAGE\_TIGRINYA  
wxLANGUAGE\_TONGA  
wxLANGUAGE\_TSONGA  
wxLANGUAGE\_TURKISH  
wxLANGUAGE\_TURKMEN  
wxLANGUAGE\_TWI  
wxLANGUAGE\_UIGHUR  
wxLANGUAGE\_UKRAINIAN  
wxLANGUAGE\_URDU  
wxLANGUAGE\_URDU\_INDIA  
wxLANGUAGE\_URDU\_PAKISTAN  
wxLANGUAGE\_UZBEK  
wxLANGUAGE\_UZBEK\_CYRILLIC

wxLANGUAGE\_UZBEK\_LATIN  
wxLANGUAGE\_VALENCIAN  
wxLANGUAGE\_VIETNAMESE  
wxLANGUAGE\_VOLAPUK  
wxLANGUAGE\_WELSH  
wxLANGUAGE\_WOLOF  
wxLANGUAGE\_XHOSA  
wxLANGUAGE\_YIDDISH  
wxLANGUAGE\_YORUBA  
wxLANGUAGE\_ZHUANG  
wxLANGUAGE\_ZULU

## Stock items

Window IDs for which stock buttons and menu items are created (see *wxButton constructor* (p. 165) and *wxMenuItem constructor* (p. 1099)):

Stock ID	Stock label
wxID_ABOUT	"&About"
wxID_ADD	"Add"
wxID_APPLY	"&Apply"
wxID_BOLD	"&Bold"
wxID_CANCEL	"&Cancel"
wxID_CLEAR	"&Clear"
wxID_CLOSE	"&Close"
wxID_COPY	"&Copy"
wxID_CUT	"Cu&t"
wxID_DELETE	"&Delete"
wxID_EDIT	"&Edit"
wxID_FIND	"&Find"
wxID_FILE	"&File"

wxID_REPLACE	"Find and rep&lac"
wxID_BACKWARD	"&Back"
wxID_DOWN	"&Down"
wxID_FORWARD	"&Forward"
wxID_UP	"&Up"
wxID_HELP	"&Help"
wxID_HOME	"&Home"
wxID_INDENT	"Indent"
wxID_INDEX	"&Index"
wxID_ITALIC	"&Italic"
wxID_JUSTIFY_CENTER	"Centered"
wxID_JUSTIFY_FILL	"Justified"
wxID_JUSTIFY_LEFT	"Align Left"
wxID_JUSTIFY_RIGHT	"Align Right"
wxID_NEW	"&New"
wxID_NO	"&No"
wxID_OK	"&OK"
wxID_OPEN	"&Open"
wxID_PASTE	"&Paste"
wxID_PREFERENCES	"&Preferences"
wxID_PRINT	"&Print"
wxID_PREVIEW	"Print previe&w"
wxID_PROPERTIES	"&Properties"
wxID_EXIT	"&Quit"
wxID_REDO	"&Redo"
wxID_REFRESH	"Refresh"
wxID_REMOVE	"Remove"
wxID_REVERT_TO_SAVED	"Revert to Saved"
wxID_SAVE	"&Save"

wxID_SAVEAS	"Save &As..."
wxID_SELECTALL	"Select all"
wxID_STOP	"&Stop"
wxID_UNDELETE	"Undelete"
wxID_UNDERLINE	"&Underline"
wxID_UNDO	"&Undo"
wxID_UNINDENT	"&Unindent"
wxID_YES	"&Yes"
wxID_ZOOM_100	"&Actual Size"
wxID_ZOOM_FIT	"Zoom to &Fit"
wxID_ZOOM_IN	"Zoom &In"
wxID_ZOOM_OUT	"Zoom &Out"

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

## Classes by category

A classification of wxWidgets classes by category.

### Managed windows

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

<i>wxTopLevelWindow</i> (p. 1713)	Any top level window, dialog or frame
<i>wxDialog</i> (p. 496)	Dialog box
<i>wxFrame</i> (p. 682)	Normal frame
<i>wxMDIChildFrame</i> (p. 1047)	MDI child frame
<i>wxMDIParentFrame</i> (p. 1051)	MDI parent frame
<i>wxMiniFrame</i> (p. 1113)	A frame with a small title bar
<i>wxPropertySheetDialog</i> (p. 1232)	Property sheet dialog
<i>wxSplashScreen</i> (p. 1504)	Splash screen class
<i>wxTipWindow</i> (p. 1691)	Shows text in a small window
<i>wxWizard</i> (p. 1857)	A wizard dialog

See also **Common dialogs**.

### Miscellaneous windows

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

<i>wxPanel</i> (p. 1170)	A window whose colour changes according to current user settings
<i>wxScrolledWindow</i> (p. 1414)	Window with automatically managed scrollbars
<i>wxGrid</i> (p. 735)	A grid (table) window
<i>wxSplitterWindow</i> (p. 1508)	Window which can be split vertically or horizontally
<i>wxStatusBar</i> (p. 1536)	Implements the status bar on a frame
<i>wxToolBar</i> (p. 1694)	Toolbar class
<i>wxNotebook</i> (p. 1135)	Notebook class
<i>wxListbook</i> (p. 974)	Similar to notebook but using list control
<i>wxChoicebook</i> (p. 189)	Similar to notebook but using choice control

<i>wxTreebook</i> (p. 1721)	Similar to notebook but using tree control
<i>wxSashWindow</i> (p. 1397)	Window with four optional sashes that can be dragged
<i>wxSashLayoutWindow</i> (p. 1394)	Window that can be involved in an IDE-like layout arrangement
<i>wxVScrolledWindow</i> (p. 1790)	As <i>wxScrolledWindow</i> but supports lines of variable height
<i>wxWizardPage</i> (p. 1863)	A base class for the page in wizard dialog.
<i>wxWizardPageSimple</i> (p. 1865)	A page in wizard dialog.

## Common dialogs

*Overview* (p. 2128)

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

<i>wxDialog</i> (p. 496)	Base class for common dialogs
<i>wxColourDialog</i> (p. 221)	Colour chooser dialog
<i>wxDirDialog</i> (p. 513)	Directory selector dialog
<i>wxFileDialog</i> (p. 600)	File selector dialog
<i>wxFindReplaceDialog</i> (p. 651)	Text search/replace dialog
<i>wxMultiChoiceDialog</i> (p. 1129)	Dialog to get one or more selections from a list
<i>wxSingleChoiceDialog</i> (p. 1435)	Dialog to get a single selection from a list and return the string
<i>wxTextEntryDialog</i> (p. 1655)	Dialog to get a single line of text from the user
<i>wxPasswordEntryDialog</i> (p. 1173)	Dialog to get a password from the user
<i>wxFontDialog</i> (p. 670)	Font chooser dialog
<i>wxPageSetupDialog</i> (p. 1158)	Standard page setup dialog
<i>wxPrintDialog</i> (p. 1206)	Standard print dialog
<i>wxProgressDialog</i> (p. 1230)	Progress indication dialog
<i>wxMessageDialog</i> (p. 1106)	Simple message box dialog
<i>wxSymbolPickerDialog</i> (p. 1586)	Symbol selector dialog
<i>wxRichTextFormattingDialog</i> (p. 1353)	A dialog for formatting the content of a <i>wxRichTextCtrl</i>
<i>wxWizard</i> (p. 1857)	A wizard dialog.



## Controls

Typically, these are small windows which provide interaction with the user. Controls that are not static can have *validators* (p. 1767) associated with them.

<i>wxAnimationCtrl</i> (p. 42)	A control to display an animation
<i>wxControl</i> (p. 285)	The base class for controls
<i>wxButton</i> (p. 164)	Push button control, displaying text
<i>wxBitmapButton</i> (p. 139)	Push button control, displaying a bitmap
<i>wxBitmapComboBox</i> (p. 135)	A combobox with bitmaps next to text items
<i>wxToggleButton</i> (p. 1692)	A button which stays pressed when clicked by user.
<i>wxCalendarCtrl</i> (p. 168)	Control showing an entire calendar month
<i>wxCheckBox</i> (p. 180)	Checkbox control
<i>wxCheckListBox</i> (p. 183)	A listbox with a checkbox to the left of each item
<i>wxChoice</i> (p. 186)	Choice control (a combobox without the editable area)
<i>wxComboBox</i> (p. 225)	A choice with an editable area
<i>wxComboCtrl</i> (p. 232)	A combobox with application defined popup
<i>wxDataViewCtrl</i> (p. 317)	A control to tabular or tree like data
<i>wxGauge</i> (p. 701)	A control to represent a varying quantity, such as time remaining
<i>wxGenericDirCtrl</i> (p. 709)	A control for displaying a directory tree
<i>wxHtmlListBox</i> (p. 853)	An abstract class for creating listboxes showing HTML content
<i>wxSimpleHtmlListBox</i> (p. 855)	A listbox showing HTML content
<i>wxStaticBox</i> (p. 1530)	A static, or group box for visually grouping related controls
<i>wxListBox</i> (p. 974)	A list of strings for single or multiple selection
<i>wxListCtrl</i> (p. 980)	A control for displaying lists of strings and/or icons, plus a multicolumn report view
<i>wxListView</i> (p. 1008)	A simpler interface ( <i>façade</i> ) for <i>wxListCtrl</i> in report mode
<i>wxOwnerDrawnComboBox</i> (p. 1152)	A combobox with owner-drawn list items
<i>wxRichTextCtrl</i> (p. 1316)	Generic rich text editing control

<i>wxTextCtrl</i> (p. 1633)	Single or multiline text editing control
<i>wxTreeCtrl</i> (p. 1728)	Tree (hierarchy) control
<i>wxScrollBar</i> (p. 1408)	Scrollbar control
<i>wxSpinButton</i> (p. 1496)	A spin or 'up-down' control
<i>wxSpinCtrl</i> (p. 1500)	A spin control - i.e. spin button and text control
<i>wxStaticText</i> (p. 1534)	One or more lines of non-editable text
<i>wxHyperlinkCtrl</i> (p. 890)	A static text which opens an URL when clicked
<i>wxStaticBitmap</i> (p. 1527)	A control to display a bitmap
<i>wxRadioBox</i> (p. 1241)	A group of radio buttons
<i>wxRadioButton</i> (p. 1249)	A round button to be used with others in a mutually exclusive way
<i>wxSlider</i> (p. 1462)	A slider that can be dragged by the user
<i>wxVListBox</i> (p. 1783)	A listbox supporting variable height rows

### Miscellaneous pickers

A picker control is a control whose appearance and behaviour is highly platform-dependent.

<i>wxColourPickerCtrl</i> (p. 222)	A control which allows the user to choose a colour
<i>wxDirPickerCtrl</i> (p. 515)	A control which allows the user to choose a directory
<i>wxFilePickerCtrl</i> (p. 629)	A control which allows the user to choose a file
<i>wxFontPickerCtrl</i> (p. 678)	A control which allows the user to choose a font
<i>wxDatePickerCtrl</i> (p. 340)	Small date picker control

### Menus

<i>wxMenu</i> (p. 1074)	Displays a series of menu items for selection
<i>wxMenuBar</i> (p. 1088)	Contains a series of menus for use with a frame
<i>wxMenuItem</i> (p. 1099)	Represents a single menu item

### wxAUI - advanced user interface

This is a new set of classes for writing a customizable application interface with built-in docking, floatable panes and a flexible MDI-like interface. Further classes for custom notebooks with draggable tabs etc. are in progress. See also *wxAUI overview* (p. 2197).

<i>wxAuiManager</i> (p. 99)	The central class for managing the interface
-----------------------------	--

<i>wxAuiNotebook</i> (p. 105)	A replacement notebook class with extra features
<i>wxAuiPanelInfo</i> (p. 110)	Describes a single pane
<i>wxAuiDockArt</i> (p. 93)	Art and metrics provider for customizing the docking user interface
<i>wxAuiTabArt</i> (p. 97)	Art and metrics provider for customizing the notebook user interface

## Window layout

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

*Sizer overview* (p. 2098) describes sizer-based layout.

These are the classes relevant to sizer-based layout.

<i>wxSizer</i> (p. 1444)	Abstract base class
<i>wxGridSizer</i> (p. 797)	A sizer for laying out windows in a grid with all fields having the same size
<i>wxFlexGridSizer</i> (p. 652)	A sizer for laying out windows in a flexible grid
<i>wxGridBagSizer</i> (p. 772)	Another grid sizer that lets you specify the cell an item is in, and items can span rows and/or columns.
<i>wxBoxSizer</i> (p. 149)	A sizer for laying out windows in a row or column
<i>wxStaticBoxSizer</i> (p. 1532)	Same as <i>wxBoxSizer</i> , but with a surrounding static box

*Constraints overview* (p. 2094) describes constraints-based layout.

These are the classes relevant to constraints-based window layout.

<i>wxIndividualLayoutConstraint</i> (p. 938)	Represents a single constraint dimension
<i>wxLayoutConstraints</i> (p. 964)	Represents the constraints for a window class

## Device contexts

*Overview* (p. 2120)

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

<i>wxAutoBufferedPaintDC</i> (p. 160)	A helper device context for double buffered drawing inside <b>OnPaint</b> .
<i>wxBufferedDC</i> (p. 157)	A helper device context for double buffered drawing.

<i>wxBufferedPaintDC</i> (p. 159)	A helper device context for double buffered drawing inside <b>OnPaint</b> .
<i>wxClientDC</i> (p. 193)	A device context to access the client area outside <b>OnPaint</b> events
<i>wxPaintDC</i> (p. 1164)	A device context to access the client area inside <b>OnPaint</b> events
<i>wxWindowDC</i> (p. 1855)	A device context to access the non-client area
<i>wxScreenDC</i> (p. 1407)	A device context to access the entire screen
<i>wxDC</i> (p. 456)	The device context base class
<i>wxMemoryDC</i> (p. 1069)	A device context for drawing into bitmaps
<i>wxMetafileDC</i> (p. 1109)	A device context for drawing into metafiles
<i>wxMirrorDC</i> (p. 1115)	A proxy device context allowing for simple mirroring.
<i>wxPostScriptDC</i> (p. 1194)	A device context for drawing into PostScript files
<i>wxPrinterDC</i> (p. 1213)	A device context for drawing to printers

## Graphics device interface

### *Bitmaps overview* (p. 2117)

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

<i>wxColour</i> (p. 214)	Represents the red, blue and green elements of a colour
<i>wxDCClipper</i> (p. 476)	Wraps the operations of setting and destroying the clipping region
<i>wxBitmap</i> (p. 123)	Represents a bitmap
<i>wxBrush</i> (p. 150)	Used for filling areas on a device context
<i>wxBrushList</i> (p. 156)	The list of previously-created brushes
<i>wxCursor</i> (p. 297)	A small, transparent bitmap representing the cursor
<i>wxFont</i> (p. 655)	Represents fonts
<i>wxFontList</i> (p. 673)	The list of previously-created fonts
<i>wxIcon</i> (p. 894)	A small, transparent bitmap for assigning to frames and drawing on device contexts
<i>wxImage</i> (p. 906)	A platform-independent image class
<i>wxImageList</i> (p. 933)	A list of images, used with some controls

<i>wxMask</i> (p. 1036)	Represents a mask to be used with a bitmap for transparent drawing
<i>wxPen</i> (p. 1175)	Used for drawing lines on a device context
<i>wxPenList</i> (p. 1182)	The list of previously-created pens
<i>wxPalette</i> (p. 1166)	Represents a table of indices into RGB values
<i>wxRegion</i> (p. 1264)	Represents a simple or complex region on a window or device context
<i>wxRendererNative</i> (p. 1276)	Abstracts high-level drawing primitives

## Events

### Overview (p. 2077)

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

<i>wxActivateEvent</i> (p. 33)	A window or application activation event
<i>wxCalendarEvent</i> (p. 176)	Used with <i>wxCalendarCtrl</i> (p. 168)
<i>wxCalculateLayoutEvent</i> (p. 167)	Used to calculate window layout
<i>wxChildFocusEvent</i> (p. 280)	A child window focus event
<i>wxClipboardTextEvent</i> (p. 198)	A clipboard copy/cut/paste treebook event event
<i>wxCloseEvent</i> (p. 200)	A close window or end session event
<i>wxCommandEvent</i> (p. 250)	An event from a variety of standard controls
<i>wxContextMenuEvent</i> (p. 281)	An event generated when the user issues a context menu command
<i>wxDateEvent</i> (p. 339)	Used with <i>wxDatePickerCtrl</i> (p. 340)
<i>wxDialUpEvent</i> (p. 505)	Event send by <i>wxDialUpManager</i> (p. 506)
<i>wxDropFilesEvent</i> (p. 557)	A drop files event
<i>wxEraseEvent</i> (p. 571)	An erase background event
<i>wxEvent</i> (p. 572)	The event base class
<i>wxFindDialogEvent</i> (p. 648)	Event sent by <i>wxFindReplaceDialog</i> (p. 651)
<i>wxFocusEvent</i> (p. 655)	A window focus event
<i>wxKeyEvent</i> (p. 956)	A keypress event
<i>wxIconizeEvent</i> (p. 903)	An iconize/restore event

<i>wxIdleEvent</i> (p. 903)	An idle event
<i>wxInitDialogEvent</i> (p. 941)	A dialog initialisation event
<i>wxJoystickEvent</i> (p. 954)	A joystick event
<i>wxListEvent</i> (p. 1000)	A list control event
<i>wxMaximizeEvent</i> (p. 1037)	A maximize event
<i>wxMenuEvent</i> (p. 1098)	A menu event
<i>wxMouseCaptureChangedEvent</i> (p. 1118)	A mouse capture changed event
<i>wxMouseCaptureLostEvent</i> (p. 1119)	A mouse capture lost event
<i>wxMouseEvent</i> (p. 1119)	A mouse event
<i>wxMoveEvent</i> (p. 1128)	A move event
<i>wxNotebookEvent</i> (p. 1144)	A notebook control event
<i>wxNotifyEvent</i> (p. 1146)	A notification event, which can be vetoed
<i>wxPaintEvent</i> (p. 1164)	A paint event
<i>wxProcessEvent</i> (p. 1229)	A process ending event
<i>wxQueryLayoutInfoEvent</i> (p. 1238)	Used to query layout information
<i>wxRichTextEvent</i> (p. 1346)	A rich text editing event
<i>wxScrollEvent</i> (p. 1423)	A scroll event from sliders, stand-alone scrollbars and spin buttons
<i>wxScrollWinEvent</i> (p. 1426)	A scroll event from scrolled windows
<i>wxSizeEvent</i> (p. 1443)	A size event
<i>wxSocketEvent</i> (p. 1490)	A socket event
<i>wxSpinEvent</i> (p. 1503)	An event from <i>wxSpinButton</i> (p. 1496)
<i>wxSplitterEvent</i> (p. 1505)	An event from <i>wxSplitterWindow</i> (p. 1508)
<i>wxSysColourChangedEvent</i> (p. 1590)	A system colour change event
<i>wxTimerEvent</i> (p. 1682)	A timer expiration event
<i>wxTreebookEvent</i> (p. 1726)	A treebook control event
<i>wxTreeEvent</i> (p. 1748)	A tree control event
<i>wxUpdateUIEvent</i> (p. 1752)	A user interface update event
<i>wxWindowCreateEvent</i> (p. 1854)	A window creation event

*wxWindowDestroyEvent* (p. 1856) A window destruction event

*wxWizardEvent* (p. 1862) A wizard event

## Validators

*Overview* (p. 2092)

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

*wxValidator* (p. 1767) Base validator class

*wxTextValidator* (p. 1668) Text control validator class

*wxGenericValidator* (p. 714) Generic control validator class

## Data structures

These are the data structure classes supported by *wxWidgets*.

*wxCmdLineParser* (p. 201) Command line parser class

*wxDateSpan* (p. 343) A logical time interval.

*wxDateTime* (p. 348) A class for date/time manipulations

*wxArray* (p. 71) A dynamic array implementation

*wxArrayString* (p. 83) An efficient container for storing *wxString* (p. 1553) objects

*wxHashMap* (p. 798) A simple hash map implementation

*wxHashSet* (p. 803) A simple hash set implementation

*wxHashTable* (p. 807) A simple hash table implementation (deprecated, use *wxHashMap*)

*wxList* (p. 966) A simple linked list implementation

*wxLongLong* (p. 1032) A portable 64 bit integer type

*wxNode* (p. 1134) Represents a node in the *wxList* implementation

*wxObject* (p. 1148) The root class for most *wxWidgets* classes

*wxPathList* (p. 1174) A class to help search multiple paths

*wxPoint* (p. 1193) Representation of a point

*wxRect* (p. 1252) A class representing a rectangle

*wxRegEx* (p. 1260) Regular expression support

*wxRegion* (p. 1264) A class representing a region

<i>wxString</i> (p. 1553)	A string class
<i>wxStringTokenizer</i> (p. 1583)	A class for interpreting a string as a list of tokens or words
<i>wxRealPoint</i> (p. 1251)	Representation of a point using floating point numbers
<i>wxSize</i> (p. 1440)	Representation of a size
<i>wxTimeSpan</i> (p. 1683)	A time interval.
<i>wxURI</i> (p. 1757)	Represents a Uniform Resource Identifier
<i>wxVariant</i> (p. 1769)	A class for storing arbitrary types that may change at run-time

## Run-time class information system

*Overview* (p. 2044)

*wxWidgets* supports run-time manipulation of class information, and dynamic creation of objects given class names.

<i>wxClassInfo</i> (p. 190)	Holds run-time class information
<i>wxObject</i> (p. 1148)	Root class for classes with run-time information
<i>RTTI macros</i> (p. 1964)	Macros for manipulating run-time information

## Logging features

*Overview* (p. 2069)

*wxWidgets* provides several classes and functions for message logging. Please see the *wxLog overview* (p. 2069) for more details.

<i>wxLog</i> (p. 1018)	The base log class
<i>wxLogStderr</i> (p. 1029)	Log messages to a C STDIO stream
<i>wxLogStream</i> (p. 1029)	Log messages to a C++ iostream
<i>wxLogTextCtrl</i> (p. 1030)	Log messages to a <i>wxTextCtrl</i> (p. 1633)
<i>wxLogWindow</i> (p. 1030)	Log messages to a log frame
<i>wxLogGui</i> (p. 1027)	Default log target for GUI programs
<i>wxLogNull</i> (p. 1027)	Temporarily suppress message logging
<i>wxLogChain</i> (p. 1025)	Allows to chain two log targets
<i>wxLogPassThrough</i> (p. 1028)	Allows to filter the log messages
<i>wxStreamToTextRedirector</i> (p. 1552)	Allows to redirect output sent to <code>cout</code> to a <i>wxTextCtrl</i> (p. 1633)



*Log functions* (p. 1971)

Error and warning logging functions

## **Debugging features**

*Overview* (p. 2072)

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

*wxDebugContext* (p. 483)

Provides memory-checking facilities

*Debugging macros* (p. 1979)

Debug macros for assertion and checking

*WXDEBUG\_NEW* (p. 1968)

Use this macro to give further debugging information

*wxDebugReport* (p. 488)

Base class for creating debug reports in case of a program crash.

*wxDebugReportCompress* (p. 492) Class for creating compressed debug reports.

*wxDebugReportUpload* (p. 494)

Class for uploading compressed debug reports via HTTP.

*wxDebugReportPreview* (p. 493)

Abstract base class for previewing the contents of a debug report.

*wxDebugReportPreviewStd* (p. 494) Standard implementation of *wxDebugReportPreview*.

## **Networking classes**

wxWidgets provides its own classes for socket based networking.

*wxDialUpManager* (p. 506)

Provides functions to check the status of network connection and to establish one

*wxIPv4address* (p. 946)

Represents an Internet address

*wxIPaddress* (p. 944)

Represents an Internet address

*wxSocketBase* (p. 1472)

Represents a socket base object

*wxSocketClient* (p. 1488)

Represents a socket client

*wxSocketServer* (p. 1492)

Represents a socket server

*wxSocketEvent* (p. 1490)

A socket event

*wxFTP* (p. 694)

FTP protocol class

*wxHTTP* (p. 889)

HTTP protocol class

*wxURL* (p. 1763)

Represents a Universal Resource Locator

## **Interprocess communication**

*Overview* (p. 2175)

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

*wxClient* (p. 191), *wxDDEClient* (p. 477) Represents a client

*wxConnection* (p. 276), *wxDDEConnection* (p. 478) Represents the connection between a client and a server

*wxServer* (p. 1431), *wxDDEServer* (p. 482) Represents a server

## **Document/view framework**

*Overview* (p. 2131)

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

*wxDocument* (p. 545) Represents a document

*wxView* (p. 1780) Represents a view

*wxDocTemplate* (p. 540) Manages the relationship between a document class and a view class

*wxDocManager* (p. 527) Manages the documents and views in an application

*wxDocChildFrame* (p. 525) A child frame for showing a document view

*wxDocParentFrame* (p. 538) A parent frame to contain views

## **Printing framework**

*Overview* (p. 2145)

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

*wxPreviewFrame* (p. 1199) Frame for displaying a print preview

*wxPreviewCanvas* (p. 1196) Canvas for displaying a print preview

*wxPreviewControlBar* (p. 1197) Standard control bar for a print preview

*wxPrintDialog* (p. 1206) Standard print dialog

*wxPageSetupDialog* (p. 1158) Standard page setup dialog

*wxPrinter* (p. 1211) Class representing the printer

*wxPrinterDC* (p. 1213) Printer device context

*wxPrintout* (p. 1214) Class representing a particular printout

*wxPrintPreview* (p. 1221) Class representing a print preview

<i>wxPrintData</i> (p. 1200)	Represents information about the document being printed
<i>wxPrintDialogData</i> (p. 1207)	Represents information about the print dialog
<i>wxPageSetupDialogData</i> (p. 1159)	Represents information about the page setup dialog

### **Drag and drop and clipboard classes**

*Drag and drop and clipboard overview* (p. 2150)

<i>wxDataObject</i> (p. 311)	Data object class
<i>wxDataFormat</i> (p. 304)	Represents a data format
<i>wxTextDataObject</i> (p. 1653)	Text data object class
<i>wxFileDataObject</i> (p. 599)	File data object class
<i>wxBitmapDataObject</i> (p. 145)	Bitmap data object class
<i>wxURLDataObject</i> (p. 1766)	URL data object class
<i>wxCustomDataObject</i> (p. 302)	Custom data object class
<i>wxClipboard</i> (p. 195)	Clipboard class
<i>wxDropTarget</i> (p. 561)	Drop target class
<i>wxFileDropTarget</i> (p. 604)	File drop target class
<i>wxTextDropTarget</i> (p. 1654)	Text drop target class
<i>wxDropSource</i> (p. 558)	Drop source class

### **File related classes**

*wxWidgets* has several small classes to work with disk files, see *file classes overview* (p. 2067) for more details.

<i>wxFileName</i> (p. 609)	Operations with the file name and attributes
<i>wxDir</i> (p. 509)	Class for enumerating files/subdirectories.
<i>wxDirTraverser</i> (p. 518)	Class used together with <i>wxDir</i> for recursively enumerating the files/subdirectories
<i>wxFile</i> (p. 591)	Low-level file input/output class.
<i>wxFFFile</i> (p. 584)	Another low-level file input/output class.
<i>wxTempFile</i> (p. 1615)	Class to safely replace an existing file
<i>wxTextFile</i> (p. 1657)	Class for working with text files as with arrays of lines
<i>wxStandardPaths</i> (p. 1522)	Paths for standard directories

## Stream classes

`wxWidgets` has its own set of stream classes, as an alternative to often buggy standard stream libraries, and to provide enhanced functionality.

<code>wxStreamBase</code> (p. 1544)	Stream base class
<code>wxStreamBuffer</code> (p. 1547)	Stream buffer class
<code>wxInputStream</code> (p. 941)	Input stream class
<code>wxOutputStream</code> (p. 1156)	Output stream class
<code>wxCountingOutputStream</code> (p. 293)	Stream class for querying what size a stream would have.
<code>wxFilterInputStream</code> (p. 646)	Filtered input stream class
<code>wxFilterOutputStream</code> (p. 647)	Filtered output stream class
<code>wxBufferedInputStream</code> (p. 161)	Buffered input stream class
<code>wxBufferedOutputStream</code> (p. 161)	Buffered output stream class
<code>wxMemoryInputStream</code> (p. 1073)	Memory input stream class
<code>wxMemoryOutputStream</code> (p. 1073)	Memory output stream class
<code>wxDataInputStream</code> (p. 308)	Platform-independent binary data input stream class
<code>wxDataOutputStream</code> (p. 337)	Platform-independent binary data output stream class
<code>wxTextInputStream</code> (p. 1663)	Platform-independent text data input stream class
<code>wxTextOutputStream</code> (p. 1666)	Platform-independent text data output stream class
<code>wxFileInputStream</code> (p. 608)	File input stream class
<code>wxFileOutputStream</code> (p. 628)	File output stream class
<code>wxFFileInputStream</code> (p. 589)	Another file input stream class
<code>wxFFileOutputStream</code> (p. 590)	Another file output stream class
<code>wxTempFileOutputStream</code> (p. 1618)	Stream to safely replace an existing file
<code>wxStringInputStream</code> (p. 1582)	String input stream class
<code>wxStringOutputStream</code> (p. 1582)	String output stream class
<code>wxZlibInputStream</code> (p. 1899)	Zlib and gzip (compression) input stream class
<code>wxZlibOutputStream</code> (p. 1901)	Zlib and gzip (compression) output stream class
<code>wxZipInputStream</code> (p. 1895)	Input stream for reading from ZIP archives

<i>wxZipOutputStream</i> (p. 1897)	Output stream for writing from ZIP archives
<i>wxTarInputStream</i> (p. 1603)	Input stream for reading from tar archives
<i>wxTarOutputStream</i> (p. 1604)	Output stream for writing from tar archives
<i>wxSocketInputStream</i> (p. 1491)	Socket input stream class
<i>wxSocketOutputStream</i> (p. 1492)	Socket output stream class

## Threading classes

*Multithreading overview* (p. 2149)

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

<i>wxThread</i> (p. 1670)	Thread class
<i>wxThreadHelper</i> (p. 1678)	Manages background threads easily
<i>wxMutex</i> (p. 1131)	Mutex class
<i>wxMutexLocker</i> (p. 1133)	Mutex locker utility class
<i>wxCriticalSection</i> (p. 294)	Critical section class
<i>wxCriticalSectionLocker</i> (p. 295)	Critical section locker utility class
<i>wxCondition</i> (p. 259)	Condition class
<i>wxSemaphore</i> (p. 1427)	Semaphore class

## HTML classes

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

<i>wxHtmlHelpController</i> (p. 838)	HTML help controller class
<i>wxHtmlWindow</i> (p. 872)	HTML window class
<i>wxHtmlEasyPrinting</i> (p. 833)	Simple class for printing HTML
<i>wxHtmlPrintout</i> (p. 863)	Generic HTML wxPrintout class
<i>wxHtmlParser</i> (p. 858)	Generic HTML parser class
<i>wxHtmlTagHandler</i> (p. 869)	HTML tag handler, pluginable into <i>wxHtmlParser</i>
<i>wxHtmlWinParser</i> (p. 883)	HTML parser class for <i>wxHtmlWindow</i>
<i>wxHtmlWinTagHandler</i> (p. 888)	HTML tag handler, pluginable into <i>wxHtmlWinParser</i>

## Rich text classes

`wxWidgets` provides a set of generic classes to edit and print simple rich text with character and paragraph formatting.

<code>wxRichTextCtrl</code> (p. 1316)	A rich text control.
<code>wxRichTextBuffer</code> (p. 1298)	The content of a rich text control.
<code>wxRichTextAttr</code> (p. 1281)	Attributes specifying rich text styling.
<code>wxRichTextCharacterStyleDefinition</code> (p. 1315)	Definition of character styling.
<code>wxRichTextParagraphStyleDefinition</code> (p. 1367)	Definition of paragraph styling.
<code>wxRichTextListStyleDefinition</code> (p. 1365)	Definition of list styling.
<code>wxRichTextStyleSheet</code> (p. 1387)	A set of style definitions.
<code>wxRichTextStyleComboCtrl</code> (p. 1378)	A drop-down control for applying styles.
<code>wxRichTextStyleListBox</code> (p. 1379)	A listbox for applying styles.
<code>wxRichTextStyleOrganiserDialog</code> (p. 1383)	A dialog that can be used for managing or browsing styles.
<code>wxRichTextEvent</code> (p. 1346)	A rich text event.
<code>wxRichTextRange</code> (p. 1373)	Specification for ranges in a rich text control or buffer.
<code>wxRichTextFileHandler</code> (p. 1350)	File handler base class.
<code>wxRichTextHTMLHandler</code> (p. 1362)	A handler for converting rich text to HTML.
<code>wxRichTextXMLHandler</code> (p. 1390)	A handler for loading and saving rich text XML.
<code>wxRichTextFormattingDialog</code> (p. 1353)	A dialog for rich text formatting.
<code>wxRichTextPrinting</code> (p. 1368)	A class for easy printing of rich text buffers.
<code>wxRichTextPrintout</code> (p. 1371)	A class used by <code>wxRichTextPrinting</code> .
<code>wxRichTextHeaderFooterData</code> (p. 1359)	Header and footer data specification.

### Virtual file system classes

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

<code>wxFSFile</code> (p. 692)	Represents a file in the virtual file system
<code>wxFileSystem</code> (p. 633)	Main interface for the virtual file system
<code>wxFileSystemHandler</code> (p. 636)	Class used to announce file system type

### XML classes

<code>wxXmlDocument</code> (p. 1866)	A class to parse XML files.
--------------------------------------	-----------------------------

<code>wxXmlNode</code> (p. 1871)	A class which represents XML nodes.
<code>wxXmlProperty</code> (p. 1877)	A class which represents XML properties.

## XML-based resource system classes

*XML-based resource system (XRC) overview* (p. 2106)

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

<code>wxXmlResource</code> (p. 1878)	The main class for working with resources.
<code>wxXmlResourceHandler</code> (p. 1883)	The base class for XML resource handlers.

## Online help

<code>wxHelpController</code> (p. 809)	Family of classes for controlling help windows
<code>wxHtmlHelpController</code> (p. 838)	HTML help controller class
<code>wxContextHelp</code> (p. 282)	Class to put application into context-sensitive help mode
<code>wxContextHelpButton</code> (p. 284)	Button class for putting application into context-sensitive help mode
<code>wxHelpProvider</code> (p. 817)	Abstract class for context-sensitive help provision
<code>wxSimpleHelpProvider</code> (p. 1432)	Class for simple context-sensitive help provision
<code>wxHelpControllerHelpProvider</code> (p. 815)	Class for context-sensitive help provision via a help controller
<code>wxToolTip</code> (p. 1712)	Class implementing tooltips

## Database classes

*Database classes overview* (p. 2152)

`wxWidgets` provides a set of classes for accessing Microsoft's ODBC (Open Database Connectivity) product, donated by Remstar. This is known as `wxODBC`.

<code>wxDb</code> (p. 375)	ODBC database connection
<code>wxDbTable</code> (p. 415)	Provides access to a database table
<code>wxDbInf</code> (p. 415)	
<code>wxDbTableInf</code> (p. 451)	
<code>wxDbColDef</code> (p. 406)	
<code>wxDbCollnf</code> (p. 408)	

*wxDboColDataPtr* (p. 406)

*wxDboColFor* (p. 407)

*wxDboConnectInf* (p. 409)

*wxDboIdxDef* (p. 414)

## **Miscellaneous**

<i>wxApp</i> (p. 45)	Application class
<i>wxCaret</i> (p. 177)	A caret (cursor) object
<i>wxCmdLineParser</i> (p. 201)	Command line parser class
<i>wxConfig</i> (p. 262)	Classes for configuration reading/writing (using either INI files or registry)
<i>wxDllLoader</i> (p. 522)	Class to work with shared libraries.
<i>wxGLCanvas</i> (p. 715)	Canvas that you can render OpenGL calls to.
<i>wxGLContext</i> (p. 719)	Class to ease sharing of OpenGL data resources.
<i>wxLayoutAlgorithm</i> (p. 961)	An alternative window layout facility
<i>wxProcess</i> (p. 1224)	Process class
<i>wxTimer</i> (p. 1680)	Timer class
<i>wxStopWatch</i> (p. 1543)	Stop watch class
<i>wxMimeTypeManager</i> (p. 1110)	MIME-types manager class
<i>wxSystemSettings</i> (p. 1594)	System settings class for obtaining various global parameters
<i>wxSystemOptions</i> (p. 1590)	System options class for run-time configuration
<i>wxAcceleratorTable</i> (p. 23)	Accelerator table
<i>wxAutomationObject</i> (p. 119)	OLE automation class
<i>wxFontMapper</i> (p. 674)	Font mapping, finding suitable font for given encoding
<i>wxEncodingConverter</i> (p. 568)	Encoding conversions
<i>wxCalendarDateAttr</i> (p. 174)	Used with <i>wxCalendarCtrl</i> (p. 168)
<i>wxQuantize</i> (p. 1237)	Class to perform quantization, or colour reduction
<i>wxSingleInstanceChecker</i> (p. 1438)	Check that only single program instance is running



## Topic overviews

This chapter contains a selection of topic overviews.

### Starting with wxWidgets

*Notes on using the reference* (p. 2026)

*Writing a wxWidgets application: a rough guide* (p. 2027)

*wxWidgets Hello World sample* (p. 2028)

*wxWidgets samples* (p. 2030)

*Introduction to wxPython* (p. 2198)

### Programming with wxWidgets

*Backward compatibility* (p. 2230)

*Runtime class information (RTTI)* (p. 2044)

*Reference counting* (p. 2046)

*Application class: wxApp* (p. 2040)

*Unicode support in wxWidgets* (p. 2056)

*Conversion between Unicode and multibyte strings* (p. 2059)

*Internationalization* (p. 2062)

*Writing non-English applications* (p. 2063)

*Debugging overview* (p. 2072)

*Logging overview* (p. 2069)

*Event handling overview* (p. 2077)

*C++ exceptions overview* (p. 2088)

*Window styles* (p. 2089)

*Window deletion overview* (p. 2089)

*Environment variables* (p. 2198)

### Overviews of non-GUI classes

*String class: wxString* (p. 2047)

*Buffer classes* (p. 2052)

*Date and time classes* (p. 2052)

*Container classes* (p. 2066)

*File classes and functions* (p. 2067)

*Stream classes* (p. 2067)

*Multi-threaded applications* (p. 2149)

*Working with program options: wxConfig* (p. 2074)

*Virtual file system: wxFileSystem* (p. 2075)

*Syntax of the built-in regular expression library* (p. 2210)

*Archive formats such as zip* (p. 2223)

*Interprocess communication* (p. 2175)

*ODBC Database classes* (p. 2152)

### Drawing related classes

*Device contexts* (p. 2120)

*Bitmaps and icons* (p. 2117)

*Fonts* (p. 2121)  
*Fonts encodings* (p. 2122)  
*Printing* (p. 2145)  
*Printing under GTK+* (p. 2149)

## **Overviews of GUI classes**

*Laying out window elements with sizers* (p. 2098)  
*XML-based resource system* (p. 2106)  
*Window sizing* (p. 2042)  
*Scrolling* (p. 2115)  
*Dialogs* (p. 2092)  
*Transferring and validating data* (p. 2092)  
*Data exchange: wxDataObject* (p. 2151)  
*Drag and drop* (p. 2150)  
*Layout constraints* (p. 2094)

## **Overviews of individual controls**

*wxHTML* (p. 2179)  
*wxRichTextCtrl* (p. 2188)  
*wxAUI (advanced user interface)* (p. 2197)  
*Common dialogs* (p. 2128)  
*Toolbar* (p. 2138)  
*wxGrid* (p. 2143)  
*wxTreeCtrl* (p. 2125)  
*wxListCtrl* (p. 2126)  
*wxSplitterWindow* (p. 2123)  
*wxImageList* (p. 2126)  
*wxBookCtrl* (p. 2127)  
*wxTipProvider* (p. 1689)  
*Document/view* (p. 2131)

## **All overview topics (uncategorized)**

## **Notes on using the reference**

In the descriptions of the wxWidgets classes and their member functions, note that descriptions of inherited member functions are not duplicated in derived classes unless their behaviour is different. So in using a class such as wxScrolledWindow, be aware that wxWindow functions may be relevant.

Note also that arguments with default values may be omitted from a function call, for brevity. Size and position arguments may usually be given a value of -1 (the default), in which case wxWidgets will choose a suitable value.

Most strings are returned as wxString objects. However, for remaining char \* return values, the strings are allocated and deallocated by wxWidgets. Therefore, return values should always be copied for long-term use, especially since the same buffer is often used by wxWidgets.

The member functions are given in alphabetical order except for constructors and destructors which appear first.

## Writing a wxWidgets application: a rough guide

To set a wxWidgets application going, you will need to derive a *wxApp* (p. 45) class and override *wxApp::OnInit* (p. 52).

An application must have a top-level *wxFrame* (p. 682) or *wxDialog* (p. 496) window. Each frame may contain one or more instances of classes such as *wxPanel* (p. 1170), *wxSplitterWindow* (p. 1508) or other windows and controls.

A frame can have a *wxMenuBar* (p. 1088), a *wxToolBar* (p. 1694), a status line, and a *wxIcon* (p. 894) for when the frame is iconized.

A *wxPanel* (p. 1170) is used to place controls (classes derived from *wxControl* (p. 285)) which are used for user interaction. Examples of controls are *wxButton* (p. 164), *wxCheckBox* (p. 180), *wxChoice* (p. 186), *wxListBox* (p. 974), *wxRadioBox* (p. 1241), *wxSlider* (p. 1462).

Instances of *wxDialog* (p. 496) can also be used for controls and they have the advantage of not requiring a separate frame.

Instead of creating a dialog box and populating it with items, it is possible to choose one of the convenient common dialog classes, such as *wxMessageDialog* (p. 1106) and *wxFileDialog* (p. 600).

You never draw directly onto a window - you use a *device context* (DC). *wxDC* (p. 456) is the base for *wxClientDC* (p. 193), *wxPaintDC* (p. 1164), *wxMemoryDC* (p. 1069), *wxPostScriptDC* (p. 1194), *wxMemoryDC* (p. 1069), *wxMetafileDC* (p. 1109) and *wxPrinterDC* (p. 1213). If your drawing functions have **wxDC** as a parameter, you can pass any of these DCs to the function, and thus use the same code to draw to several different devices. You can draw using the member functions of **wxDC**, such as *wxDC::DrawLine* (p. 462) and *wxDC::DrawText* (p. 464). Control colour on a window (*wxColour* (p. 214)) with brushes (*wxBrush* (p. 150)) and pens (*wxPen* (p. 1175)).

To intercept events, you add a `DECLARE_EVENT_TABLE` macro to the window class declaration, and put a `BEGIN_EVENT_TABLE ... END_EVENT_TABLE` block in the implementation file. Between these macros, you add event macros which map the event (such as a mouse click) to a member function. These might override predefined event handlers such as for *wxKeyEvent* (p. 956) and *wxMouseEvent* (p. 1119).

Most modern applications will have an on-line, hypertext help system; for this, you need *wxHelp* and the *wxHelpController* (p. 809) class to control *wxHelp*.

GUI applications aren't all graphical wizardry. List and hash table needs are catered for by *wxList* (p. 966) and *wxHashMap* (p. 798). You will undoubtedly need some platform-independent *file functions* (p. 1919), and you may find it handy to maintain and search a list of paths using *wxPathList* (p. 1174). There's a *miscellany* (p. 1951) of operating system and other functions.

See also *Classes by Category* (p. 2007) for a list of classes.

## wxWidgets Hello World sample

As many people have requested a mini-sample to be published here so that some quick judgment concerning syntax and basic principles can be made, you can now look at wxWidgets' "Hello World":

You have to include wxWidgets' header files, of course. This can be done on a file by file basis (such as `#include "wx/window.h"`) or using one global include (`#include "wx/wx.h"`). This is also useful on platforms which support precompiled headers such as all major compilers on the Windows platform.

```
//
// file name: hworld.cpp
//
// purpose: wxWidgets "Hello world"
//

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

#ifdef __BORLANDC__
    #pragma hdrstop
#endif

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

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

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

The main window is created by deriving a class from `wxFrame` and giving it a menu and a status bar in its constructor. Also, any class that wishes to respond to any "event" (such as mouse clicks or messages from the menu or a button) must declare an event table using the macro below. Finally, the way to react to such events must be done in "handlers". In our sample, we react to two menu items, one for "Quit" and one for displaying an "About" window. These handlers should not be virtual.

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

    void OnQuit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);

private:
    DECLARE_EVENT_TABLE()
```

```
};
```

In order to be able to react to a menu command, it must be given a unique identifier such as a const or an enum.

```
enum
{
    ID_Quit = 1,
    ID_About,
};
```

We then proceed to actually implement an event table in which the events are routed to their respective handler functions in the class `MyFrame`. There are predefined macros for routing all common events, ranging from the selection of a list box entry to a resize event when a user resizes a window on the screen. If -1 is given as the ID, the given handler will be invoked for any event of the specified type, so that you could add just one entry in the event table for all menu commands or all button commands etc. The origin of the event can still be distinguished in the event handler as the (only) parameter in an event handler is a reference to a `wxEvent` object, which holds various information about the event (such as the ID of and a pointer to the class, which emitted the event).

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(ID_Quit, MyFrame::OnQuit)
    EVT_MENU(ID_About, MyFrame::OnAbout)
END_EVENT_TABLE()
```

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

```
IMPLEMENT_APP(MyApp)
```

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

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

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

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const
wxSize& size)
    : wxFrame((wxFrame *)NULL, -1, title, pos, size)
{
```

```
wxMenu *menuFile = new wxMenu;

menuFile->Append( ID_About, "&About..." );
menuFile->AppendSeparator();
menuFile->Append( ID_Quit, "E&xit" );

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

SetMenuBar( menuBar );

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

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

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

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

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

## wxWidgets samples

Probably the best way to learn wxWidgets is by reading the source of some 50+ samples provided with it. Many aspects of wxWidgets programming can be learnt from them, but sometimes it is not simple to just choose the right sample to look at. This overview aims at describing what each sample does/demonstrates to make it easier to find the relevant one if a simple grep through all sources didn't help. They also provide some notes about using the samples and what features of wxWidgets are they supposed to test.

There are currently more than 50 different samples as part of wxWidgets and this list is not complete. You should start your tour of wxWidgets with the *minimal sample* (p. 2031) which is the wxWidgets version of "Hello, world!". It shows the basic structure of wxWidgets program and is the most commented sample of all - looking at its source code is recommended.

The next most useful samples are probably *widgets* (p. 2040) and *controls* (p. 2032) which show many of wxWidgets native and generic controls, such as buttons, listboxes, checkboxes, comboboxes etc.

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

<i>wxCalendarCtrl</i> (p. 2031)	Calendar a.k.a. date picker control
<i>wxListCtrl</i> (p. 2036)	List view control
<i>wxTreeCtrl</i> (p. 2039)	Tree view control
<i>wxGrid</i> (p. 2035)	Grid control

Finally, it might be helpful to do a search in the entire sample directory if you can't find the sample showing the control you are interested in by name. Most classes contained in *wxWidgets* occur in at least one of the samples.

### **Minimal sample**

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

### **Animate sample**

The *animate* sample shows how you can use *wxAnimationCtrl* (p. 42) control and shows concept of a platform-dependent animation encapsulated in *wxAnimation* (p. 39).

### **Art provider sample**

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

### **Calendar sample**

This font shows the *calendar control* (p. 168) in action. It shows how to configure the control (see the different options in the calendar menu) and also how to process the notifications from it.

### **Checklist sample**

This sample demonstrates use of the *wxCheckListBox* (p. 183) class intercepting check, select and double click events. It also tests use of various methods modifying the control, such as by deleting items from it or inserting new ones (these functions are actually implemented in the parent class *wxListBox* (p. 974) so the sample tests that class as well). The layout of the dialog is created using a *wxBoxSizer* (p. 149) demonstrating a simple dynamic layout.

### **Config sample**

This sample demonstrates the *wxConfig* (p. 262) classes in a platform independent way, i.e. it uses text based files to store a given configuration under Unix and uses the Registry under Windows.

See *wxConfig overview* (p. 2074) for the descriptions of all features of this class.

## Controls sample

The controls sample is the main test program for most simple controls used in *wxWidgets*. The sample tests their basic functionality, events, placement, modification in terms of colour and font as well as the possibility to change the controls programmatically, such as adding an item to a list box etc. Apart from that, the sample uses a *wxNotebook* (p. 1135) and tests most features of this special control (using bitmap in the tabs, using *wxSizers* (p. 1444) and *constraints* (p. 964) within notebook pages, advancing pages programmatically and vetoing a page change by intercepting the *wxNotebookEvent* (p. 1144).

The various controls tested are listed here:

<i>wxButton</i> (p. 164)	Push button control, displaying text
<i>wxBitmapButton</i> (p. 139)	Push button control, displaying a bitmap
<i>wxCheckBox</i> (p. 180)	Checkbox control
<i>wxChoice</i> (p. 186)	Choice control (a combobox without the editable area)
<i>wxComboBox</i> (p. 225)	A choice with an editable area
<i>wxGauge</i> (p. 701)	A control to represent a varying quantity, such as time remaining
<i>wxStaticBox</i> (p. 1530)	A static, or group box for visually grouping related controls
<i>wxListBox</i> (p. 974)	A list of strings for single or multiple selection
<i>wxSpinCtrl</i>	A spin ctrl with a text field and a 'up-down' control
<i>wxSpinButton</i> (p. 1496)	A spin or 'up-down' control
<i>wxStaticText</i> (p. 1534)	One or more lines of non-editable text
<i>wxStaticBitmap</i> (p. 1527)	A control to display a bitmap
<i>wxRadioBox</i> (p. 1241)	A group of radio buttons
<i>wxRadioButton</i> (p. 1249)	A round button to be used with others in a mutually exclusive way
<i>wxSlider</i> (p. 1462)	A slider that can be dragged by the user

## Database sample

The database sample is a small test program showing how to use the ODBC classes



written by Remstar Intl. Obviously, this sample requires a database with ODBC support to be correctly installed on your system.

### **DebugRpt sample**

This sample shows how to use *wxDebugReport* (p. 488) class to generate a debug report in case of a program crash or otherwise. On start up, it proposes to either crash itself (by dereferencing a NULL pointer) or generate debug report without doing it. Next it initializes the debug report with standard information adding a custom file to it (just a timestamp) and allows to view the information gathered using *wxDebugReportPreview* (p. 493).

For the report processing part of the sample to work you should make available a Web server accepting form uploads, otherwise *wxDebugReportUpload* (p. 494) will report an error.

### **Dialogs sample**

This sample shows how to use the common dialogs available from *wxWidgets*. These dialogs are described in detail in the *Common dialogs overview* (p. 2128).

### **Dialup sample**

This sample shows the *wxDialUpManager* (p. 506) class. In the status bar, it displays the information gathered through its interface: in particular, the current connection status (online or offline) and whether the connection is permanent (in which case a string 'LAN' appears in the third status bar field - but note that you may be on a LAN not connected to the Internet, in which case you will not see this) or not.

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

### **DnD sample**

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

When you run the sample, its screen is split in several parts. On the top, there are two listboxes which show the standard derivations of *wxDropTarget* (p. 561): *wxTextDropTarget* (p. 1654) and *wxFileDropTarget* (p. 604).

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

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

So far, everything we mentioned was implemented with minimal amount of code using

standard `wxWidgets` classes. The more advanced features are demonstrated if you create a shape frame from the main frame menu. A shape is a geometric object which has a position, size and color. It models some application-specific data in this sample. A shape object supports its own private `wxDataFormat` (p. 304) which means that you may cut and paste it or drag and drop (between one and the same or different shapes) from one sample instance to another (or the same). However, chances are that no other program supports this format and so shapes can also be rendered as bitmaps which allows them to be pasted/dropped in many other applications (and, under Windows, also as metafiles which are supported by most of Windows programs as well - try Write/Wordpad, for example).

Take a look at `DnDShapeDataObject` class to see how you may use `wxDataObject` (p. 311) to achieve this.

### **Dynamic sample**

This sample is a very small sample that demonstrates use of the `wxEvtHandler::Connect` (p. 577) method. This method should be used whenever it is not known at compile time which control will receive which event or which controls are actually going to be in a dialog or frame. This is most typically the case for any scripting language that would work as a wrapper for `wxWidgets` or programs where forms or similar datagrams can be created by the users.

See also the *event sample* (p. 2034)

### **Event sample**

The event sample demonstrates various features of the `wxWidgets` events. It shows using dynamic events and connecting/disconnecting the event handlers during run time and also using `PushEventHandler()` (p. 1829) and `PopEventHandler()` (p. 1828).

It replaces the old dynamic sample.

### **Except(ions) sample**

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

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

### **Exec sample**

The exec sample demonstrates the `wxExecute` (p. 1913) and `wxShell` (p. 1916) functions. Both of them are used to execute the external programs and the sample shows how to do this synchronously (waiting until the program terminates) or asynchronously (notification will come later).

It also shows how to capture the output of the child process in both synchronous and asynchronous cases and how to kill the processes with `wxProcess::Kill` (p. 1227) and test

for their existence with `wxProcess::Exists` (p. 1228).

## Font sample

The font sample demonstrates `wxFont` (p. 655), `wxFontEnumerator` (p. 672) and `wxFontMapper` (p. 674) classes. It allows you to see the fonts available (to `wxWidgets`) on the computer and shows all characters of the chosen font as well.

## Grid sample

TODO.

## HTML samples

Eight HTML samples (you can find them in directory `samples/html`) cover all features of the HTML sub-library.

**Test** demonstrates how to create `wxHtmlWindow` (p. 872) and also shows most supported HTML tags.

**Widget** shows how you can embed ordinary controls or windows within an HTML page. It also nicely explains how to write new tag handlers and extend the library to work with unsupported tags.

**About** may give you an idea how to write good-looking About boxes.

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

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

**Printing** explains use of `wxHtmlEasyPrinting` (p. 833) class which serves as as-simple-as-possible interface for printing HTML documents without much work. In fact, only few function calls are sufficient.

**Help** and **Helpview** are variations on displaying HTML help (compatible with MS HTML Help Workshop). *Help* shows how to embed `wxHtmlHelpController` (p. 838) in your application while *Helpview* is a simple tool that only pops up the help window and displays help books given at command line.

## Image sample

The image sample demonstrates use of the `wxImage` (p. 906) class and shows how to download images in a variety of formats, currently PNG, GIF, TIFF, JPEG, BMP, PNM and PCX. The top of the sample shows two rectangles, one of which is drawn directly in the window, the other one is drawn into a `wxBitmap` (p. 123), converted to a `wxImage`, saved as a PNG image and then reloaded from the PNG file again so that conversions between `wxImage` and `wxBitmap` as well as loading and saving PNG files are tested.

At the bottom of the main frame there is a test for using a monochrome bitmap by drawing into a *wxMemoryDC* (p. 1069). The bitmap is then drawn specifying the foreground and background colours with *wxDC::SetTextForeground* (p. 475) and *wxDC::SetTextBackground* (p. 475) (on the left). The bitmap is then converted to a *wxImage* and the foreground colour (black) is replaced with red using *wxImage::Replace* (p. 923).

### **Internat(ionalization) sample**

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

More information about this sample can be found in the `readme.txt` file in its directory. Please see also *i18n overview* (p. 2062).

### **Layout sample**

The layout sample demonstrates the two different layout systems offered by *wxWidgets*. When starting the program, you will see a frame with some controls and some graphics. The controls will change their size whenever you resize the entire frame and the exact behaviour of the size changes is determined using the *wxLayoutConstraints* (p. 964) class. See also the *overview* (p. 2094) and the *wxIndividualLayoutConstraint* (p. 938) class for further information.

The menu in this sample offers two more tests, one showing how to use a *wxBoxSizer* (p. 149) in a simple dialog and the other one showing how to use sizers in connection with a *wxNotebook* (p. 1135) class. See also *wxSizer* (p. 1444).

### **Listctrl sample**

This sample shows the *wxListCtrl* (p. 980) control. Different modes supported by the control (list, icons, small icons, report) may be chosen from the menu.

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

### **Mediaplayer sample**

This sample demonstrates how to use all the features of *wxMediaCtrl* (p. 1058) and play various types of sound, video, and other files.

It replaces the old dynamic sample.

### **Notebook sample**

This samples shows *wxBookCtrl* (p. 2127) family of controls. Although initially it was written to demonstrate *wxNotebook* (p. 1135) only, it can now be also used to see

*wxListbook* (p. 974), *wxChoicebook* (p. 189) and *wxTreebook* (p. 1721) in action. Test each of the controls, their orientation, images and pages using commands through menu.

### Render sample

This sample shows how to replace the default *wxWidgetsrenderer* (p. 1276) and also how to write a shared library (DLL) implementing a renderer and load and unload it during the run-time.

### Rotate sample

This is a simple example which demonstrates how to rotate an image with the *wxImage::Rotate* (p. 924) method. The rotation can be done without interpolation (left mouse button) which will be faster, or with interpolation (right mouse button) which is slower but gives better results.

### Scroll subwindow sample

This sample demonstrates use of the *wxScrolledWindow* (p. 1414) class including placing subwindows into it and drawing simple graphics. It uses the *SetTargetWindow* (p. 1423) method and thus the effect of scrolling does not show in the scrolled window itself, but in one of its subwindows.

Additionally, this sample demonstrates how to optimize drawing operations in *wxWidgets*, in particular using the *wxWindow::IsExposed* (p. 1824) method with the aim to prevent unnecessary drawing in the window and thus reducing or removing flicker on screen.

### Sockets sample

The sockets sample demonstrates how to use the communication facilities provided by *wxSocket* (p. 1472). There are two different applications in this sample: a server, which is implemented using *awxSocketServer* (p. 1492) object, and a client, which is implemented as a *wxSocketClient* (p. 1488).

The server binds to the local address, using TCP port number 3000, sets up an event handler to be notified of incoming connection requests (**wxSOCKET\_CONNECTION** events), and sits there, waiting for clients (*listening*, in socket parlance). For each accepted connection, a new *wxSocketBase* (p. 1472) object is created. These socket objects are independent from the server that created them, so they set up their own event handler, and then request to be notified of **wxSOCKET\_INPUT** (incoming data) or **wxSOCKET\_LOST** (connection closed at the remote end) events. In the sample, the event handler is the same for all connections; to find out which socket the event is addressed to, the *GetSocket* (p. 1491) function is used.

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

The client starts up unconnected, so you can use the *Connect...* option to specify the

address of the server you are going to connect to (the TCP port number is hard-coded as 3000). Once connected, a number of tests are possible. Currently, three tests are implemented. They show how to use the basic IO calls in *wxSocketBase* (p. 1472), such as *Read* (p. 1482), *Write* (p. 1487), *ReadMsg* (p. 1483) and *WriteMsg* (p. 1487), and how to set up the correct IO flags depending on what you are going to do. See the comments in the code for more information. Note that because both clients and connection objects in the server set up an event handler to catch **wxSOCKET\_LOST** events, each one is immediately notified if the other end closes the connection.

There is also a URL test which shows how to use the *wxURL* (p. 1763) class to fetch data from a given URL.

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

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

## Sound sample

The *sound* sample shows how to use *wxSound* (p. 1494) for simple audio output (e.g. notifications).

## Statbar sample

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

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

## Text sample

This sample demonstrates four features: firstly the use and many variants of the *wxTextCtrl* (p. 1633) class (single line, multi line, read only, password, ignoring TAB, ignoring ENTER).

Secondly it shows how to intercept a *wxKeyEvent* (p. 956) in both the raw form using the `EVT_KEY_UP` and `EVT_KEY_DOWN` macros and the higher level from using the `EVT_CHAR` macro. All characters will be logged in a log window at the bottom of the main window. By pressing some of the function keys, you can test some actions in the text ctrl as well as get statistics on the text ctrls, which is useful for testing if these statistics actually are correct.

Thirdly, on platforms which support it, the sample will offer to copy text to the *wxClipboard* (p. 195) and to paste text from it. The GTK version will use the so called PRIMARY SELECTION, which is the pseudo clipboard under X and best known from pasting text to the XTerm program.

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

## Thread sample

This sample demonstrates use of threads in connection with GUI programs. There are two fundamentally different ways to use threads in GUI programs and either way has to take care of the fact that the GUI library itself usually is not multi-threading safe, i.e. that it might crash if two threads try to access the GUI class simultaneously. One way to prevent that is have a normal GUI program in the main thread and some worker threads which work in the background. In order to make communication between the main thread and the worker threads possible, wxWidgets offers the *wxPostEvent* (p. 1960) function and this sample makes use of this function.

The other way to use a so called Mutex (such as those offered in the *wxMutex* (p. 1131) class) that prevent threads from accessing the GUI classes as long as any other thread accesses them. For this, wxWidgets has the *wxMutexGuiEnter* (p. 1919) and *wxMutexGuiLeave* (p. 1919) functions, both of which are used and tested in the sample as well.

See also *Multithreading overview* (p. 2149) and *wxThread* (p. 1670).

## Toolbar sample

The toolbar sample shows the *wxToolBar* (p. 1694) class in action.

The following things are demonstrated:

- Creating the toolbar using *wxToolBar::AddTool* (p. 1698) and *wxToolBar::AddControl* (p. 1698): see *MyApp::InitToolbar* in the sample.
- Using *EVT\_UPDATE\_UI* handler for automatically enabling/disabling toolbar buttons without having to explicitly call *EnableTool*. This is done in *MyFrame::OnUpdateCopyAndCut*.
- Using *wxToolBar::DeleteTool* (p. 1700) and *wxToolBar::InsertTool* (p. 1705) to dynamically update the toolbar.

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

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

## Treectrl sample

This sample demonstrates using the *wxTreeCtrl* (p. 1728) class. Here you may see how to process various notification messages sent by this control and also when they occur (by looking at the messages in the text control in the bottom part of the frame).

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

## Widgets sample

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

## Wizard sample

This sample shows the so-called wizard dialog (implemented using *wxWizard* (p. 1857) and related classes). It shows almost all features supported:

- Using bitmaps with the wizard and changing them depending on the page shown (notice that *wxValidationPage* in the sample has a different image from the other ones)
- Using *TransferDataFromWindow* (p. 1851) to verify that the data entered is correct before passing to the next page (done in *wxValidationPage* which forces the user to check a checkbox before continuing).
- Using more elaborated techniques to allow returning to the previous page, but not continuing to the next one or vice versa (in *wxRadioboxPage*)
- This (*wxRadioboxPage*) page also shows how the page may process the `Cancel` button itself instead of relying on the wizard parent to do it.
- Normally, the order of the pages in the wizard is known at compile-time, but sometimes it depends on the user choices: *wxCheckboxPage* shows how to dynamically decide which page to display next (see also *wxWizardPage* (p. 1863))

## wxApp overview

Classes: *wxApp* (p. 45)

A wxWidgets application does not have a *main* procedure; the equivalent is the *OnInit* (p. 52) member defined for a class derived from *wxApp*. *OnInit* will usually create a top window as a bare minimum.

Unlike in earlier versions of wxWidgets, *OnInit* does not return a frame. Instead it returns a boolean value which indicates whether processing should continue (true) or not (false). You call *wxApp::SetTopWindow* (p. 54) to let wxWidgets know about the top window.

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

An application closes by destroying all windows. Because all frames must be destroyed for



the application to exit, it is advisable to use parent frames wherever possible when creating new frames, so that deleting the top level frame will automatically delete child frames. The alternative is to explicitly delete child frames in the top-level frame's *wxCloseEvent* (p. 200) handler.

In emergencies the *wxExit* (p. 1915) function can be called to kill the application however normally the application shuts down automatically, see *below* (p. 2041).

An example of defining an application follows:

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

IMPLEMENT_APP(DerivedApp)

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

    return true;
}
```

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

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

## Application shutdown

The application normally shuts down when the last of its top level windows is closed. This is normally the expected behaviour and means that it is enough to call *Close()* (p. 1802) in response to the "Exit" menu command if your program has a single top level window. If this behaviour is not desirable *wxApp::SetExitOnFrameDelete* (p. 54) can be called to change it. Note that starting from wxWidgets 2.3.3 such logic doesn't apply for the windows shown before the program enters the main loop: in other words, you can safely show a dialog from *wxApp::OnInit* (p. 52) and not be afraid that your application terminates when this dialog -- which is the last top level window for the moment -- is closed.

Another aspect of the application shutdown is *OnExit* (p. 51) which is called when the application exits but *before* wxWidgets cleans up its internal structures. You should delete all wxWidgets object that you created by the time *OnExit* finishes. In particular, do **not** destroy them from application class' destructor!

For example, this code may crash:

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

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

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

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

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

## Window Sizing Overview

It can sometimes be confusing to keep track of the various size-related attributes of a *wxWindow* (p. 1795), how they relate to each other, and how they interact with sizers. This document will attempt to clear the fog a little, and give some simple explanations of things.

**BestSize:** The best size of a widget depends on what kind of widget it is, and usually also on the contents of the widget. For example *awxListBox* (p. 974)'s best size will be calculated based on how many items it has, up to a certain limit, or *awxButton* (p. 164)'s best size will be calculated based on its label size, but normally won't be smaller than the platform default button size (unless a style flag overrides that). Get the picture? There is a special virtual method in the C++ window classes called `DoGetBestSize()` that a class needs to override if it wants to calculate its own best size based on its content. The default `DoGetBestSize()` is designed for use in container windows, such as *wxPanel* (p. 1170), and works something like this:

1. the window has a sizer then it is used to calculate the best size.
2. if the window has layout constraints then that is used to calculate the best size.

3. if the window has children then the best size is set to be large enough to show all the children.
4. if there are no children then the window's min size will be used for the best size.
5. if there is no min size set, then the current size is used for the best size.

**MinSize:** The min size of a widget is a size that is normally explicitly set by the programmer either with the `SetMinSize()` method or the `SetSizeHints()` method. Most controls will also set the min size to the size given in the control's constructor if a non-default value is passed. Top-level windows such as `aswxFrame` (p. 682) will not allow the user to resize the frame below the min size.

**Size:** The size of a widget can be explicitly set or fetched with the `SetSize()` or `GetSize()` methods. This size value is the size that the widget is currently using on screen and is the way to change the size of something that is not being managed by a sizer.

**ClientSize:** The client size represents the widget's area inside of any borders belonging to the widget and is the area that can be drawn upon in a `EVT_PAINT` event. If a widget doesn't have a border then its client size is the same as its size.

**InitialSize:** The initial size of a widget is the size given to the constructor of the widget, if any. As mentioned above most controls will also set this size value as the control's min size. If the size passed to the constructor is the `defaultwxDefaultSize`, or if the size is not fully specified (such as `aswxSize(150, -1)`) then most controls will fill in the missing size components using the best size and will set the initial size of the control to the resulting size.

**GetEffectiveMinSize():** (formerly `GetBestFittingSize`) A blending of the widget's min size and best size, giving precedence to the min size. For example, if a widget's min size is set to (150, -1) and the best size is (80, 22) then the best fitting size is (150, 22). If the min size is (50, 20) then the best fitting size is (50, 20). This method is what is called by the sizers when determining what the requirements of each item in the sizer is, and is used for calculating the overall minimum needs of the sizer.

**SetInitialSize(size):** (formerly `SetBestFittingSize`) This is a little different than the typical size setters. Rather than just setting an "initial size" attribute it actually sets the minsize to the value passed in, blends that value with the best size, and then sets the size of the widget to be the result. So you can consider this method to be a "Smart SetSize". This method is what is called by the constructor of most controls to set the minsize and initial size of the control.

**window.Fit():** The `Fit()` method sets the size of a window to fit around its children. If it has no children then nothing is done, if it does have children then the size of the window is set to the window's best size.

**sizer.Fit(window):** This sets the size of the window to be large enough to accommodate the minimum size needed by the sizer, (along with a few other constraints...) If the sizer is the one that is assigned to the window then this should be equivalent to `window.Fit()`.

**sizer.Layout():** Recalculates the minimum space needed by each item in the sizer, and

then lays out the items within the space currently allotted to the sizer.

**window.Layout():** If the window has a sizer then it sets the space given to the sizer to the current size of the window, which results in a call to `sizer.Layout()`. If the window has layout constraints instead of a sizer then the constraints algorithm is run. The `Layout()` method is what is called by the default `EVT_SIZE` handler for container windows.

## Runtime class information (aka RTTI) overview

Classes: *wxObject* (p. 1148), *wxClassInfo* (p. 190).

One of the failings of C++ used to be that no run-time information was provided about a class and its position in the inheritance hierarchy. Another, which still persists, 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 *wxWidgets* is no exception. As it originated before the addition of RTTI to the C++ standard and as support for it is still missing from some (albeit old) compilers, *wxWidgets* doesn't (yet) use it, but provides its own macro-based RTTI system.

In the future, the standard C++ RTTI will be used though and you're encouraged to use whenever possible the *wxDynamicCast()* (p. 1969) macro which, for the implementations that support it, is defined just as `dynamic_cast<>` and uses *wxWidgets* RTTI for all the others. This macro is limited to *wxWidgets* classes only and only works with pointers (unlike the real `dynamic_cast<>` which also accepts references).

Each class that you wish to be known to the type system should have a macro such as `DECLARE_DYNAMIC_CLASS` just inside the class declaration. The macro `IMPLEMENT_DYNAMIC_CLASS` should be in the implementation file. Note that these are entirely optional; use them if you wish to check object types, or create instances of classes using the class name. However, it is good to get into the habit of adding these macros for all classes.

Variations on these *macros* (p. 1964) 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* (p. 1149) uses the linked list of *wxClassInfo*. It takes a *wxClassInfo* argument, so use `CLASSINFO(className)` to return an appropriate *wxClassInfo* pointer to use in this function.

The function *wxCreateDynamicObject* (p. 1968) 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` (p. 190).

## **wxClassInfo**

*Runtime class information (aka RTTI) overview* (p. 2044)

Class: `wxClassInfo` (p. 190)

This class stores meta-information about classes. An application may use macros such as `DECLARE_DYNAMIC_CLASS` and `IMPLEMENT_DYNAMIC_CLASS` to record run-time information about a class, including:

- its position in the inheritance hierarchy;
- the base class name(s) (up to two base classes are permitted);
- a string representation of the class name;
- a function that can be called to construct an instance of this class.

The `DECLARE_...` macros declare a static `wxClassInfo` variable in a class, which is initialized by macros of the form `IMPLEMENT_...` in the implementation C++ file. Classes whose instances may be constructed dynamically are given a global constructor function which returns a new object.

You can get the `wxClassInfo` for a class by using the `CLASSINFO` macro, e.g. `CLASSINFO(wxFrame)`. You can get the `wxClassInfo` for an object using `wxObject::GetClassInfo`.

See also `wxObject` (p. 1148) and `wxCreateDynamicObject` (p. 1968).

## **Example**

In a header file `frame.h`:

```
class wxFrame : public wxWindow
{
    DECLARE_DYNAMIC_CLASS(wxFrame)

private:
    wxString m_title;

public:
    ...
};
```

In a C++ file `frame.cpp`:

```
IMPLEMENT_DYNAMIC_CLASS(wxFrame, wxWindow)

wxFrame::wxFrame()
{
    ...
}
```

```
}
```

## Reference counting

### Why you shouldn't care about it

Many `wxWidgets` objects use a technique known as *reference counting*, also known as *copy on write* (COW). This means that when an object is assigned to another, no copying really takes place: only the reference count on the shared object data is incremented and both objects share the same data (a very fast operation).

But as soon as one of the two (or more) objects is modified, the data has to be copied because the changes to one of the objects shouldn't be seen in the others. As data copying only happens when the object is written to, this is known as COW.

What is important to understand is that all this happens absolutely transparently to the class users and that whether an object is shared or not is not seen from the outside of the class - in any case, the result of any operation on it is the same.

### Object comparison

The `==` and `!=` operators of *wxWidgets COW objects* (p. 2046) always do a deep comparison.

This means that the equality operator will return `true` if two objects are identical and not only if they share the same data.

Note that `wxWidgets` follows the *STL philosophy*: when a comparison operator cannot be implemented efficiently (like for e.g. `wxImage`'s `==` operator which would need to compare pixel-by-pixel the entire image's data), it's not implemented at all.

That's why not all reference-counted `wxWidgets` classes provide comparison operators.

Also note that if you only need to do a `shallow` comparison between two `wxObject` (p. 1148)-derived classes, you should not use the `==` and `!=` operators but rather the `wxObject::IsSameAs` (p. 1150) function.

### Object destruction

When a COW object destructor is called, it may not delete the data: if it's shared, the destructor will just decrement the shared data's reference count without destroying it.

Only when the destructor of the last object owning the data is called, the data is really destroyed. As for all other COW-things, this happens transparently to the class users so that you shouldn't care about it.

### List of reference-counted `wxWidgets` classes

The following classes in `wxWidgets` have efficient (i.e. fast) assignment operators and

copy constructors since they are reference-counted:

*wxAcceleratorTable* (p. 23)  
*wxAnimation* (p. 39)  
*wxBitmap* (p. 123)  
*wxBrush* (p. 150)  
*wxCursor* (p. 297)  
*wxFont* (p. 655)  
*wxIcon* (p. 894)  
*wxImage* (p. 906)  
*wxMetafile* (p. 1108)  
*wxPalette* (p. 1166)  
*wxPen* (p. 1175)  
*wxRegion* (p. 1264)  
*wxString* (p. 1553)  
*wxVariant* (p. 1769)  
*wxVariantData* (p. 1778)

**Note that the list above reports the objects which are reference-counted in all ports of wxWidgets; some ports may use this technique also for other classes. Make your own reference-counted class**

Reference counting can be implemented easily using *wxObject* (p. 1148) and *wxObjectRefData* (p. 1151) classes.

First, derive a new class from *wxObjectRefData* (p. 1151) and put there the memory-consuming data.

Then derive a new class from *wxObject* (p. 1148) and implement there the public interface which will be seen by the user of your class. You'll probably want to add a function to your class which does the cast from *wxObjectRefData* (p. 1151) to your class-specific shared data; e.g.:

```
MyClassRefData *GetData() const { return  
wx_static_cast(MyClassRefData*, m_refData); }
```

in fact, all times you'll need to read the data from your *wxObject*-derived class, you'll need to call such function.

Very important, all times you need to actually modify the data placed inside your *wxObject*-derived class, you must first call the *wxObject::UnShare* function to be sure that the modifications won't affect other instances which are eventually sharing your object's data.

## wxString overview

Classes: *wxString* (p. 1553), *wxArrayString* (p. 83), *wxStringTokenizer* (p. 1583)

### Introduction

`wxString` is a class which represents a character string of arbitrary length (limited by `MAX_INT` which is usually 2147483647 on 32 bit machines) and containing arbitrary characters. The ASCII NUL character is allowed, but be aware that in the current string implementation some methods might not work correctly in this case.

`wxString` works with both ASCII (traditional, 7 or 8 bit, characters) as well as Unicode (wide characters) strings.

This class has all the standard operations you can expect to find in a string class: dynamic memory management (string extends to accommodate new characters), construction from other strings, C strings and characters, assignment operators, access to individual characters, string concatenation and comparison, substring extraction, case conversion, trimming and padding (with spaces), searching and replacing and both C-like `Printf()` (p. 1570) and stream-like insertion functions as well as much more - see *wxString* (p. 1553) for a list of all functions.

### Comparison of `wxString` to other string classes

The advantages of using a special string class instead of working directly with C strings are so obvious that there is a huge number of such classes available. The most important advantage is the need to always remember to allocate/free memory for C strings; working with fixed size buffers almost inevitably leads to buffer overflows. At last, C++ has a standard string class (`std::string`). So why the need for `wxString`?

There are several advantages:

1. **Efficiency** This class was made to be as efficient as possible: both in terms of size (each `wxString` objects takes exactly the same space as a `char *` pointer, sing *reference counting* (p. 2050)) and speed. It also provides performance *statistics gathering code* (p. 2051) which may be enabled to fine tune the memory allocation strategy for your particular application - and the gain might be quite big.
2. **Compatibility** This class tries to combine almost full compatibility with the old `wxWidgets 1.xx` `wxString` class, some reminiscence to MFC `CString` class and 90% of the functionality of `std::string` class.
3. **Rich set of functions** Some of the functions present in `wxString` are very useful but don't exist in most of other string classes: for example, *AfterFirst* (p. 1562), *BeforeLast* (p. 1563), *operator<<* (p. 1578) or *Printf* (p. 1570). Of course, all the standard string operations are supported as well.
4. **Unicode** `wxString` is Unicode friendly: it allows to easily convert to and from ANSI and Unicode strings in any build mode (see the *Unicode overview* (p. 2056) for more details) and maps to either `string` or `wstring` transparently depending on the current mode.
5. **Used by `wxWidgets`** And, of course, this class is used everywhere inside `wxWidgets` so there is no performance loss which would result from conversions of objects of any other string class (including `std::string`) to `wxString` internally by `wxWidgets`.

However, there are several problems as well. The most important one is probably that



there are often several functions to do exactly the same thing: for example, to get the length of the string either one of `length()`, `Len()` (p. 1569) or `Length()` (p. 1569) may be used. The first function, as almost all the other functions in lowercase, is `std::string` compatible. The second one is "native" `wxString` version and the last one is `wxWidgets` 1.xx way. So the question is: which one is better to use? And the answer is that:

**The usage of `std::string` compatible functions is strongly advised!** It will both make your code more familiar to other C++ programmers (who are supposed to have knowledge of `std::string` but not of `wxString`), let you reuse the same code in both `wxWidgets` and other programs (by just typedefing `wxString` as `std::string` when used outside `wxWidgets`) and by staying compatible with future versions of `wxWidgets` which will probably start using `std::string` sooner or later too.

In the situations where there is no corresponding `std::string` function, please try to use the new `wxString` methods and not the old `wxWidgets` 1.xx variants which are deprecated and may disappear in future versions.

### Some advice about using `wxString`

Probably the main trap with using this class is the implicit conversion operator to `const char *`. It is advised that you use `c_str()` (p. 1563) instead to clearly indicate when the conversion is done. Specifically, the danger of this implicit conversion may be seen in the following code fragment:

```
// this function converts the input string to uppercase, output it to
// the screen
// and returns the result
const char *SayHELLO(const wxString& input)
{
    wxString output = input.Upper();

    printf("Hello, %s!\n", output);

    return output;
}
```

There are two nasty bugs in these three lines. First of them is in the call to the `printf()` function. Although the implicit conversion to C strings is applied automatically by the compiler in the case of

```
puts(output);
```

because the argument of `puts()` is known to be of the type `const char *`, this is **not** done for `printf()` which is a function with variable number of arguments (and whose arguments are of unknown types). So this call may do anything at all (including displaying the correct string on screen), although the most likely result is a program crash. The solution is to use `c_str()` (p. 1563): just replace this line with

```
printf("Hello, %s!\n", output.c_str());
```

The second bug is that returning `output` doesn't work. The implicit cast is used again, so the code compiles, but as it returns a pointer to a buffer belonging to a local variable which is deleted as soon as the function exits, its contents is totally arbitrary. The solution to this

problem is also easy: just make the function return `wxString` instead of a C string.

This leads us to the following general advice: all functions taking string arguments should take `const wxString&` (this makes assignment to the strings inside the function faster because of *reference counting* (p. 2050)) and all functions returning strings should return `wxString` - this makes it safe to return local variables.

## Other string related functions and classes

As most programs use character strings, the standard C library provides quite a few functions to work with them. Unfortunately, some of them have rather counter-intuitive behaviour (like `strncpy()` which doesn't always terminate the resulting string with a `NULL`) and are in general not very safe (passing `NULL` to them will probably lead to program crash). Moreover, some very useful functions are not standard at all. This is why in addition to all `wxString` functions, there are also a few global string functions which try to correct these problems: `wxIsEmpty()` (p. 1930) verifies whether the string is empty (returning `true` for `NULL` pointers), `wxStrlen()` (p. 1931) also handles `NULL`s correctly and returns 0 for them and `wxStricmp()` (p. 1931) is just a platform-independent version of case-insensitive string comparison function known either as `stricmp()` or `strcasecmp()` on different platforms.

The `<wx/string.h>` header also defines `wxSnprintf` (p. 1932) and `wxVsnprintf` (p. 1933) functions which should be used instead of the inherently dangerous standard `sprintf()` and which use `snprintf()` instead which does buffer size checks whenever possible. Of course, you may also use `wxString::Printf` (p. 1570) which is also safe.

There is another class which might be useful when working with `wxString`: `wxStringTokenizer` (p. 1583). It is helpful when a string must be broken into tokens and replaces the standard C library `strtok()` function.

And the very last string-related class is `wxArrayString` (p. 83): it is just a version of the "template" dynamic array class which is specialized to work with strings. Please note that this class is specially optimized (using its knowledge of the internal structure of `wxString`) for storing strings and so it is vastly better from a performance point of view than a `wxObjectArray` of `wxStrings`.

## Reference counting and why you shouldn't care about it

All considerations for `wxObject`-derived *reference counted* (p. 2046) objects are valid also for `wxString`, even if it does not derive from `wxObject`.

Probably the unique case when you might want to think about reference counting is when a string character is taken from a string which is not a constant (or a constant reference). In this case, due to C++ rules, the "read-only" *operator[]* (which is the same as `GetChar()` (p. 1567)) cannot be chosen and the "read/write" *operator[]* (the same as `GetWritableChar()` (p. 1567)) is used instead. As the call to this operator may modify the string, its data is unshared (COW is done) and so if the string was really shared there is some performance loss (both in terms of speed and memory consumption). In the rare cases when this may be important, you might prefer using `GetChar()` (p. 1567) instead of the array subscript operator for this reasons. Please note that `at()` (p. 1558) method has

the same problem as the subscript operator in this situation and so using it is not really better. Also note that if all string arguments to your functions are passed as *const wxString*& (see the section *Some advice* (p. 2049)) this situation will almost never arise because for constant references the correct operator is called automatically.

## Tuning wxString for your application

**Note:** this section is strictly about performance issues and is absolutely not necessary to read for using wxString class. Please skip it unless you feel familiar with profilers and relative tools. If you do read it, please also read the preceding section about *reference counting* (p. 2050).

For the performance reasons wxString doesn't allocate exactly the amount of memory needed for each string. Instead, it adds a small amount of space to each allocated block which allows it to not reallocate memory (a relatively expensive operation) too often as when, for example, a string is constructed by subsequently adding one character at a time to it, as for example in:

```
// delete all vowels from the string
wxString DeleteAllVowels(const wxString& original)
{
    wxString result;

    size_t len = original.length();
    for ( size_t n = 0; n < len; n++ )
    {
        if ( strchr("aeuio", tolower(original[n])) == NULL )
            result += original[n];
    }

    return result;
}
```

This is quite a common situation and not allocating extra memory at all would lead to very bad performance in this case because there would be as many memory (re)allocations as there are consonants in the original string. Allocating too much extra memory would help to improve the speed in this situation, but due to a great number of wxString objects typically used in a program would also increase the memory consumption too much.

The very best solution in precisely this case would be to use *Alloc()* (p. 1561) function to preallocate, for example, len bytes from the beginning - this will lead to exactly one memory allocation being performed (because the result is at most as long as the original string).

However, using *Alloc()* is tedious and so wxString tries to do its best. The default algorithm assumes that memory allocation is done in granularity of at least 16 bytes (which is the case on almost all of wide-spread platforms) and so nothing is lost if the amount of memory to allocate is rounded up to the next multiple of 16. Like this, no memory is lost and 15 iterations from 16 in the example above won't allocate memory but use the already allocated pool.

The default approach is quite conservative. Allocating more memory may bring important performance benefits for programs using (relatively) few very long strings. The amount of

memory allocated is configured by the setting of *EXTRA\_ALLOC* in the file *string.cpp* during compilation (be sure to understand why its default value is what it is before modifying it!). You may try setting it to greater amount (say twice *nLen*) or to 0 (to see performance degradation which will follow) and analyse the impact of it on your program. If you do it, you will probably find it helpful to also define *WXSTRING\_STATISTICS* symbol which tells the *wxString* class to collect performance statistics and to show them on *stderr* on program termination. This will show you the average length of strings your program manipulates, their average initial length and also the percent of times when memory wasn't reallocated when string concatenation was done but the already preallocated memory was used (this value should be about 98% for the default allocation policy, if it is less than 90% you should really consider fine tuning *wxString* for your application).

It goes without saying that a profiler should be used to measure the precise difference the change to *EXTRA\_ALLOC* makes to your program.

## Buffer classes overview

*wxWidgets* uses two classes of classes for dealing with buffers in memory.

The first is one for dealing with character buffers, namely *wxCharBuffer* for *char* pointer or multi-byte *c* strings and *wxWCharBuffer* for *wchar\_t* pointer or wide character *c* strings.

Secondly, *wxWidgets* uses, although only rarely currently, *wxMemoryBuffer* for dealing with raw buffers in memory.

### **wxXCharBuffer Overview**

#### **General Usage**

As mentioned, *wxCharBuffer* and its wide character variant *wxWCharBuffer* deal with *c* strings in memory. They have two constructors, one in which you pass the *c* string you want them to have a copy of, and another where you specify the size of the buffer in memory in characters you want.

*wxCharBuffer* and its variant only contain the *c* string as a member, so they can be used safely to *c* functions with variable arguments such as *printf*. They also contain standard assignment, character access operators and a copy constructor.

#### **Destruction**

It should be noted that on destruction *wxCharBuffer* and its wide character variant delete the *c* string that hold onto. If you want to get the pointer to the buffer and don't want *wxCharBuffer* to delete it on destruction, use the member function *release* to do so.

## Date and time classes overview

Classes: *wxDateTime* (p. 348), *wxDateSpan* (p. 343), *wxTimeSpan* (p. 1683), *wxCalendarCtrl* (p. 168)

### **Introduction**

`wxWidgets` provides a set of powerful classes to work with dates and times. Some of the supported features of `wxDateTime` (p. 348) class are:

Wide range	The range of supported dates goes from about 4714 B.C. to some 480 million years in the future.
Precision	Not using floating point calculations anywhere ensures that the date calculations don't suffer from rounding errors.
Many features	Not only all usual calculations with dates are supported, but also more exotic week and year day calculations, work day testing, standard astronomical functions, conversion to and from strings in either strict or free format.
Efficiency	Objects of <code>wxDateTime</code> are small (8 bytes) and working with them is fast

### All date/time classes at a glance

There are 3 main classes declared in `<wx/datetime.h>`: except `wxDateTime` (p. 348) itself which represents an absolute moment in time, there are also two classes - `wxTimeSpan` (p. 1683) and `wxDateSpan` (p. 343) - which represent the intervals of time.

There are also helper classes which are used together with `wxDateTime`: `wxDateTimeHolidayAuthority` (p. 375) which is used to determine whether a given date is a holiday or not and `wxDateTimeWorkDays` (p. 375) which is a derivation of this class for which (only) Saturdays and Sundays are the holidays. See more about these classes in the discussion of the *holidays* (p. 2056).

Finally, in other parts of this manual you may find mentions of `wxDate` and `wxTime` classes. *These classes* (p. 2056) are obsolete and superseded by `wxDateTime`.

### wxDateTime characteristics

`wxDateTime` (p. 348) stores the time as a signed number of milliseconds since the Epoch which is fixed, by convention, to Jan 1, 1970 - however this is not visible to the class users (in particular, dates prior to the Epoch are handled just as well (or as bad) as the dates after it). But it does mean that the best resolution which can be achieved with this class is 1 millisecond.

The size of `wxDateTime` object is 8 bytes because it is represented as a 64 bit integer. The resulting range of supported dates is thus approximatively 580 million years, but due to the current limitations in the Gregorian calendar support, only dates from Nov 24, 4714BC are supported (this is subject to change if there is sufficient interest in doing it).

Finally, the internal representation is time zone independent (always in GMT) and the time zones only come into play when a date is broken into year/month/day components. See more about *timezones* (p. 2055) below.

Currently, the only supported calendar is Gregorian one (which is used even for the dates prior to the historic introduction of this calendar which was first done on Oct 15, 1582 but is, generally speaking, country, and even region, dependent). Future versions will probably have Julian calendar support as well and support for other calendars (Maya, Hebrew, Chinese...) is not ruled out.

### **Difference between `wxDateSpan` and `wxTimeSpan`**

While there is only one logical way to represent an absolute moment in the time (and hence only one `wxDateTime` class), there are at least two methods to describe a time interval.

First, there is the direct and self-explaining way implemented by `wxTimeSpan` (p. 1683): it is just a difference in milliseconds between two moments in time. Adding or subtracting such an interval to `wxDateTime` is always well-defined and is a fast operation.

But in the daily life other, calendar-dependent time interval specifications are used. For example, 'one month later' is commonly used. However, it is clear that this is not the same as `wxTimeSpan` of  $60*60*24*31$  seconds because 'one month later' Feb 15 is Mar 15 and not Mar 17 or Mar 16 (depending on whether the year is leap or not).

This is why there is another class for representing such intervals called `wxDateSpan` (p. 343). It handles these sort of operations in the most natural way possible, but note that manipulating with intervals of this kind is not always well-defined. Consider, for example, Jan 31 + '1 month': this will give Feb 28 (or 29), i.e. the last day of February and not the non-existent Feb 31. Of course, this is what is usually wanted, but you still might be surprised to notice that now subtracting back the same interval from Feb 28 will result in Jan 28 and **not** Jan 31 we started with!

So, unless you plan to implement some kind of natural language parsing in the program, you should probably use `wxTimeSpan` instead of `wxDateSpan` (which is also more efficient). However, `wxDateSpan` may be very useful in situations when you do need to understand what 'in a month' means (of course, it is just `wxDateTime::Now() + wxDateSpan::Month()`).

### **Date arithmetics**

Many different operations may be performed with the dates, however not all of them make sense. For example, multiplying a date by a number is an invalid operation, even though multiplying either of the time span classes by a number is perfectly valid.

Here is what can be done:

#### **Addition**

a `wxTimeSpan` or `wxDateSpan` can be added to `wxDateTime` resulting in a new `wxDateTime` object and also 2 objects of the same span class can be added together giving another object of the same class.

#### **Subtraction**

the same types of operations as above are allowed and, additionally, a difference between

two `wxDateTime` objects can be taken and this will yield `wxTimeSpan`.

**Multiplication**

a `wxTimeSpan` or `wxDateSpan` object can be multiplied by an integer number resulting in an object of the same type.

**Unary minus**

a `wxTimeSpan` or `wxDateSpan` object may finally be negated giving an interval of the same magnitude but of opposite time direction.

For all these operations there are corresponding global (overloaded) operators and also member functions which are synonyms for them: `Add()`, `Subtract()` and `Multiply()`. Unary minus as well as composite assignment operations (like `+=`) are only implemented as members and `Neg()` is the synonym for unary minus.

**Time zone considerations**

Although the time is always stored internally in GMT, you will usually work in the local time zone. Because of this, all `wxDateTime` constructors and setters which take the broken down date assume that these values are for the local time zone. Thus, `wxDateTime(1, wxDateTime::Jan, 1970)` will not correspond to the `wxDateTime` Epoch unless you happen to live in the UK.

All methods returning the date components (year, month, day, hour, minute, second...) will also return the correct values for the local time zone by default, so, generally, doing the natural things will lead to natural and correct results.

If you only want to do this, you may safely skip the rest of this section. However, if you want to work with different time zones, you should read it to the end.

In this (rare) case, you are still limited to the local time zone when constructing `wxDateTime` objects, i.e. there is no way to construct a `wxDateTime` corresponding to the given date in, say, Pacific Standard Time. To do it, you will need to call *ToTimezone* (p. 374) or *MakeTimezone* (p. 374) methods to adjust the date for the target time zone. There are also special versions of these functions *ToUTC* (p. 374) and *MakeUTC* (p. 374) for the most common case - when the date should be constructed in UTC.

You also can just retrieve the value for some time zone without converting the object to it first. For this you may pass `TimeZone` argument to any of the methods which are affected by the time zone (all methods getting date components and the date formatting ones, for example). In particular, the `Format()` family of methods accepts a `TimeZone` parameter and this allows to simply print time in any time zone.

To see how to do it, the last issue to address is how to construct a `TimeZone` object which must be passed to all these methods. First of all, you may construct it manually by specifying the time zone offset in seconds from GMT, but usually you will just use one of the *symbolic time zone names* (p. 348) and let the conversion constructor do the job. I.e. you would just write

```
wxDateTime dt(...whatever...);  
printf("The time is %s in local time zone", dt.FormatTime().c_str());
```

```
printf("The time is %s in GMT",  
dt.FormatTime(wxDateTime::GMT).c_str());
```

## Daylight saving time (DST)

DST (a.k.a. 'summer time') handling is always a delicate task which is better left to the operating system which is supposed to be configured by the administrator to behave correctly. Unfortunately, when doing calculations with date outside of the range supported by the standard library, we are forced to deal with these issues ourselves.

Several functions are provided to calculate the beginning and end of DST in the given year and to determine whether it is in effect at the given moment or not, but they should not be considered as absolutely correct because, first of all, they only work more or less correctly for only a handful of countries (any information about other ones appreciated!) and even for them the rules may perfectly well change in the future.

The time zone handling *methods* (p. 2055) use these functions too, so they are subject to the same limitations.

## wxDateTime and Holidays

TODO.

## Compatibility

The old classes for date/time manipulations ported from wxWidgets version 1.xx are still included but are reimplemented in terms of wxDateTime. However, using them is strongly discouraged because they have a few quirks/bugs and were not 'Y2K' compatible.

## Unicode support in wxWidgets

This section briefly describes the state of the Unicode support in wxWidgets. Read it if you want to know more about how to write programs able to work with characters from languages other than English.

### What is Unicode?

wxWidgets has support for compiling in Unicode mode on the platforms which support it. Unicode is a standard for character encoding which addresses the shortcomings of the previous, 8 bit standards, by using at least 16 (and possibly 32) bits for encoding each character. This allows to have at least 65536 characters (what is called the BMP, or basic multilingual plane) and possible  $2^{32}$  of them instead of the usual 256 and is sufficient to encode all of the world languages at once. More details about Unicode may be found at [www.unicode.org](http://www.unicode.org).

As this solution is obviously preferable to the previous ones (think of incompatible encodings for the same language, locale chaos and so on), many modern operating systems support it. The probably first example is Windows NT which uses only Unicode internally since its very first version.



Writing internationalized programs is much easier with Unicode and, as the support for it improves, it should become more and more so. Moreover, in the Windows NT/2000 case, even the program which uses only standard ASCII can profit from using Unicode because they will work more efficiently - there will be no need for the system to convert all strings the program uses to/from Unicode each time a system call is made.

## Unicode and ANSI modes

As not all platforms supported by wxWidgets support Unicode (fully) yet, in many cases it is unwise to write a program which can only work in Unicode environment. A better solution is to write programs in such way that they may be compiled either in ANSI (traditional) mode or in the Unicode one.

This can be achieved quite simply by using the means provided by wxWidgets. Basically, there are only a few things to watch out for:

- Character type (`char` or `wchar_t`)
- Literal strings (i.e. `"Hello, world!"` or `'*'`)
- String functions (`strlen()`, `strcpy()`, ...)
- Special preprocessor tokens (`__FILE__`, `__DATE__` and `__TIME__`)

Let's look at them in order. First of all, each character in an Unicode program takes 2 bytes instead of usual one, so another type should be used to store the characters (`char` only holds 1 byte usually). This type is called `wchar_t` which stands for *wide-character type*.

Also, the string and character constants should be encoded using wide characters (`wchar_t` type) which typically take 2 or 4 bytes instead of `char` which only takes one. This is achieved by using the standard C (and C++) way: just put the letter `'L'` after any string constant and it becomes a *long* constant, i.e. a wide character one. To make things a bit more readable, you are also allowed to prefix the constant with `'L'` instead of putting it after it.

Of course, the usual standard C functions don't work with `wchar_t` strings, so another set of functions exists which do the same thing but accept `wchar_t *` instead of `char *`. For example, a function to get the length of a wide-character string is called `wcslen()` (compare with `strlen()` - you see that the only difference is that the "str" prefix standing for "string" has been replaced with "wcs" standing for "wide-character string").

And finally, the standard preprocessor tokens enumerated above expand to ANSI strings but it is more likely that Unicode strings are wanted in the Unicode build. wxWidgets provides the macros `__TFILE__`, `__TDATE__` and `__TTIME__` which behave exactly as the standard ones except that they produce ANSI strings in ANSI build and Unicode ones in the Unicode build.

To summarize, here is a brief example of how a program which can be compiled in both ANSI and Unicode modes could look like:

```
#ifdef __UNICODE__
    wchar_t wch = L'*';
```

```
const wchar_t *ws = L"Hello, world!";
int len = wcslen(ws);

wprintf(L"Compiled at %s\n", __TDATE__);
#else // ANSI
char ch = '*';
const char *s = "Hello, world!";
int len = strlen(s);

printf("Compiled at %s\n", __DATE__);
#endif // Unicode/ANSI
```

Of course, it would be nearly impossible to write such programs if it had to be done this way (try to imagine the number of `#ifdef UNICODE` an average program would have had!). Luckily, there is another way - see the next section.

## Unicode support in wxWidgets

In wxWidgets, the code fragment from above should be written instead:

```
wxChar ch = wxT('*');
wxString s = wxT("Hello, world!");
int len = s.Len();
```

What happens here? First of all, you see that there are no more `#ifdefs` at all. Instead, we define some types and macros which behave differently in the Unicode and ANSI builds and allow us to avoid using conditional compilation in the program itself.

We have a `wxChar` type which maps either on `char` or `wchar_t` depending on the mode in which program is being compiled. There is no need for a separate type for strings though, because the standard `wxString` (p. 1553) supports Unicode, i.e. it stores either ANSI or Unicode strings depending on the compile mode.

Finally, there is a special `wxT()` (p. 1932) macro which should enclose all literal strings in the program. As it is easy to see comparing the last fragment with the one above, this macro expands to nothing in the (usual) ANSI mode and prefixes `'L'` to its argument in the Unicode mode.

The important conclusion is that if you use `wxChar` instead of `char`, avoid using C style strings and use `wxString` instead and don't forget to enclose all string literals inside `wxT()` (p. 1932) macro, your program automatically becomes (almost) Unicode compliant!

Just let us state once again the rules:

- Always use `wxChar` instead of `char`
- Always enclose literal string constants in `wxT()` (p. 1932) macro unless they're already converted to the right representation (another standard wxWidgets macro `_()` (p. 1933) does it, for example, so there is no need for `wxT( )` in this case) or you intend to pass the constant directly to an external function which doesn't accept wide-character strings.
- Use `wxString` instead of C style strings.

## Unicode and the outside world

We have seen that it was easy to write Unicode programs using wxWidgets types and macros, but it has been also mentioned that it isn't quite enough. Although everything works fine inside the program, things can get nasty when it tries to communicate with the outside world which, sadly, often expects ANSI strings (a notable exception is the entire Win32 API which accepts either Unicode or ANSI strings and which thus makes it unnecessary to ever perform any conversions in the program). GTK 2.0 only accepts UTF-8 strings.

To get an ANSI string from a wxString, you may use the `mb_str()` function which always returns an ANSI string (independently of the mode - while the usual `c_str()` (p. 1563) returns a pointer to the internal representation which is either ASCII or Unicode). More rarely used, but still useful, is `wc_str()` function which always returns the Unicode string.

Sometimes it is also necessary to go from ANSI strings to wxString. In this case, you can use the converter-constructor, as follows:

```
const char* ascii_str = "Some text";
wxString str(ascii_str, wxConvUTF8);
```

This code also compiles fine under a non-Unicode build of wxWidgets, but in that case the converter is ignored.

For more information about converters and Unicode see the *wxMBConv classes overview* (p. 2059).

## Unicode-related compilation settings

You should define `wxUSE_UNICODE` to 1 to compile your program in Unicode mode. This currently works for wxMSW, wxGTK, wxMac and wxX11. If you compile your program in ANSI mode you can still define `wxUSE_WCHAR_T` to get some limited support for `wchar_t` type.

This will allow your program to perform conversions between Unicode strings and ANSI ones (using *wxMBConv classes* (p. 2059)) and construct wxString objects from Unicode strings (presumably read from some external file or elsewhere).

## wxMBConv classes overview

Classes: *wxMBConv* (p. 1038), *wxMBConvLibc*, *wxMBConvUTF7* (p. 1044), *wxMBConvUTF8* (p. 1044), *wxCSCnv* (p. 296), *wxMBConvUTF16* (p. 1045), *wxMBConvUTF32* (p. 1046)

The wxMBConv classes in wxWidgets enable an Unicode-aware application to easily convert between Unicode and the variety of 8-bit encoding systems still in use.

## Background: The need for conversion

As programs are becoming more and more globalized, and users exchange documents across country boundaries as never before, applications increasingly need to take into account all the different character sets in use around the world. It is no longer enough to

just depend on the default byte-sized character set that computers have traditionally used.

A few years ago, a solution was proposed: the Unicode standard. Able to contain the complete set of characters in use in one unified global coding system, it would resolve the character set problems once and for all.

But it hasn't happened yet, and the migration towards Unicode has created new challenges, resulting in "compatibility encodings" such as UTF-8. A large number of systems out there still depends on the old 8-bit encodings, hampered by the huge amounts of legacy code still widely deployed. Even sending Unicode data from one Unicode-aware system to another may need encoding to an 8-bit multibyte encoding (UTF-7 or UTF-8 is typically used for this purpose), to pass unhindered through any traditional transport channels.

### **Background: The wxString class**

If you have compiled wxWidgets in Unicode mode, the wxChar type will become identical to wchar\_t rather than char, and a wxString stores wxChars. Hence, all wxString manipulation in your application will then operate on Unicode strings, and almost as easily as working with ordinary char strings (you just need to remember to use the wxT() macro to encapsulate any string literals).

But often, your environment doesn't want Unicode strings. You could be sending data over a network, or processing a text file for some other application. You need a way to quickly convert your easily-handled Unicode data to and from a traditional 8-bit encoding. And this is what the wxMBConv classes do.

### **wxMBConv classes**

The base class for all these conversions is the wxMBConv class (which itself implements standard libc locale conversion). Derived classes include wxMBConvLibc, several different wxMBConvUTFxxx classes, and wxCSConv, which implement different kinds of conversions. You can also derive your own class for your own custom encoding and use it, should you need it. All you need to do is override the MB2WC and WC2MB methods.

### **wxMBConv objects**

Several of the wxWidgets-provided wxMBConv classes have predefined instances (wxConvLibc, wxConvFileName, wxConvUTF7, wxConvUTF8, wxConvLocal). You can use these predefined objects directly, or you can instantiate your own objects.

A variable, wxConvCurrent, points to the conversion object that the user interface is supposed to use, in the case that the user interface is not Unicode-based (like with GTK+ 1.2). By default, it points to wxConvLibc or wxConvLocal, depending on which works best on the current platform.

### **wxCSSConv**

The wxCSConv class is special because when it is instantiated, you can tell it which character set it should use, which makes it meaningful to keep many instances of them

around, each with a different character set (or you can create a `wxCSCnv` instance on the fly).

The predefined `wxCSCnv` instance, `wxCnvLocal`, is preset to use the default user character set, but you should rarely need to use it directly, it is better to go through `wxCnvCurrent`.

## Converting strings

Once you have chosen which object you want to use to convert your text, here is how you would use them with `wxString`. These examples all assume that you are using a Unicode build of `wxWidgets`, although they will still compile in a non-Unicode build (they just won't convert anything).

Example 1: Constructing a `wxString` from input in current encoding.

```
wxString str(input_data, *wxCnvCurrent);
```

Example 2: Input in UTF-8 encoding.

```
wxString str(input_data, wxCnvUTF8);
```

Example 3: Input in KOI8-R. Construction of `wxCSCnv` instance on the fly.

```
wxString str(input_data, wxCSCnv(wxT("koi8-r")));
```

Example 4: Printing a `wxString` to stdout in UTF-8 encoding.

```
puts(str.mb_str(wxCnvUTF8));
```

Example 5: Printing a `wxString` to stdout in custom encoding. Using preconstructed `wxCSCnv` instance.

```
wxCSCnv cust(user_encoding);  
printf("Data: %s\n", (const char*) str.mb_str(cust));
```

Note: Since `mb_str()` returns a temporary `wxCharBuffer` to hold the result of the conversion, you need to explicitly cast it to `const char*` if you use it in a vararg context (like with `printf`).

## Converting buffers

If you have specialized needs, or just don't want to use `wxString`, you can also use the conversion methods of the conversion objects directly. This can even be useful if you need to do conversion in a non-Unicode build of `wxWidgets`; converting a string from UTF-8 to the current encoding should be possible by doing this:

```
wxString str(wxCnvUTF8.cMB2WC(input_data), *wxCnvCurrent);
```

Here, `cMB2WC` of the `UTF8` object returns a `wxWCharBuffer` containing a Unicode string. The `wxString` constructor then converts it back to an 8-bit character set using the passed conversion object, `*wxCnvCurrent`. (In a Unicode build of `wxWidgets`, the constructor ignores the passed conversion object and retains the Unicode data.)

This could also be done by first making a `wxString` of the original data:

```
wxString input_str(input_data);  
wxString str(input_str.wc_str(wxConvUTF8), *wxConvCurrent);
```

To print a `wxChar` buffer to a non-Unicode stdout:

```
printf("Data: %s\n", (const char*)  
wxConvCurrent->cWX2MB(unicode_data));
```

If you need to do more complex processing on the converted data, you may want to store the temporary buffer in a local variable:

```
const wxWX2MBbuf tmp_buf = wxConvCurrent->cWX2MB(unicode_data);  
const char *tmp_str = (const char*) tmp_buf;  
printf("Data: %s\n", tmp_str);  
process_data(tmp_str);
```

If a conversion had taken place in `cWX2MB` (i.e. in a Unicode build), the buffer will be deallocated as soon as `tmp_buf` goes out of scope. (The macro `wxWX2MBbuf` reflects the correct return value of `cWX2MB` (either `char*` or `wxCharBuffer`), except for the `const`.)

## Internationalization

Although internationalization of an application (i18n for short) involves far more than just translating its text messages to another message - date, time and currency formats need changing too, some languages are written left to right and others right to left, character encoding may differ and many other things may need changing too - it is a necessary first step. `wxWidgets` provides facilities for message translation with its *wxLocale* (p. 1011) class and is itself fully translated into several languages. Please consult `wxWidgets` home page for the most up-to-date translations - and if you translate it into one of the languages not done yet, your translations would be gratefully accepted for inclusion into future versions of the library!

The `wxWidgets` approach to i18n closely follows the GNU gettext package. `wxWidgets` uses the message catalogs which are binary compatible with gettext catalogs and this allows to use all of the programs in this package to work with them. But note that no additional libraries are needed during run-time, however, so you have only the message catalogs to distribute and nothing else.

During program development you will need the gettext package for working with message catalogs. **Warning:** gettext versions < 0.10 are known to be buggy, so you should find a later version of it!

There are two kinds of message catalogs: source catalogs which are text files with extension `.po` and binary catalogs which are created from the source ones with *msgfmt* program (part of gettext package) and have the extension `.mo`. Only the binary files are needed during program execution.

The program i18n involves several steps:

1. Translating the strings in the program text using *wxGetTranslation* (p. 1930) or

equivalently the `_()` (p. 1933) macro.

2. Extracting the strings to be translated from the program: this uses the work done in the previous step because `xgettext` program used for string extraction recognises the standard `_()` as well as (using its `-k` option) our `wxGetTranslation` and extracts all strings inside the calls to these functions. Alternatively, you may use `-a` option to extract all the strings, but it will usually result in many strings being found which don't have to be translated at all. This will create a text message catalog - a `.po` file.
3. Translating the strings extracted in the previous step to other language(s). It involves editing the `.po` file.
4. Compiling the `.po` file into `.mo` file to be used by the program.
5. Installing the `.mo` files with your application in the appropriate location for the target system which is the one returned by `wxStandardPaths::GetLocalizedResourcesDir(wxStandardPaths::ResourceCat_Messages)` (p. 1525). If the message catalogs are not installed in this default location you may explicitly use `AddCatalogLookupPathPrefix()` (p. 1013) to still allow `wxWidgets` to find them but it is strongly recommended to use the default directory.
6. Setting the appropriate locale in your program to use the strings for the given language: see `wxLocale` (p. 1011).

See also the GNU `gettext` documentation linked from `docs/html/index.htm` in your `wxWidgets` distribution.

See also *Writing non-English applications* (p. 2063). It focuses on handling charsets related problems.

Finally, take a look at the *i18n sample* (p. 2036) which shows you how all this looks in practice.

### Translating menu accelerators

If you translate the accelerator modifier names (Ctrl, Alt and Shift) in your menu labels, you may find the accelerators no longer work. In your message catalogs, you need to provide individual translations of these modifiers from their lower case names (`ctrl`, `alt`, `shift`) so that the `wxWidgets` accelerator code can recognise them even when translated. `wxWidgets` does not provide translations for all of these currently. `wxWidgets` does not yet handle translated special key names such as Backspace, End, Insert, etc.

## Writing non-English applications

This article describes how to write applications that communicate with the user in a language other than English. Unfortunately many languages use different charsets under Unix and Windows (and other platforms, to make the situation even more complicated). These charsets usually differ in so many characters that it is impossible to use the same texts under all platforms.

The wxWidgets library provides a mechanism that helps you avoid distributing many identical, only differently encoded, packages with your application (e.g. help files and menu items in iso8859-13 and windows-1257). Thanks to this mechanism you can, for example, distribute only iso8859-13 data and it will be handled transparently under all systems.

Please read *Internationalization* (p. 2062) which describes the locales concept.

In the following text, wherever *iso8859-2* and *windows-1250* are used, any encodings are meant and any encodings may be substituted there.

## Locales

The best way to ensure correctly displayed texts in a GUI across platforms is to use locales. Write your in-code messages in English or without diacritics and put real messages into the message catalog (see *Internationalization* (p. 2062)).

A standard .po file begins with a header like this:

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR Free Software Foundation, Inc.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"POT-Creation-Date: 1999-02-19 16:03+0100\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: ENCODING\n"
```

Note this particular line:

```
"Content-Type: text/plain; charset=CHARSET\n"
```

It specifies the charset used by the catalog. All strings in the catalog are encoded using this charset.

You have to fill in proper charset information. Your .po file may look like this after doing so:

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR Free Software Foundation, Inc.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"POT-Creation-Date: 1999-02-19 16:03+0100\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=iso8859-2\n"
```



```
"Content-Transfer-Encoding: 8bit\n"
```

(Make sure that the header is **not** marked as *fuzzy*.)

`wxWidgets` is able to use this catalog under any supported platform (although `iso8859-2` is a Unix encoding and is normally not understood by Windows).

How is this done? When you tell the `wxLocale` class to load a message catalog that contains a correct header, it checks the charset. The catalog is then converted to the charset used (see `wxLocale::GetSystemEncoding` (p. 1016) and `wxLocale::GetSystemEncodingName` (p. 1016)) by the user's operating system. This is the default behaviour of the `wxLocale` (p. 1011) class; you can disable it by **not** passing `wxLOCALE_CONV_ENCODING` to `wxLocale::Init` (p. 1017).

### Non-English strings or 8-bit characters in the source code

By convention, you should only use characters without diacritics (i.e. 7-bit ASCII strings) for msgids in the source code and write them in English.

If you port software to `wxWindows`, you may be confronted with legacy source code containing non-English string literals. Instead of translating the strings in the source code to English and putting the original strings into message catalog, you may configure `wxWidgets` to use non-English msgids and translate to English using message catalogs:

1. you use the program `xgettext` to extract the strings from the source code, specify the option `--from-code=<source code charset>`.
2. the source code language and charset as arguments to `wxLocale::AddCatalog` (p. 1012). For example:

```
locale.AddCatalog(_T("myapp"),
                  wxLANGUAGE_GERMAN, _T("iso-8859-1"));
```

### Font mapping

You can use `wxMBConv` classes (p. 2059) and `wxFontMapper` (p. 674) to display text:

```
if (!wxFontMapper::Get()->IsEncodingAvailable(enc, facename))
{
    wxFontEncoding alternative;
    if (wxFontMapper::Get()->GetAltForEncoding(enc, &alternative,
                                              facename, false))
    {
        wxCSConv
        convFrom(wxFontMapper::Get()->GetEncodingName(enc));
        wxCSConv
        convTo(wxFontMapper::Get()->GetEncodingName(alternative));
        text = wxString(text.mb_str(convFrom), convTo);
    }
    else
        ...failure (or we may try iso8859-1/7bit ASCII)...
}
...display text...
```

### Converting data

You may want to store all program data (created documents etc.) in the same encoding, let's say `utf-8`. You can use `wxCSCnv` (p. 296) class to convert data to the encoding used by the system your application is running on (see `wxLocale::GetSystemEncoding` (p. 1016)).

### Help files

If you're using `wxHtmlHelpController` (p. 838) there is no problem at all. You only need to make sure that all the HTML files contain the META tag, e.g.

```
<meta http-equiv="Content-Type" content="text/html;
charset=iso8859-2">
```

and that the hhp project file contains one additional line in the `OPTIONS` section:

```
Charset=iso8859-2
```

This additional entry tells the HTML help controller what encoding is used in contents and index tables.

## Container classes overview

Classes: `wxList` (p. 966), `wxArray` (p. 71)

`wxWidgets` uses itself several container classes including doubly-linked lists and dynamic arrays (i.e. arrays which expand automatically when they become full). For both historical and portability reasons `wxWidgets` does not use STL which provides the standard implementation of many container classes in C++. First of all, `wxWidgets` has existed since well before STL was written, and secondly we don't believe that today compilers can deal really well with all of STL classes (this is especially true for some less common platforms). Of course, the compilers are evolving quite rapidly and hopefully their progress will allow to base future versions of `wxWidgets` on STL - but this is not yet the case.

`wxWidgets` container classes don't pretend to be as powerful or full as STL ones, but they are quite useful and may be compiled with absolutely any C++ compiler. They're used internally by `wxWidgets`, but may, of course, be used in your programs as well if you wish.

The list classes in `wxWidgets` are doubly-linked lists which may either own the objects they contain (meaning that the list deletes the object when it is removed from the list or the list itself is destroyed) or just store the pointers depending on whether you called or not `wxList::DeleteContents` (p. 968) method.

Dynamic arrays resemble C arrays but with two important differences: they provide run-time range checking in debug builds and they automatically expand the allocated memory when there is no more space for new items. They come in two sorts: the "plain" arrays which store either built-in types such as "char", "int" or "bool" or the pointers to arbitrary objects, or "object arrays" which own the object pointers to which they store.

For the same portability reasons, the container classes implementation in `wxWidgets` does not use templates, but is rather based on C preprocessor i.e. is done with the macros: `WX_DECLARE_LIST` and `WX_DEFINE_LIST` for the linked lists and `WX_DECLARE_ARRAY`, `WX_DECLARE_OBJARRAY` and `WX_DEFINE_OBJARRAY`

for the dynamic arrays. The "DECLARE" macro declares a new container class containing the elements of given type and is needed for all three types of container classes: lists, arrays and objarrays. The "DEFINE" classes must be inserted in your program in a place where the **full declaration of container element class is in scope** (i.e. not just forward declaration), otherwise destructors of the container elements will not be called! As array classes never delete the items they contain anyhow, there is no `WX_DEFINE_ARRAY` macro for them.

Examples of usage of these macros may be found in *wxList* (p. 966) and *wxArray* (p. 71) documentation.

Finally, `wxWidgets` predefines several commonly used container classes. `wxList` is defined for compatibility with previous versions as a list containing `wxObjects` and `wxStringList` as a list of C-style strings (`char *`), both of these classes are deprecated and should not be used in new programs. The following array classes are defined: `wxArrayInt`, `wxArrayLong`, `wxArrayPtrVoid` and `wxArrayString`. The first three store elements of corresponding types, but `wxArrayString` is somewhat special: it is an optimized version of `wxArray` which uses its knowledge about *wxString* (p. 1553) reference counting schema.

## File classes and functions overview

Classes: *wxFile* (p. 591), *wxDir* (p. 509), *wxTempFile* (p. 1615), *wxTextFile* (p. 1657)

Functions: see *file functions* (p. 1919).

`wxWidgets` provides some functions and classes to facilitate working with files. As usual, the accent is put on cross-platform features which explains, for example, the *wxTextFile* (p. 1657) class which may be used to convert between different types of text files (DOS/Unix/Mac).

`wxFile` may be used for low-level IO. It contains all the usual functions to work with files (opening/closing, reading/writing, seeking, and so on) but compared with using standard C functions, has error checking (in case of an error a message is logged using *wxLog* (p. 1018) facilities) and closes the file automatically in the destructor which may be quite convenient.

`wxTempFile` is a very small file designed to make replacing the files contents safer - see its *documentation* (p. 1615) for more details.

`wxTextFile` is a general purpose class for working with small text files on line by line basis. It is especially well suited for working with configuration files and program source files. It can be also used to work with files with "non native" line termination characters and write them as "native" files if needed (in fact, the files may be written in any format).

`wxDir` is a helper class for enumerating the files or subdirectories of a directory. It may be used to enumerate all files, only files satisfying the given template mask or only non-hidden files.

## wxStreams overview

Classes: *wxStreamBase* (p. 1544), *wxStreamBuffer* (p. 1547), *wxInputStream* (p. 941), *wxOutputStream* (p. 1156), *wxFilterInputStream* (p. 646), *wxFilterOutputStream* (p. 647)

### **Purpose of wxStream**

Standard C++ streams can cause problems on several platforms: they work quite well in most cases, but in the multi-threaded case, for example, they have many problems. Some Borland compilers refuse to work at all with them and using iostreams on Linux makes writing programs that are binary compatible across different Linux distributions, impossible.

Therefore, wxStreams have been added to wxWidgets so that applications can reliably compile and run on all supported platforms without dependence on a particular release of libg++.

wxStreams is divided in two main parts:

1. the core: *wxStreamBase*, *wxStreamBuffer*, *wxInputStream*, *wxOutputStream*, *wxFilterIn/OutputStream*
2. the "IO" classes: *wxSocketIn/OutputStream*, *wxDataIn/OutputStream*, *wxFileIn/OutputStream*, ...

*wxStreamBase* is the base definition of a stream. It defines, for example, the API of *OnSysRead*, *OnSysWrite*, *OnSysSeek* and *OnSysTell*. These functions are really implemented by the "IO" classes. *wxInputStream* and *wxOutputStream* inherit from it.

*wxStreamBuffer* is a cache manager for *wxStreamBase*: it manages a stream buffer linked to a stream. One stream can have multiple stream buffers but one stream have always one autoinitialized stream buffer.

*wxInputStream* is the base class for read-only streams. It implements *Read*, *Seek* (I for Input), and all read or IO generic related functions. *wxOutputStream* does the same thing but it is for write-only streams.

*wxFilterIn/OutputStream* is the base class definition for stream filtering. Stream filtering means a stream which does no syscall but filters data which are passed to it and then pass them to another stream. For example, *wxZLibInputStream* is an inline stream decompressor.

The "IO" classes implements the specific parts of the stream. This could be nothing in the case of *wxMemoryIn/OutputStream* which bases itself on *wxStreamBuffer*. This could also be a simple link to the a true syscall (for example *read(...)*, *write(...)*).

### **Generic usage: an example**

Usage is simple. We can take the example of *wxFileInputStream* and here is some sample code:

```
...
// The constructor initializes the stream buffer and open the file
descriptor
// associated to the name of the file.
wxFileInputStream in_stream("the_file_to_be_read");
```

```
// Ok, read some bytes ... nb_datas is expressed in bytes.
in_stream.Read(data, nb_datas);
if (in_stream.LastError() != wxSTREAM_NOERROR) {
    // Oh oh, something bad happens.
    // For a complete list, look into the documentation at wxStreamBase.
}

// You can also inline all like this.
if (in_stream.Read(data, nb_datas).LastError() != wxSTREAM_NOERROR)
{
    // Do something.
}

// You can also get the last number of bytes REALLY put into the buffer.
size_t really_read = in_stream.LastRead();

// Ok, moves to the beginning of the stream. SeekI returns the last
position
// in the stream counted from the beginning.
off_t old_position = in_stream.SeekI(0, wxFromBeginning);

// What is my current position ?
off_t position = in_stream.TellI();

// wxFileInputStream will close the file descriptor on destruction.
```

## wxLog classes overview

Classes: *wxLog* (p. 1018),  
*wxLogStderr* (p. 1029),  
*wxLogStream* (p. 1029),  
*wxLogTextCtrl* (p. 1030),  
*wxLogWindow* (p. 1030),  
*wxLogGui* (p. 1027),  
*wxLogNull* (p. 1027),  
*wxLogChain* (p. 1025),  
*wxLogPassThrough* (p. 1028),  
*wxStreamToTextRedirector* (p. 1552)

This is a general overview of logging classes provided by wxWidgets. The word logging here has a broad sense, including all of the program output, not only non-interactive messages. The logging facilities included in wxWidgets provide the base *wxLog* class which defines the standard interface for a *log target* as well as several standard implementations of it and a family of functions to use with them.

First of all, no knowledge of *wxLog* classes is needed to use them. For this, you should only know about *wxLogXXX()* functions. All of them have the same syntax as *printf()* or *vprintf()*, i.e. they take the format string as the first argument and respectively a variable number of arguments or a variable argument list pointer. Here are all of them:

- **wxLogFatalError** which is like *wxLogError*, but also terminates the program with

the exit code 3 (using *abort()* standard function). Unlike for all the other logging functions, this function can't be overridden by a log target.

- **wxLogError** is the function to use for error messages, i.e. the messages that must be shown to the user. The default processing is to pop up a message box to inform the user about it.
- **wxLogWarning** for warnings - they are also normally shown to the user, but don't interrupt the program work.
- **wxLogMessage** is for all normal, informational messages. They also appear in a message box by default (but it can be changed, see below).
- **wxLogVerbose** is for verbose output. Normally, it is suppressed, but might be activated if the user wishes to know more details about the program progress (another, but possibly confusing name for the same function is **wxLogInfo**).
- **wxLogStatus** is for status messages - they will go into the status bar of the active or specified (as the first argument) *wxFrame* (p. 682) if it has one.
- **wxLogSysError** is mostly used by *wxWidgets* itself, but might be handy for logging errors after system call (API function) failure. It logs the specified message text as well as the last system error code (*errno* or *::GetLastError()* depending on the platform) and the corresponding error message. The second form of this function takes the error code explicitly as the first argument.
- **wxLogDebug** is the right function for debug output. It only does anything at all in the debug mode (when the preprocessor symbol `__WXDEBUG__` is defined) and expands to nothing in release mode (otherwise). **Tip:** under Windows, you must either run the program under debugger or use a 3rd party program such as *DbgView* (<http://www.sysinternals.com>) to actually see the debug output.
- **wxLogTrace** as **wxLogDebug** only does something in debug build. The reason for making it a separate function from it is that usually there are a lot of trace messages, so it might make sense to separate them from other debug messages which would be flooded in them. Moreover, the second version of this function takes a trace mask as the first argument which allows to further restrict the amount of messages generated.

The usage of these functions should be fairly straightforward, however it may be asked why not use the other logging facilities, such as C standard *stdio* functions or C++ streams. The short answer is that they're all very good generic mechanisms, but are not really adapted for *wxWidgets*, while the log classes are. Some of advantages in using *wxWidgets* log functions are:

- **Portability** It is a common practice to use *printf()* statements or *cout/cerr* C++ streams for writing out some (debug or otherwise) information. Although it works just fine under Unix, these messages go strictly nowhere under Windows where the *stdout* of GUI programs is not assigned to anything. Thus, you might view *wxLogMessage()* as a simple substitute for *printf()*.

You can also redirect the `wxLogXXX` calls to `cout` by just writing: `wxLog *logger=new wxLogStream(&cout); wxLog::SetActiveTarget(logger);`

Finally, there is also a possibility to redirect the output sent to `cout` to a `wxTextCtrl` (p. 1633) by using the `wxStreamToTextRedirector` (p. 1552) class.

- **Flexibility** The output of `wxLog` functions can be redirected or suppressed entirely based on their importance, which is either impossible or difficult to do with traditional methods. For example, only error messages, or only error messages and warnings might be logged, filtering out all informational messages.
- **Completeness** Usually, an error message should be presented to the user when some operation fails. Let's take a quite simple but common case of a file error: suppose that you're writing your data file on disk and there is not enough space. The actual error might have been detected inside `wxWidgets` code (say, in `wxFile::Write`), so the calling function doesn't really know the exact reason of the failure, it only knows that the data file couldn't be written to the disk. However, as `wxWidgets` uses `wxLogError()` in this situation, the exact error code (and the corresponding error message) will be given to the user together with "high level" message about data file writing error.

After having enumerated all the functions which are normally used to log the messages, and why would you want to use them we now describe how all this works.

`wxWidgets` has the notion of a *log target*: it is just a class deriving from `wxLog` (p. 1018). As such, it implements the virtual functions of the base class which are called when a message is logged. Only one log target is *active* at any moment, this is the one used by `wxLogXXX()` functions. The normal usage of a log object (i.e. object of a class derived from `wxLog`) is to install it as the active target with a call to `SetActiveTarget()` and it will be used automatically by all subsequent calls to `wxLogXXX()` functions.

To create a new log target class you only need to derive it from `wxLog` and implement one (or both) of `DoLog()` and `DoLogString()` in it. The second one is enough if you're happy with the standard `wxLog` message formatting (prepending "Error:" or "Warning:", timestamping &c) but just want to send the messages somewhere else. The first one may be overridden to do whatever you want but you have to distinguish between the different message types yourself.

There are some predefined classes deriving from `wxLog` and which might be helpful to see how you can create a new log target class and, of course, may also be used without any change. There are:

- **wxLogStderr** This class logs messages to a *FILE \**, using `stderr` by default as its name suggests.
- **wxLogStream** This class has the same functionality as `wxLogStderr`, but uses `ostream` and `cerr` instead of *FILE \** and `stderr`.
- **wxLogGui** This is the standard log target for `wxWidgets` applications (it is used by default if you don't do anything) and provides the most reasonable handling of all

types of messages for given platform.

- **wxLogWindow** This log target provides a "log console" which collects all messages generated by the application and also passes them to the previous active log target. The log window frame has a menu allowing user to clear the log, close it completely or save all messages to file.
- **wxLogNull** The last log class is quite particular: it doesn't do anything. The objects of this class may be instantiated to (temporarily) suppress output of *wxLogXXX()* functions. As an example, trying to open a non-existing file will usually provoke an error message, but if for some reasons it is unwanted, just use this construction:

```
wxFile file;

// wxFile.Open() normally complains if file can't be opened, we don't
want it
{
    wxLogNull logNo;
    if ( !file.Open("bar") )
        ... process error ourselves ...
} // ~wxLogNull called, old log sink restored

wxLogMessage("..."); // ok
```

The log targets can also be combined: for example you may wish to redirect the messages somewhere else (for example, to a log file) but also process them as normally. For this the *wxLogChain* (p. 1025) and *wxLogPassThrough* (p. 1028) can be used.

## Debugging overview

Classes, functions and macros: *wxDebugContext* (p. 483), *wxObject* (p. 1148), *wxLog* (p. 1018), *Log functions* (p. 1971), *Debug macros* (p. 1979)

Various classes, functions and macros are provided in *wxWidgets* to help you debug your application. Most of these are only available if you compile both *wxWidgets*, your application and *all* libraries that use *wxWidgets* with the `__WXDEBUG__` symbol defined. You can also test the `__WXDEBUG__` symbol in your own applications to execute code that should be active only in debug mode.

### wxDebugContext

*wxDebugContext* (p. 483) is a class that never gets instantiated, but ties together various static functions and variables. It allows you to dump all objects to that stream, write statistics about object allocation, and check memory for errors.

It is good practice to define a *wxObject::Dump* (p. 1148) member function for each class you derive from a *wxWidgets* class, so that *wxDebugContext::Dump* (p. 483) can call it and give valuable information about the state of the application.

If you have difficulty tracking down a memory leak, recompile in debugging mode and call



`wxDebugContext::Dump` (p. 483) and `wxDebugContext::PrintStatistics` (p. 485) at appropriate places. They will tell you what objects have not yet been deleted, and what kinds of object they are. In fact, in debug mode `wxWidgets` will automatically detect memory leaks when your application is about to exit, and if there are any leaks, will give you information about the problem. (How much information depends on the operating system and compiler -- some systems don't allow all memory logging to be enabled). See the memcheck sample for example of usage.

For `wxDebugContext` to do its work, the *new* and *delete* operators for `wxObject` have been redefined to store extra information about dynamically allocated objects (but not statically declared objects). This slows down a debugging version of an application, but can find difficult-to-detect memory leaks (objects are not deallocated), overwrites (writing past the end of your object) and underwrites (writing to memory in front of the object).

If debugging mode is on and the symbols `wxUSE_GLOBAL_MEMORY_OPERATORS` and `wxUSE_DEBUG_NEW_ALWAYS` are set to 1 in `setup.h`, 'new' is defined to be:

```
#define new new(__FILE__, __LINE__)
```

All occurrences of 'new' in `wxWidgets` and your own application will use the overridden form of the operator with two extra arguments. This means that the debugging output (and error messages reporting memory problems) will tell you what file and on what line you allocated the object. Unfortunately not all compilers allow this definition to work properly, but most do.

### Debug macros

You should also use *debug macros* (p. 1979) as part of a 'defensive programming' strategy, scattering `wxASSERT`s liberally to test for problems in your code as early as possible. Forward thinking will save a surprising amount of time in the long run.

`wxASSERT` (p. 1980) is used to pop up an error message box when a condition is not true. You can also use `wxASSERT_MSG` (p. 1980) to supply your own helpful error message. For example:

```
void MyClass::MyFunction(wxObject* object)
{
    wxASSERT_MSG( (object != NULL), "object should not be NULL in
MyFunction!" );

    ...
};
```

The message box allows you to continue execution or abort the program. If you are running the application inside a debugger, you will be able to see exactly where the problem was.

### Logging functions

You can use the `wxLogDebug` (p. 1973) and `wxLogTrace` (p. 1974) functions to output debugging information in debug mode; it will do nothing for non-debugging code.

## **wxDebugContext overview**

*Debugging overview* (p. 2072)

Class: *wxDebugContext* (p. 483)

*wxDebugContext* is a class for performing various debugging and memory tracing operations.

This class has only static data and function members, and there should be no instances. Probably the most useful members are *SetFile* (for directing output to a file, instead of the default standard error or debugger output); *Dump* (for dumping the dynamically allocated objects) and *PrintStatistics* (for dumping information about allocation of objects). You can also call *Check* to check memory blocks for integrity.

Here's an example of use. The *SetCheckpoint* ensures that only the allocations done after the checkpoint will be dumped.

```
wxDebugContext::SetCheckpoint();  
  
wxDebugContext::SetFile("c:\\temp\\debug.log");  
  
wxString *thing = new wxString;  
  
char *ordinaryNonObject = new char[1000];  
  
wxDebugContext::Dump();  
wxDebugContext::PrintStatistics();
```

You can use *wxDebugContext* if `__WXDEBUG__` is defined, or you can use it at any other time (if `wxUSE_DEBUG_CONTEXT` is set to 1 in `setup.h`). It is not disabled in non-debug mode because you may not wish to recompile *wxWidgets* and your entire application just to make use of the error logging facility.

Note: *wxDebugContext::SetFile* has a problem at present, so use the default stream instead. Eventually the logging will be done through the *wxLog* facilities instead.

## **wxConfig classes overview**

Classes: *wxConfig* (p. 262)

This overview briefly describes what the config classes are and what they are for. All the details about how to use them may be found in the description of the *wxConfigBase* (p. 262) class and the documentation of the file, registry and INI file based implementations mentions all the features/limitations specific to each one of these versions.

The config classes provide a way to store some application configuration information. They were especially designed for this usage and, although may probably be used for many other things as well, should be limited to it. It means that this information should be:

1. Typed, i.e. strings or numbers for the moment. You can not store binary data, for example.

2. Small. For instance, it is not recommended to use the Windows registry for amounts of data more than a couple of kilobytes.
3. Not performance critical, neither from speed nor from a memory consumption point of view.

On the other hand, the features provided make them very useful for storing all kinds of small to medium volumes of hierarchically-organized, heterogeneous data. In short, this is a place where you can conveniently stuff all your data (numbers and strings) organizing it in a tree where you use the filesystem-like paths to specify the location of a piece of data. In particular, these classes were designed to be as easy to use as possible.

From another point of view, they provide an interface which hides the differences between the Windows registry and the standard Unix text format configuration files. Other (future) implementations of `wxConfigBase` might also understand GTK resource files or their analogues on the KDE side.

In any case, each implementation of `wxConfigBase` does its best to make the data look the same way everywhere. Due to limitations of the underlying physical storage, it may not implement 100% of the base class functionality.

There are groups of entries and the entries themselves. Each entry contains either a string or a number (or a boolean value; support for other types of data such as dates or timestamps is planned) and is identified by the full path to it: something like `/MyApp/UserPreferences/Colors/Foreground`. The previous elements in the path are the group names, and each name may contain an arbitrary number of entries and subgroups. The path components are **always** separated with a slash, even though some implementations use the backslash internally. Further details (including how to read/write these entries) may be found in the documentation for `wxConfigBase` (p. 262).

## wxFileSystem

The `wxHTML` library uses a **virtual file systems** mechanism similar to the one used in Midnight Commander, Dos Navigator, FAR or almost any modern file manager. It allows the user to access data stored in archives as if they were ordinary files. On-the-fly generated files that exist only in memory are also supported.

### Classes

Three classes are used in order to provide virtual file systems mechanism:

- The `wxFsFile` (p. 692) class provides information about opened file (name, input stream, mime type and anchor).
- The `wxFileSystem` (p. 633) class is the interface. Its main methods are `ChangePathTo()` and `OpenFile()`. This class is most often used by the end user.
- The `wxFileSystemHandler` (p. 636) is the core of virtual file systems mechanism. You can derive your own handler and pass it to the VFS mechanism. You can derive your own handler and pass it to `wxFileSystem`'s `AddHandler()` method. In the new handler you only need to override the `OpenFile()` and `CanOpen()` methods.

## Locations

Locations (aka filenames aka addresses) are constructed from four parts:

- **protocol** - handler can recognize if it is able to open a file by checking its protocol. Examples are "http", "file" or "ftp".
- **right location** - is the name of file within the protocol. In "http://www.wxwidgets.org/index.html" the right location is "//www.wxwidgets.org/index.html".
- **anchor** - an anchor is optional and is usually not present. In "index.htm#chapter2" the anchor is "chapter2".
- **left location** - this is usually an empty string. It is used by 'local' protocols such as ZIP. See Combined Protocols paragraph for details.

## Combined Protocols

The left location precedes the protocol in the URL string. It is not used by global protocols like HTTP but it becomes handy when nesting protocols - for example you may want to access files in a ZIP archive:

```
file:archives/cpp_doc.zip#zip:reference/fopen.htm#syntax
```

In this example, the protocol is "zip", right location is "reference/fopen.htm", anchor is "syntax" and left location is "file:archives/cpp\_doc.zip".

There are **two** protocols used in this example: "zip" and "file".

## File Systems Included in wxHTML

The following virtual file system handlers are part of wxWidgets so far:

<b>wxArchiveFSHandler</b>	A handler for archives such as zip and tar. Include file is <wx/fs_arc.h>. URLs examples: "archive.zip#zip:filename", "archive.tar.gz#gzip:#tar:filename".
<b>wxFilterFSHandler</b>	A handler for compression schemes such as gzip. Header is <wx/fs_filter.h>. URLs are in the form, e.g.: "document.ps.gz#gzip:".
<b>wxInternetFSHandler</b>	A handler for accessing documents via HTTP or FTP protocols. Include file is <wx/fs_inet.h>.
<b>wxMemoryFSHandler</b>	This handler allows you to access data stored in memory (such as bitmaps) as if they were regular files. See <i>wxMemoryFSHandler documentation</i> (p. 1071) for details. Include file is <wx/fs_mem.h>. URL is prefixed with memory:, e.g. "memory:myfile.htm"

In addition, wxFileSystem itself can access local files.

## Initializing file system handlers

Use `wxFileSystem::AddHandler` (p. 634) to initialize a handler, for example:

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

...

bool MyApp::OnInit()
{
    wxFileSystem::AddHandler(new wxMemoryFSHandler);
    ...
}
```

## Event handling overview

Classes: `wxEvtHandler` (p. 576), `wxWindow` (p. 1795), `wxEvent` (p. 572)

### Introduction

Before version 2.0 of wxWidgets, events were handled by the application either by supplying callback functions, or by overriding virtual member functions such as **OnSize**.

From wxWidgets 2.0, *event tables* are used instead, with a few exceptions.

An event table is placed in an implementation file to tell wxWidgets how to map events to member functions. These member functions are not virtual functions, but they are all similar in form: they take a single `wxEvent`-derived argument, and have a void return type.

Here's an example of an event table.

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU      (wxID_EXIT, MyFrame::OnExit)
    EVT_MENU      (DO_TEST,  MyFrame::DoTest)
    EVT_SIZE      (          MyFrame::OnSize)
    EVT_BUTTON    (BUTTON1,   MyFrame::OnButton1)
END_EVENT_TABLE()
```

The first two entries map menu commands to two different member functions. The `EVT_SIZE` macro doesn't need a window identifier, since normally you are only interested in the current window's size events.

The `EVT_BUTTON` macro demonstrates that the originating event does not have to come from the window class implementing the event table -- if the event source is a button within a panel within a frame, this will still work, because event tables are searched up through the hierarchy of windows for the command events. In this case, the button's event table will be searched, then the parent panel's, then the frame's.

As mentioned before, the member functions that handle events do not have to be virtual. Indeed, the member functions should not be virtual as the event handler ignores that the functions are virtual, i.e. overriding a virtual member function in a derived class will not have any effect. These member functions take an event argument, and the class of event

differs according to the type of event and the class of the originating window. For size events, *wxSizeEvent* (p. 1443) is used. For menu commands and most control commands (such as button presses), *wxCommandEvent* (p. 250) is used. When controls get more complicated, then specific event classes are used, such as *wxTreeEvent* (p. 1748) for events from *wxTreeCtrl* (p. 1728) windows.

As well as the event table in the implementation file, there must also be a `DECLARE_EVENT_TABLE` macro somewhere in the class declaration. For example:

```
class MyFrame : public wxFrame
{
public:
    ...
    void OnExit(wxCommandEvent& event);
    void OnSize(wxSizeEvent& event);

protected:
    int      m_count;
    ...

    DECLARE_EVENT_TABLE()
};
```

Note that this macro may occur in any section of the class (public, protected or private) but that it is probably better to insert it at the end, as shown, because this macro implicitly changes the access to protected which may be quite unexpected if there is anything following it.

Finally, if you don't like using macros for static initialization of the event tables you may also use *wxEvtHandler::Connect* (p. 577) to connect the events to the handlers dynamically, during run-time. See the *event sample* (p. 2034) for an example of doing it.

## How events are processed

When an event is received from the windowing system, *wxWidgets* calls *wxEvtHandler::ProcessEvent* (p. 580) on the first event handler object belonging to the window generating the event.

It may be noted that *wxWidgets*' event processing system implements something very close to virtual methods in normal C++, i.e. it is possible to alter the behaviour of a class by overriding its event handling functions. In many cases this works even for changing the behaviour of native controls. For example it is possible to filter out a number of key events sent by the system to a native text control by overriding *wxTextCtrl* and defining a handler for key events using `EVT_KEY_DOWN`. This would indeed prevent any key events from being sent to the native control - which might not be what is desired. In this case the event handler function has to call *Skip()* so as to indicate that the search for the event handler should continue.

To summarize, instead of explicitly calling the base class version as you would have done with C++ virtual functions (i.e. *wxTextCtrl::OnChar()*), you should instead call *Skip* (p. 575).

In practice, this would look like this if the derived text control only accepts 'a' to 'z' and 'A' to

'Z':

```
void MyTextCtrl::OnChar(wxKeyEvent& event)
{
    if ( isalpha( event.KeyCode() ) )
    {
        // key code is within legal range. we call event.Skip() so the
        // event can be processed either in the base wxWidgets class
        // or the native control.

        event.Skip();
    }
    else
    {
        // illegal key hit. we don't call event.Skip() so the
        // event is not processed anywhere else.

        wxBell();
    }
}
```

The normal order of event table searching by `ProcessEvent` is as follows:

1. If the object is disabled (via a call to `wxEvtHandler::SetEvtHandlerEnabled` (p. 583)) the function skips to step (6).
2. If the object is a `wxWindow`, **ProcessEvent** is recursively called on the window's `wxValidator` (p. 1767). If this returns true, the function exits.
3. **SearchEventTable** is called for this event handler. If this fails, the base class table is tried, and so on until no more tables exist or an appropriate function was found, in which case the function exits.
4. The search is applied down the entire chain of event handlers (usually the chain has a length of one). If this succeeds, the function exits.
5. If the object is a `wxWindow` and the event is set to propagate (in the library only `wxCommandEvent` based events are set to propagate), **ProcessEvent** is recursively applied to the parent window's event handler. If this returns true, the function exits.
6. Finally, **ProcessEvent** is called on the `wxApp` object.

**Pay close attention to Step 5.** People often overlook or get confused by this powerful feature of the `wxWidgets` event processing system. To put it a different way, events set to propagate (See: `wxEvent::ShouldPropagate` (p. 575)) (most likely derived either directly or indirectly from `wxCommandEvent`) will travel up the containment hierarchy from child to parent until the maximal propagation level is reached or an event handler is found that doesn't call `event.Skip()` (p. 575).

Finally, there is another additional complication (which, in fact, simplifies life of `wxWidgets` programmers significantly): when propagating the command events upwards to the parent window, the event propagation stops when it reaches the parent dialog, if any. This means that you don't risk to get unexpected events from the dialog controls (which might be left

unprocessed by the dialog itself because it doesn't care about them) when a modal dialog is popped up. The events do propagate beyond the frames, however. The rationale for this choice is that there are only a few frames in a typical application and their parent-child relation are well understood by the programmer while it may be very difficult, if not impossible, to track down all the dialogs which may be popped up in a complex program (remember that some are created automatically by wxWidgets). If you need to specify a different behaviour for some reason, you can use *SetExtraStyle(wxWS\_EX\_BLOCK\_EVENTS)* (p. 1839) explicitly to prevent the events from being propagated beyond the given window or unset this flag for the dialogs which have it on by default.

Typically events that deal with a window as a window (size, motion, paint, mouse, keyboard, etc.) are sent only to the window. Events that have a higher level of meaning and/or are generated by the window itself, (button click, menu select, tree expand, etc.) are command events and are sent up to the parent to see if it is interested in the event.

Note that your application may wish to override *ProcessEvent* to redirect processing of events. This is done in the document/view framework, for example, to allow event handlers to be defined in the document or view. To test for command events (which will probably be the only events you wish to redirect), you may use *wxEvt::IsCommandEvent* (p. 574) for efficiency, instead of using the slower run-time type system.

As mentioned above, only command events are recursively applied to the parents event handler in the library itself. As this quite often causes confusion for users, here is a list of system events which will NOT get sent to the parent's event handler:

<i>wxEvt</i> (p. 572)	The event base class
<i>wxActivateEvent</i> (p. 33)	A window or application activation event
<i>wxCloseEvent</i> (p. 200)	A close window or end session event
<i>wxEraseEvent</i> (p. 571)	An erase background event
<i>wxFocusEvent</i> (p. 655)	A window focus event
<i>wxKeyEvent</i> (p. 956)	A keypress event
<i>wxIdleEvent</i> (p. 903)	An idle event
<i>wxInitDialogEvent</i> (p. 941)	A dialog initialisation event
<i>wxJoystickEvent</i> (p. 954)	A joystick event
<i>wxMenuEvent</i> (p. 1098)	A menu event
<i>wxMouseEvent</i> (p. 1119)	A mouse event
<i>wxMoveEvent</i> (p. 1128)	A move event
<i>wxPaintEvent</i> (p. 1164)	A paint event
<i>wxQueryLayoutInfoEvent</i> (p. 1238)	Used to query layout information



<i>wxSetCursorEvent</i> (p. 1429)	Used for special cursor processing based on current mouse position
<i>wxSizeEvent</i> (p. 1443)	A size event
<i>wxScrollWinEvent</i> (p. 1426)	A scroll event sent by a scrolled window (not a scroll bar)
<i>wxSysColourChangedEvent</i> (p. 1590)	A system colour change event

In some cases, it might be desired by the programmer to get a certain number of system events in a parent window, for example all key events sent to, but not used by, the native controls in a dialog. In this case, a special event handler will have to be written that will override *ProcessEvent()* in order to pass all events (or any selection of them) to the parent window.

### Events generated by the user vs programmatically generated events

While generically *wxEvents* (p. 572) can be generated both by user actions (e.g. resize of a *wxWindow* (p. 1795)) and by calls to functions (e.g. *wxWindow::SetSize* (p. 1845)), *wxWidgets* controls normally send *wxCommandEvent* (p. 250)-derived events only for the user-generated events. The only **exceptions** to this rule are:

<i>wxNotebook::AddPage</i> (p. 1138)	No event-free alternatives
<i>wxNotebook::AdvanceSelection</i> (p. 1139)	No event-free alternatives
<i>wxNotebook::DeletePage</i> (p. 1139)	No event-free alternatives
<i>wxNotebook::SetSelection</i> (p. 1143)	Use <i>wxNotebook::ChangeSelection</i> (p. 1144) instead, as <i>wxNotebook::SetSelection</i> (p. 1143) is deprecated
<i>wxTreeCtrl::Delete</i> (p. 1734)	No event-free alternatives
<i>wxTreeCtrl::DeleteAllItems</i> (p. 1734)	No event-free alternatives
<i>wxTreeCtrl::EditLabel</i> (p. 1734)	No event-free alternatives
All <i>wxTextCtrl</i> (p. 1633) methods	<i>wxTextCtrl::ChangeValue</i> (p. 1651) can be used instead of <i>wxTextCtrl::SetValue</i> (p. 1651) but the other functions, such as <i>Replace</i> (p. 1647) or <i>WriteText</i> (p. 1652) don't have event-free equivalents

### Pluggable event handlers

In fact, you don't have to derive a new class from a window class if you don't want to. You can derive a new class from *wxEvtHandler* instead, defining the appropriate event table, and then call *wxWindow::SetEventHandler* (p. 1838) (or, preferably, *wxWindow::PushEventHandler* (p. 1829)) to make this event handler the object that responds to events. This way, you can avoid a lot of class derivation, and use instances of the same event handler class (but different objects as the same event handler object

shouldn't be used more than once) to handle events from instances of different widget classes. If you ever have to call a window's event handler manually, use the `GetEventHandler` function to retrieve the window's event handler and use that to call the member function. By default, `GetEventHandler` returns a pointer to the window itself unless an application has redirected event handling using `SetEventHandler` or `PushEventHandler`.

One use of `PushEventHandler` is to temporarily or permanently change the behaviour of the GUI. For example, you might want to invoke a dialog editor in your application that changes aspects of dialog boxes. You can grab all the input for an existing dialog box, and edit it 'in situ', before restoring its behaviour to normal. So even if the application has derived new classes to customize behaviour, your utility can indulge in a spot of body-snatching. It could be a useful technique for on-line tutorials, too, where you take a user through a series of steps and don't want them to diverge from the lesson. Here, you can examine the events coming from buttons and windows, and if acceptable, pass them through to the original event handler. Use `PushEventHandler/PopEventHandler` to form a chain of event handlers, where each handler processes a different range of events independently from the other handlers.

## Window identifiers

Window identifiers are integers, and are used to uniquely determine window identity in the event system (though you can use it for other purposes). In fact, identifiers do not need to be unique across your entire application just so long as they are unique within a particular context you're interested in, such as a frame and its children. You may use the `wxID_OK` identifier, for example, on any number of dialogs so long as you don't have several within the same dialog.

If you pass `wxID_ANY` to a window constructor, an identifier will be generated for you automatically by `wxWidgets`. This is useful when you don't care about the exact identifier either because you're not going to process the events from the control being created at all or because you process the events from all controls in one place (in which case you should specify `wxID_ANY` in the event table or `wxEvtHandler::Connect` (p. 577) call as well. The automatically generated identifiers are always negative and so will never conflict with the user-specified identifiers which must be always positive.

The following standard identifiers are supplied. You can use `wxID_HIGHEST` to determine the number above which it is safe to define your own identifiers. Or, you can use identifiers below `wxID_LOWEST`.

```
#define wxID_ANY          -1

#define wxID_LOWEST      4999

#define wxID_OPEN        5000
#define wxID_CLOSE       5001
#define wxID_NEW          5002
#define wxID_SAVE         5003
#define wxID_SAVEAS       5004
#define wxID_REVERT       5005
#define wxID_EXIT         5006
#define wxID_UNDO         5007
```

```
#define wxID_REDO                5008
#define wxID_HELP                5009
#define wxID_PRINT               5010
#define wxID_PRINT_SETUP        5011
#define wxID_PREVIEW            5012
#define wxID_ABOUT              5013
#define wxID_HELP_CONTENTS      5014
#define wxID_HELP_COMMANDS     5015
#define wxID_HELP_PROCEDURES    5016
#define wxID_HELP_CONTEXT       5017

#define wxID_CUT                 5030
#define wxID_COPY                5031
#define wxID_PASTE               5032
#define wxID_CLEAR               5033
#define wxID_FIND                5034
#define wxID_DUPLICATE           5035
#define wxID_SELECTALL           5036
#define wxID_DELETE              5037
#define wxID_REPLACE             5038
#define wxID_REPLACE_ALL         5039
#define wxID_PROPERTIES          5040

#define wxID_VIEW_DETAILS        5041
#define wxID_VIEW_LARGEICONS    5042
#define wxID_VIEW_SMALLICONS    5043
#define wxID_VIEW_LIST           5044
#define wxID_VIEW_SORTDATE       5045
#define wxID_VIEW_SORTNAME       5046
#define wxID_VIEW_SORTSIZE       5047
#define wxID_VIEW_SORTTYPE       5048

#define wxID_FILE1               5050
#define wxID_FILE2               5051
#define wxID_FILE3               5052
#define wxID_FILE4               5053
#define wxID_FILE5               5054
#define wxID_FILE6               5055
#define wxID_FILE7               5056
#define wxID_FILE8               5057
#define wxID_FILE9               5058

#define wxID_OK                  5100
#define wxID_CANCEL              5101
#define wxID_APPLY               5102
#define wxID_YES                 5103
#define wxID_NO                  5104
#define wxID_STATIC              5105

#define wxID_HIGHEST             5999
```

## Event macros summary

### Macros listed by event class

The documentation for specific event macros is organised by event class. Please refer to

these sections for details.

<i>wxActivateEvent</i> (p. 33)	The EVT_ACTIVATE and EVT_ACTIVATE_APP macros intercept activation and deactivation events.
<i>wxCommandEvent</i> (p. 250)	A range of commonly-used control events.
<i>wxCloseEvent</i> (p. 200)	The EVT_CLOSE macro handles window closure called via <i>wxWindow::Close</i> (p. 1802).
<i>wxDropFilesEvent</i> (p. 557)	The EVT_DROP_FILES macros handles file drop events.
<i>wxEraseEvent</i> (p. 571)	The EVT_ERASE_BACKGROUND macro is used to handle window erase requests.
<i>wxFocusEvent</i> (p. 655)	The EVT_SET_FOCUS and EVT_KILL_FOCUS macros are used to handle keyboard focus events.
<i>wxKeyEvent</i> (p. 956)	EVT_CHAR, EVT_KEY_DOWN and EVT_KEY_UP macros handle keyboard input for any window.
<i>wxIdleEvent</i> (p. 903)	The EVT_IDLE macro handle application idle events (to process background tasks, for example).
<i>wxInitDialogEvent</i> (p. 941)	The EVT_INIT_DIALOG macro is used to handle dialog initialisation.
<i>wxListEvent</i> (p. 1000)	These macros handle <i>wxListCtrl</i> (p. 980) events.
<i>wxMenuEvent</i> (p. 1098)	These macros handle special menu events (not menu commands).
<i>wxMouseEvent</i> (p. 1119)	Mouse event macros can handle either individual mouse events or all mouse events.
<i>wxMoveEvent</i> (p. 1128)	The EVT_MOVE macro is used to handle a window move.
<i>wxPaintEvent</i> (p. 1164)	The EVT_PAINT macro is used to handle window paint requests.
<i>wxScrollEvent</i> (p. 1423)	These macros are used to handle scroll events from <i>wxScrollBar</i> (p. 1408), <i>wxSlider</i> (p. 1462), and <i>wxSpinButton</i> (p. 1496).

<i>wxSetCursorEvent</i> (p. 1429)	The EVT_SET_CURSOR macro is used for special cursor processing.
<i>wxSizeEvent</i> (p. 1443)	The EVT_SIZE macro is used to handle a window resize.
<i>wxSplitterEvent</i> (p. 1505)	The EVT_SPLITTER_SASH_POS_CHANGED, EVT_SPLITTER_UNSPLOT and EVT_SPLITTER_DCLICK macros are used to handle the various splitter window events.
<i>wxSysColourChangedEvent</i> (p. 1590)	The EVT_SYS_COLOUR_CHANGED macro is used to handle events informing the application that the user has changed the system colours (Windows only).
<i>wxTreeEvent</i> (p. 1748)	These macros handle <i>wxTreeCtrl</i> (p. 1728) events.
<i>wxUpdateUIEvent</i> (p. 1752)	The EVT_UPDATE_UI macro is used to handle user interface update pseudo-events, which are generated to give the application the chance to update the visual state of menus, toolbars and controls.

## Custom event summary

### General approach

Since version 2.2.x of wxWidgets, each event type is identified by ID which is given to the event type *at runtime* which makes it possible to add new event types to the library or application without risking ID clashes (two different event types mistakingly getting the same event ID). This event type ID is stored in a struct of type **const wxEventType**.

In order to define a new event type, there are principally two choices. One is to define a entirely new event class (typically deriving from *wxEvent* (p. 572) or *wxCommandEvent* (p. 250). The other is to use the existing event classes and give them an new event type. You'll have to define and declare a new event type using either way, and this is done using the following macros:

```
// in the header of the source file
BEGIN_DECLARE_EVENT_TYPES()
DECLARE_EVENT_TYPE(name, value)
END_DECLARE_EVENT_TYPES()

// in the implementation
DEFINE_EVENT_TYPE(name)
```

You can ignore the *value* parameter of the DECLARE\_EVENT\_TYPE macro since it used only for backwards compatibility with wxWidgets 2.0.x based applications where you have

to give the event type ID an explicit value.

See also the *event sample* (p. 2034) for an example of code defining and working with the custom event types.

### Using existing event classes

If you just want to use a *wxCommandEvent* (p. 250) with a new event type, you can then use one of the generic event table macros listed below, without having to define a new macro yourself. This also has the advantage that you won't have to define a new *wxEvent::Clone()* (p. 573) method for posting events between threads etc. This could look like this in your code:

```
DECLARE_EVENT_TYPE(wxEVT_MY_EVENT, -1)

DEFINE_EVENT_TYPE(wxEVT_MY_EVENT)

// user code intercepting the event

BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU      (wxID_EXIT, MyFrame::OnExit)
    // ....
    EVT_COMMAND   (ID_MY_WINDOW, wxEVT_MY_EVENT, MyFrame::OnMyEvent)
END_EVENT_TABLE()

void MyFrame::OnMyEvent( wxCommandEvent &event )
{
    // do something
    wxString text = event.GetText();
}

// user code sending the event

void MyWindow::SendEvent()
{
    wxCommandEvent event( wxEVT_MY_EVENT, GetId() );
    event.SetEventObject( this );
    // Give it some contents
    event.SetText( wxT("Hallo") );
    // Send it
    GetEventHandler()->ProcessEvent( event );
}
```

### Generic event table macros

<b>EVT_CUSTOM(event, id, func)</b>	Allows you to add a custom event table entry by specifying the event identifier (such as <code>wxEVT_SIZE</code> ), the window identifier, and a member function to call.
<b>EVT_CUSTOM_RANGE(event, id1, id2, func)</b>	The same as <code>EVT_CUSTOM</code> , but responds to a range of window identifiers.
<b>EVT_COMMAND(id, event, func)</b>	The same as <code>EVT_CUSTOM</code> , but expects

a member function with a  
wxCommandEvent argument.

**EVT\_COMMAND\_RANGE(id1, id2, event, func)** The same as  
EVT\_CUSTOM\_RANGE, but expects a  
member function with a  
wxCommandEvent argument.

**EVT\_NOTIFY(event, id, func)** The same as EVT\_CUSTOM, but expects  
a member function with a wxNotifyEvent  
argument.

**EVT\_NOTIFY\_RANGE(event, id1, id2, func)** The same as EVT\_CUSTOM\_RANGE,  
but expects a member function with a  
wxNotifyEvent argument.

### Defining your own event class

Under certain circumstances, it will be required to define your own event class e.g. for sending more complex data from one place to another. Apart from defining your event class, you will also need to define your own event table macro (which is quite long). Watch out to put in enough casts to the inherited event function. Here is an example, taken mostly from the *wxPlot* library, which is in the *contrib* section of the wxWidgets sources.

```
// code defining event

class wxPlotEvent: public wxNotifyEvent
{
public:
    wxPlotEvent( wxEventType commandType = wxEVT_NULL, int id = 0 );

    // accessors
    wxPlotCurve *GetCurve()
        { return m_curve; }

    // required for sending with wxPostEvent()
    wxEvent* Clone();

private:
    wxPlotCurve    *m_curve;
};

DECLARE_EVENT_TYPE( wxEVT_PLOT_ACTION, -1 )

typedef void (wxEvtHandler::*wxPlotEventFunction)(wxPlotEvent&);

#define EVT_PLOT(id, fn) \
    DECLARE_EVENT_TABLE_ENTRY( wxEVT_PLOT_ACTION, id, -1, \
        (wxObjectEventFunction) (wxEventFunction) \
        (wxCommandEventFunction) (wxNotifyEventFunction) \
        wxStaticCastEvent( wxPlotEventFunction, &fn ), (wxObject *) \
        NULL ),

// code implementing the event type and the event class
```

```
DEFINE_EVENT_TYPE( wxEVT_PLOT_ACTION )

wxPlotEvent::wxPlotEvent( ...

// user code intercepting the event

BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_PLOT (ID_MY_WINDOW, MyFrame::OnPlot)
END_EVENT_TABLE()

void MyFrame::OnPlot( wxPlotEvent &event )
{
    wxPlotCurve *curve = event.GetCurve();
}

// user code sending the event

void MyWindow::SendEvent()
{
    wxPlotEvent event( wxEVT_PLOT_ACTION, GetId() );
    event.SetEventObject( this );
    event.SetCurve( m_curve );
    GetEventHandler()->ProcessEvent( event );
}
```

## C++ exceptions overview

### Introduction

wxWidgets had been started long before the exceptions were introduced in C++ so it is not very surprising that it is not built around using them as some more modern C++ libraries are. For instance, the library doesn't throw exceptions to signal about the errors. Moreover, up to (and including) the version 2.4 of wxWidgets, even using the exceptions in the user code was dangerous because the library code wasn't exception-safe and so an exception propagating through it could result in memory and/or resource leaks, and also not very convenient.

Starting from the version 2.5.1 wxWidgets becomes more exception-friendly. It still doesn't use the exceptions by itself but it should be now safe to use the exceptions in the user code and the library tries to help you with this. Please note that making the library exception-safe is still work in progress.

### Strategies for exceptions handling

There are several choice for using the exceptions in wxWidgets programs. First of all, you may not use them at all. As stated above, the library doesn't throw any exceptions by itself and so you don't have to worry about exceptions at all unless your own code throws them.



This is, of course, the simplest solution but may be not the best one to deal with all possible errors.

Another strategy is to use exceptions only to signal truly fatal errors. In this case you probably don't expect to recover from them and the default behaviour -- to simply terminate the program -- may be appropriate. If it is not, you may override *OnUnhandledException()* (p. 52) in your *wxApp*-derived class to perform any clean up tasks. Note, however, that any information about the exact exception type is lost when this function is called, so if you need you should override *OnRun()* (p. 52) and add a try/catch clause around the call of the base class version. This would allow you to catch any exceptions generated during the execution of the main event loop. To deal with the exceptions which may arise during the program startup and/or shutdown you should insert try/catch clauses in *OnInit()* (p. 52) and/or *OnExit()* (p. 51) as well.

Finally, you may also want to continue running even when certain exceptions occur. If all of your exceptions may happen only in the event handlers of a single class (or only in the classes derived from it), you may centralize your exception handling code in *ProcessEvent* (p. 580) method of this class. If this is impractical, you may also consider overriding the *wxApp::HandleEvent()* (p. 55) which allows you to handle all the exceptions thrown by any event handler.

## Technicalities

To use any kind of exception support in the library you need to build it with `wxUSE_EXCEPTIONS` set to 1. This should be the case by default but if it isn't, you should edit the `include/wx/msw/setup.h` file under Windows or run `configure` with `--enable-exceptions` argument under Unix.

On the other hand, if you do *not* plan to use exceptions, setting this flag to 0 or using `--disable-exceptions` could result in a leaner and slightly faster library.

As for any other library feature, there is a *sample* (p. 2034) showing how to use it. Please look at its sources for further information.

## Window styles

Window styles are used to specify alternative behaviour and appearances for windows, when they are created. The symbols are defined in such a way that they can be combined in a 'bit-list' using the C++ *bitwise-or* operator. For example:

```
wxCAPTION | wxMINIMIZE_BOX | wxMAXIMIZE_BOX | wxRESIZE_BORDER
```

For the window styles specific to each window class, please see the documentation for the window. Most windows can use the generic styles listed for *wxWindow* (p. 1795) in addition to their own styles.

## Window deletion overview

Classes: *wxCloseEvent* (p. 200), *wxWindow* (p. 1795)

Window deletion can be a confusing subject, so this overview is provided to help make it clear when and how you delete windows, or respond to user requests to close windows.

### **What is the sequence of events in a window deletion?**

When the user clicks on the system close button or system close command, in a frame or a dialog, wxWidgets calls `wxWindow::Close` (p. 1802). This in turn generates an EVT\_CLOSE event: see `wxCloseEvent` (p. 200).

It is the duty of the application to define a suitable event handler, and decide whether or not to destroy the window. If the application is for some reason forcing the application to close (`wxCloseEvent::CanVeto` (p. 201) returns false), the window should always be destroyed, otherwise there is the option to ignore the request, or maybe wait until the user has answered a question before deciding whether it is safe to close. The handler for EVT\_CLOSE should signal to the calling code if it does not destroy the window, by calling `wxCloseEvent::Veto` (p. 201). Calling this provides useful information to the calling code.

The `wxCloseEvent` handler should only call `wxWindow::Destroy` (p. 1804) to delete the window, and not use the **delete** operator. This is because for some window classes, wxWidgets delays actual deletion of the window until all events have been processed, since otherwise there is the danger that events will be sent to a non-existent window.

As reinforced in the next section, calling `Close` does not guarantee that the window will be destroyed. Call `wxWindow::Destroy` (p. 1804) if you want to be certain that the window is destroyed.

### **How can the application close a window itself?**

Your application can either use `wxWindow::Close` (p. 1802) event just as the framework does, or it can call `wxWindow::Destroy` (p. 1804) directly. If using `Close()`, you can pass a true argument to this function to tell the event handler that we definitely want to delete the frame and it cannot be vetoed.

The advantage of using `Close` instead of `Destroy` is that it will call any clean-up code defined by the EVT\_CLOSE handler; for example it may close a document contained in a window after first asking the user whether the work should be saved. `Close` can be vetoed by this process (return false), whereas `Destroy` definitely destroys the window.

### **What is the default behaviour?**

The default close event handler for `wxDialog` simulates a Cancel command, generating a `wxID_CANCEL` event. Since the handler for this cancel event might itself call **Close**, there is a check for infinite looping. The default handler for `wxID_CANCEL` hides the dialog (if modeless) or calls `EndModal(wxID_CANCEL)` (if modal). In other words, by default, the dialog *is not destroyed* (it might have been created on the stack, so the assumption of dynamic creation cannot be made).

The default close event handler for `wxFrame` destroys the frame using `Destroy()`.

### **What should I do when the user calls up Exit from a menu?**

You can simply call `wxWindow::Close` (p. 1802) on the frame. This will invoke your own close event handler which may destroy the frame.

You can do checking to see if your application can be safely exited at this point, either from within your close event handler, or from within your exit menu command handler. For example, you may wish to check that all files have been saved. Give the user a chance to save and quit, to not save but quit anyway, or to cancel the exit command altogether.

### What should I do to upgrade my 1.xx OnClose to 2.0?

In wxWidgets 1.xx, the **OnClose** function did not actually delete 'this', but signaled to the calling function (either **Close**, or the wxWidgets framework) to delete or not delete the window.

To update your code, you should provide an event table entry in your frame or dialog, using the EVT\_CLOSE macro. The event handler function might look like this:

```
void MyFrame::OnCloseWindow(wxCloseEvent& event)
{
    if (MyDataHasBeenModified())
    {
        wxMessageDialog* dialog = new wxMessageDialog(this,
            "Save changed data?", "My app", wxYES_NO|wxCANCEL);

        int ans = dialog->ShowModal();
        dialog->Destroy();

        switch (ans)
        {
            case wxID_YES:          // Save, then destroy, quitting app
                SaveMyData();
                this->Destroy();
                break;
            case wxID_NO:           // Don't save; just destroy, quitting app
                this->Destroy();
                break;
            case wxID_CANCEL:       // Do nothing - so don't quit app.
            default:
                if (!event.CanVeto()) // Test if we can veto this deletion
                    this->Destroy(); // If not, destroy the window anyway.
                else
                    event.Veto();    // Notify the calling code that we didn't
delete the frame.
                break;
        }
    }
}
```

### How do I exit the application gracefully?

A wxWidgets application automatically exits when the last top level window (*wxFrame* (p. 682) or *wxDialog* (p. 496)), is destroyed. Put any application-wide cleanup code in *wxApp::OnExit* (p. 51) (this is a virtual function, not an event handler).

### Do child windows get deleted automatically?

Yes, child windows are deleted from within the parent destructor. This includes any children that are themselves frames or dialogs, so you may wish to close these child frame

or dialog windows explicitly from within the parent close handler.

### What about other kinds of window?

So far we've been talking about 'managed' windows, i.e. frames and dialogs. Windows with parents, such as controls, don't have delayed destruction and don't usually have close event handlers, though you can implement them if you wish. For consistency, continue to use the `wxWindow::Destroy` (p. 1804) function instead of the **delete** operator when deleting these kinds of windows explicitly.

## wxDialog overview

Classes: `wxDialog` (p. 496)

A dialog box is similar to a panel, in that it is a window which can be used for placing controls, with the following exceptions:

1. A surrounding frame is implicitly created.
2. Extra functionality is automatically given to the dialog box, such as tabbing between items (currently Windows only).
3. If the dialog box is *modal*, the calling program is blocked until the dialog box is dismissed.

Under Windows 3, modal dialogs have to be emulated using modeless dialogs and a message loop. This is because Windows 3 expects the contents of a modal dialog to be loaded from a resource file or created on receipt of a dialog initialization message. This is too restrictive for `wxWidgets`, where any window may be created and displayed before its contents are created.

For a set of dialog convenience functions, including file selection, see *Dialog functions* (p. 1934).

See also `wxPanel` (p. 1170) and `wxWindow` (p. 1795) for inherited member functions. Validation of data in controls is covered in *Validator overview* (p. 2092).

## wxValidator overview

Classes: `wxValidator` (p. 1767), `wxTextValidator` (p. 1668), `wxGenericValidator` (p. 714)

The aim of the validator concept is to make dialogs very much easier to write. A validator is an object that can be plugged into a control (such as a `wxTextCtrl`), and mediates between C++ data and the control, transferring the data in either direction and validating it. It also is able to intercept events generated by the control, providing filtering behaviour without the need to derive a new control class.

You can use a stock validator, such as `wxTextValidator` (p. 1668) (which does text control data transfer, validation and filtering) and `wxGenericValidator` (p. 714) (which does data transfer for a range of controls); or you can write your own.

## Example

Here is an example of `wxTextValidator` usage.

```
wxTextCtrl *txt1 = new wxTextCtrl(this, -1, wxT(""),
    wxPoint(10, 10), wxSize(100, 80), 0,
    wxTextValidator(wxFILTER_ALPHA, &g_data.m_string));
```

In this example, the text validator object provides the following functionality:

1. It transfers the value of `g_data.m_string` (a `wxString` variable) to the `wxTextCtrl` when the dialog is initialised.
2. It transfers the `wxTextCtrl` data back to this variable when the dialog is dismissed.
3. It filters input characters so that only alphabetic characters are allowed.

The validation and filtering of input is accomplished in two ways. When a character is input, `wxTextValidator` checks the character against the allowed filter flag (`wxFILTER_ALPHA` in this case). If the character is inappropriate, it is vetoed (does not appear) and a warning beep sounds. The second type of validation is performed when the dialog is about to be dismissed, so if the default string contained invalid characters already, a dialog box is shown giving the error, and the dialog is not dismissed.

## Anatomy of a validator

A programmer creating a new validator class should provide the following functionality.

A validator constructor is responsible for allowing the programmer to specify the kind of validation required, and perhaps a pointer to a C++ variable that is used for storing the data for the control. If such a variable address is not supplied by the user, then the validator should store the data internally.

The `wxValidator::Validate` (p. 1769) member function should return true if the data in the control (not the C++ variable) is valid. It should also show an appropriate message if data was not valid.

The `wxValidator::TransferToWindow` (p. 1769) member function should transfer the data from the validator or associated C++ variable to the control.

The `wxValidator::TransferFromWindow` (p. 1769) member function should transfer the data from the control to the validator or associated C++ variable.

There should be a copy constructor, and a `wxValidator::Clone` (p. 1768) function which returns a copy of the validator object. This is important because validators are passed by reference to window constructors, and must therefore be cloned internally.

You can optionally define event handlers for the validator, to implement filtering. These handlers will capture events before the control itself does.

For an example implementation, see the `valtext.h` and `valtext.cpp` files in the `wxWidgets` library.

## How validators interact with dialogs

For validators to work correctly, validator functions must be called at the right times during dialog initialisation and dismissal.

When a `wxDialog::Show` (p. 504) is called (for a modeless dialog) or `wxDialog::ShowModal` (p. 505) is called (for a modal dialog), the function `wxWindow::InitDialog` (p. 1823) is automatically called. This in turn sends an initialisation event to the dialog. The default handler for the `wxEVT_INIT_DIALOG` event is defined in the `wxWindow` class to simply call the function `wxWindow::TransferDataToWindow` (p. 1851). This function finds all the validators in the window's children and calls the `TransferToWindow` function for each. Thus, data is transferred from C++ variables to the dialog just as the dialog is being shown.

If you are using a window or panel instead of a dialog, you will need to call `wxWindow::InitDialog` (p. 1823) explicitly before showing the window.

When the user clicks on a button, for example the OK button, the application should first call `wxWindow::Validate` (p. 1853), which returns false if any of the child window validators failed to validate the window data. The button handler should return immediately if validation failed. Secondly, the application should call `wxWindow::TransferDataFromWindow` (p. 1851) and return if this failed. It is then safe to end the dialog by calling `EndModal` (if modal) or `Show` (if modeless).

In fact, `wxDialog` contains a default command event handler for the `wxID_OK` button. It goes like this:

```
void wxDialog::OnOK(wxCommandEvent& event)
{
    if ( Validate() && TransferDataFromWindow() )
    {
        if ( IsModal() )
            EndModal(wxID_OK);
        else
        {
            SetReturnCode(wxID_OK);
            this->Show(false);
        }
    }
}
```

So if using validators and a normal OK button, you may not even need to write any code for handling dialog dismissal.

If you load your dialog from a resource file, you will need to iterate through the controls setting validators, since validators can't be specified in a dialog resource.

## Constraints overview

Classes: `wxLayoutConstraints` (p. 964), `wxIndividualLayoutConstraint` (p. 938).

**Note:** constraints are now deprecated and you should use *sizers* (p. 2098) instead.

Objects of class `wxLayoutConstraint` can be associated with a window to define the way it is laid out, with respect to its siblings or the 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

The constraints are initially set to have the relationship `wxUnconstrained`, which means that their values should be calculated by looking at known constraints. To calculate the position and size of the control, the layout algorithm needs to know exactly 4 constraints (as it has 4 numbers to calculate from them), so you should always set exactly 4 of the constraints from the above table.

If you want the controls height or width to have the default value, you may use a special value for the constraint: `wxAsIs`. If the constraint is `wxAsIs`, the dimension will not be changed which is useful for the dialog controls which often have the default size (e.g. the buttons whose size is determined by their label).

The constraints calculation is done in `wxWindow::Layout` (p. 1825) function which evaluates constraints. To call it you can either call `wxWindow::SetAutoLayout` (p. 1834) if the parent window is a frame, panel or a dialog to tell default `OnSize` handlers to call `Layout` automatically whenever the window size changes, or override `OnSize` and call `Layout` yourself (note that you do have to call `Layout` (p. 1825) yourself if the parent window is not a frame, panel or dialog).

### Constraint layout: more details

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 exactly one subwindow (not counting toolbar and statusbar which are also positioned by the frame automatically), where `wxFrame::OnSize` takes care of resizing the child to always fill the frame.

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 `wxWidgets`, 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 wxWidgets 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, however you may reduce the number of iterations (and thus speed up the layout calculations) by creating the controls in such order that as many constraints as possible can be calculated during the first iteration. For example, if you have 2 buttons which you'd like to position in the lower right corner, it is slightly more efficient to first create the second button and specify that its right border is `IsSameAs(parent, wxRight)` and then create the first one by specifying that it should be `LeftOf()` the second one than to do in a more natural left-to-right order.

## Window layout examples

### Example 1: subwindow layout

This example specifies a panel and a window side by side, with a text subwindow below it.

```
frame->panel = new wxPanel(frame, -1, wxPoint(0, 0), wxSize(1000,
500), 0);
frame->scrollWindow = new MyScrolledWindow(frame, -1, wxPoint(0, 0),
wxSize(400, 400), wxRETAINED);
frame->text_window = new MyTextWindow(frame, -1, wxPoint(0, 250),
wxSize(400, 250));

// Set constraints for panel subwindow
wxLayoutConstraints *cl = new wxLayoutConstraints;

cl->left.SameAs      (frame, wxLeft);
cl->top.SameAs       (frame, wxTop);
cl->right.PercentOf  (frame, wxWidth, 50);
cl->height.PercentOf (frame, wxHeight, 50);
```



```
frame->panel->SetConstraints(c1);

// Set constraints for scrollWindow subwindow
wxLayoutConstraints *c2 = new wxLayoutConstraints;

c2->left.SameAs      (frame->panel, wxRight);
c2->top.SameAs        (frame, wxTop);
c2->right.SameAs      (frame, wxRight);
c2->height.PercentOf (frame, wxHeight, 50);

frame->scrollWindow->SetConstraints(c2);

// Set constraints for text subwindow
wxLayoutConstraints *c3 = new wxLayoutConstraints;
c3->left.SameAs      (frame, wxLeft);
c3->top.Below        (frame->panel);
c3->right.SameAs     (frame, wxRight);
c3->bottom.SameAs    (frame, wxBottom);

frame->text_window->SetConstraints(c3);
```

### Example 2: panel item layout

This example sizes a button width to 80 percent of the panel width, and centres it horizontally. A listbox and multitext item are placed below it. The listbox takes up 40 percent of the panel width, and the multitext item takes up the remainder of the width. Margins of 5 pixels are used.

```
// Create some panel items
wxButton *btn1 = new wxButton(frame->panel, -1, "A button") ;

wxLayoutConstraints *b1 = new wxLayoutConstraints;
b1->centreX.SameAs      (frame->panel, wxCentreX);
b1->top.SameAs           (frame->panel, wxTop, 5);
b1->width.PercentOf     (frame->panel, wxWidth, 80);
b1->height.PercentOf    (frame->panel, wxHeight, 10);
btn1->SetConstraints(b1);

wxListBox *list = new wxListBox(frame->panel, -1, "A list",
                                wxPoint(-1, -1), wxSize(200,
100));

wxLayoutConstraints *b2 = new wxLayoutConstraints;
b2->top.Below           (btn1, 5);
b2->left.SameAs         (frame->panel, wxLeft, 5);
b2->width.PercentOf     (frame->panel, wxWidth, 40);
b2->bottom.SameAs       (frame->panel, wxBottom, 5);
list->SetConstraints(b2);

wxTextCtrl *mtext = new wxTextCtrl(frame->panel, -1, "Multiline
text", "Some text",
                                wxPoint(-1, -1), wxSize(150, 100),
wxTE_MULTILINE);

wxLayoutConstraints *b3 = new wxLayoutConstraints;
```

```
b3->top.Below      (btn1, 5);  
b3->left.RightOf   (list, 5);  
b3->right.SameAs   (frame->panel, wxRight, 5);  
b3->bottom.SameAs  (frame->panel, wxBottom, 5);  
mtext->SetConstraints(b3);
```

## Sizer overview

Classes: *wxSizer* (p. 1444), *wxGridSizer* (p. 797), *wxFlexGridSizer* (p. 652), *wxBoxSizer* (p. 149), *wxStaticBoxSizer* (p. 1532), *CreateButtonSizer* (p. 2105)

Sizers, as represented by the *wxSizer* class and its descendants in the *wxWidgets* class hierarchy, have become the method of choice to define the layout of controls in dialogs in *wxWidgets* because of their ability to create visually appealing dialogs independent of the platform, taking into account the differences in size and style of the individual controls. Unlike the original *wxWidgets* Dialog Editor, editors such as *wxDesigner*, *DialogBlocks*, *XRCed* and *wxWorkshop* create dialogs based exclusively on sizers, practically forcing the user to create platform independent layouts without compromises.

The next section describes and shows what can be done with sizers. The following sections briefly describe how to program with individual sizer classes.

For information about the new *wxWidgets* resource system, which can describe sizer-based dialogs, see the *XML-based resource system overview* (p. 2106).

## The idea behind sizers

The layout algorithm used by sizers in *wxWidgets* is closely related to layout systems in other GUI toolkits, such as Java's AWT, the GTK toolkit or the Qt toolkit. It is based upon the idea of individual subwindows reporting their minimal required size and their ability to get stretched if the size of the parent window has changed. This will most often mean that the programmer does not set the start-up size of a dialog, the dialog will rather be assigned a sizer and this sizer will be queried about the recommended size. This sizer in turn will query its children (which can be normal windows, empty space or other sizers) so that a hierarchy of sizers can be constructed. Note that *wxSizer* does not derive from *wxWindow* and thus does not interfere with tab ordering and requires very few resources compared to a real window on screen.

What makes sizers so well fitted for use in *wxWidgets* is the fact that every control reports its own minimal size and the algorithm can handle differences in font sizes or different window (dialog item) sizes on different platforms without problems. For example, if the standard font as well as the overall design of Linux/GTK widgets requires more space than on Windows, the initial dialog size will automatically be bigger on Linux/GTK than on Windows.

There are currently five different kinds of sizers available in *wxWidgets*. Each represents either a certain way to lay out dialog items in a dialog or it fulfills a special task such as wrapping a static box around a dialog item (or another sizer). These sizers will be discussed one by one in the text below. For more detailed information on how to use sizers programmatically, please refer to the section *Programming with Sizers* (p. 2102).

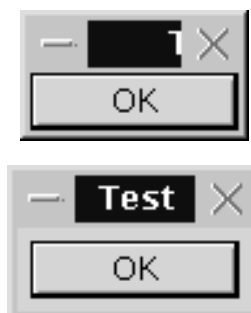
## Common features

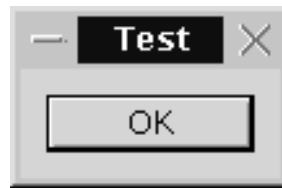
All sizers are containers, that is, they are used to lay out one dialog item (or several dialog items), which they contain. Such items are sometimes referred to as the children of the sizer. Independent of how the individual sizers lay out their children, all children have certain features in common:

**A minimal size:** This minimal size is usually identical to the initial size of the controls and may either be set explicitly in the `wxSize` field of the control constructor or may be calculated by `wxWidgets`, typically by setting the height and/or the width of the item to -1. Note that only some controls can calculate their size (such as a checkbox) whereas others (such as a listbox) don't have any natural width or height and thus require an explicit size. Some controls can calculate their height, but not their width (e.g. a single line text control):

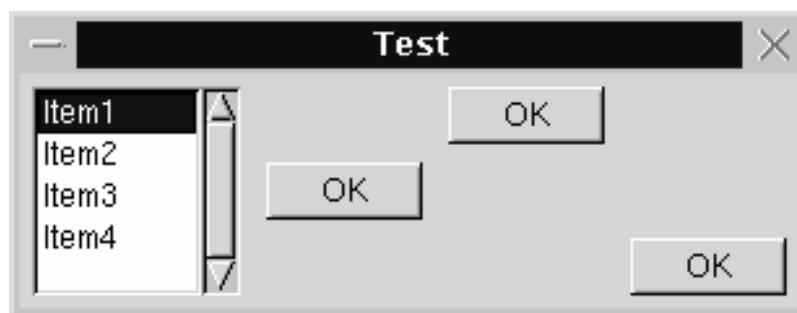


**A border:** The border is just empty space and is used to separate dialog items in a dialog. This border can either be all around, or at any combination of sides such as only above and below the control. The thickness of this border must be set explicitly, typically 5 points. The following samples show dialogs with only one dialog item (a button) and a border of 0, 5, and 10 pixels around the button:

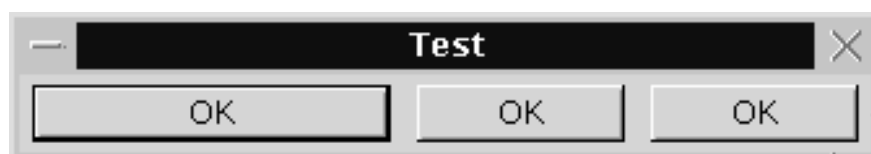




**An alignment:** Often, a dialog item is given more space than its minimal size plus its border. Depending on what flags are used for the respective dialog item, the dialog item can be made to fill out the available space entirely, i.e. it will grow to a size larger than the minimal size, or it will be moved to either the centre of the available space or to either side of the space. The following sample shows a listbox and three buttons in a horizontal box sizer; one button is centred, one is aligned at the top, one is aligned at the bottom:



**A stretch factor:** If a sizer contains more than one child and it is offered more space than its children and their borders need, the question arises how to distribute the surplus space among the children. For this purpose, a stretch factor may be assigned to each child, where the default value of 0 indicates that the child will not get more space than its requested minimum size. A value of more than zero is interpreted in relation to the sum of all stretch factors in the children of the respective sizer, i.e. if two children get a stretch factor of 1, they will get half the extra space each *independent of whether one control has a minimal sizer inferior to the other or not*. The following sample shows a dialog with three buttons, the first one has a stretch factor of 1 and thus gets stretched, whereas the other two buttons have a stretch factor of zero and keep their initial width:



Within wxDesigner, this stretch factor gets set from the *Option* menu.

## Hiding controls using sizers

You can hide controls contained in sizers the same way you would hide any control, using the `wxWindow::Show` (p. 1850) method.

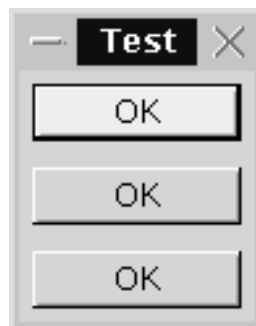
However, `wxSizer` also offers a separate method which can tell the sizer not to consider that control in its size calculations. To hide a window using the sizer, call `wxSizer::Show` (p. 1455). You must then call `Layout` on the sizer to force an update.

This is useful when hiding parts of the interface, since you can avoid removing the controls from the sizer and having to add them back later.

Note: This is supported only by `wxBoxSizer` and `wxFlexGridSizer`.

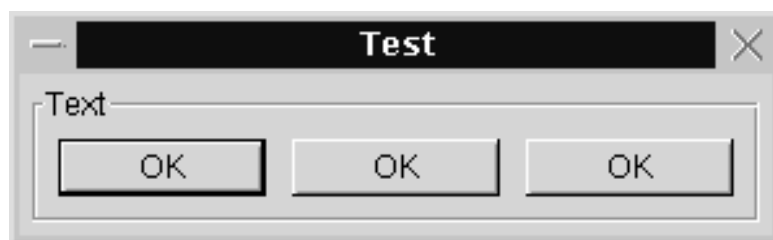
### **wxBoxSizer**

`wxBoxSizer` (p. 149) can lay out its children either vertically or horizontally, depending on what flag is being used in its constructor. When using a vertical sizer, each child can be centered, aligned to the right or aligned to the left. Correspondingly, when using a horizontal sizer, each child can be centered, aligned at the bottom or aligned at the top. The stretch factor described in the last paragraph is used for the main orientation, i.e. when using a horizontal box sizer, the stretch factor determines how much the child can be stretched horizontally. The following sample shows the same dialog as in the last sample, only the box sizer is a vertical box sizer now:



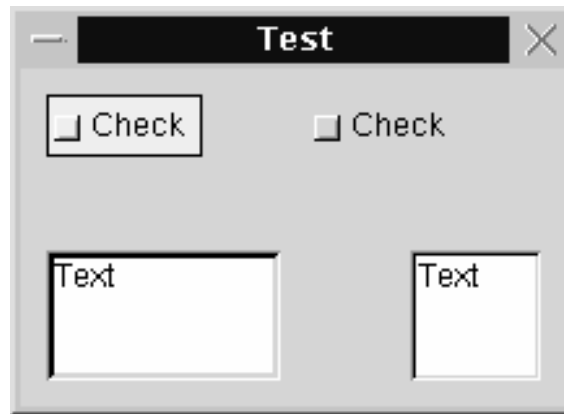
### **wxStaticBoxSizer**

`wxStaticBoxSizer` (p. 1532) is the same as a `wxBoxSizer`, but surrounded by a static box. Here is a sample:



### **wxGridSizer**

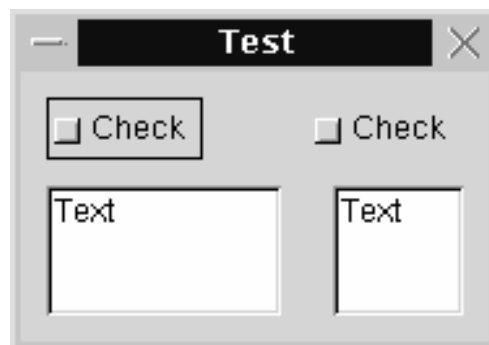
`wxGridSizer` (p. 797) is a two-dimensional sizer. All children are given the same size, which is the minimal size required by the biggest child, in this case the text control in the left bottom border. Either the number of columns or the number of rows is fixed and the grid sizer will grow in the respectively other orientation if new children are added:



For programming information, see *wxGridSizer* (p. 797).

### **wxFlexGridSizer**

Another two-dimensional sizer derived from *wxGridSizer*. The width of each column and the height of each row are calculated individually according to the minimal requirements from the respectively biggest child. Additionally, columns and rows can be declared to be stretchable if the sizer is assigned a size different from the one it requested. The following sample shows the same dialog as the one above, but using a flex grid sizer:



### **Programming with wxBoxSizer**

The basic idea behind a *wxBoxSizer* (p. 149) is that windows will most often be laid out in rather simple basic geometry, typically in a row or a column or several hierarchies of either.

As an example, we will construct a dialog that will contain a text field at the top and two buttons at the bottom. This can be seen as a top-hierarchy column with the text at the top and buttons at the bottom and a low-hierarchy row with an OK button to the left and a Cancel button to the right. In many cases (particularly dialogs under Unix and normal frames) the main window will be resizable by the user and this change of size will have to get propagated to its children. In our case, we want the text area to grow with the dialog, whereas the button shall have a fixed size. In addition, there will be a thin border around all controls to make the dialog look nice and - to make matter worse - the buttons shall be centred as the width of the dialog changes.

It is the unique feature of a box sizer, that it can grow in both directions (height and width)

but can distribute its growth in the main direction (horizontal for a row) *unevenly* among its children. In our example case, the vertical sizer is supposed to propagate all its height changes to only the text area, not to the button area. This is determined by the *proportion* parameter when adding a window (or another sizer) to a sizer. It is interpreted as a weight factor, i.e. it can be zero, indicating that the window may not be resized at all, or above zero. If several windows have a value above zero, the value is interpreted relative to the sum of all weight factors of the sizer, so when adding two windows with a value of 1, they will both get resized equally much and each half as much as the sizer owning them. Then what do we do when a column sizer changes its width? This behaviour is controlled by *flags* (the second parameter of the Add() function): Zero or no flag indicates that the window will preserve its original size, wxGROW flag (same as wxEXPAND) forces the window to grow with the sizer, and wxSHAPED flag tells the window to change its size proportionally, preserving original aspect ratio. When wxGROW flag is not used, the item can be aligned within available space. wxALIGN\_LEFT, wxALIGN\_TOP, wxALIGN\_RIGHT, wxALIGN\_BOTTOM, wxALIGN\_CENTER\_HORIZONTAL and wxALIGN\_CENTER\_VERTICAL do what they say. wxALIGN\_CENTRE (same as wxALIGN\_CENTER) is defined as (wxALIGN\_CENTER\_HORIZONTAL | wxALIGN\_CENTER\_VERTICAL). Default alignment is wxALIGN\_LEFT | wxALIGN\_TOP.

As mentioned above, any window belonging to a sizer may have border, and it can be specified which of the four sides may have this border, using the wxTOP, wxLEFT, wxRIGHT and wxBOTTOM constants or wxALL for all directions (and you may also use wxNORTH, wxWEST etc instead). These flags can be used in combination with the alignment flags above as the second parameter of the Add() method using the binary or operator |. The size of the border also must be made known, and it is the third parameter in the Add() method. This means, that the entire behaviour of a sizer and its children can be controlled by the three parameters of the Add() method.

```
// we want to get a dialog that is stretchable because it
// has a text ctrl at the top and two buttons at the bottom

MyDialog::MyDialog(wxFrame *parent, wxWindowID id, const wxString
&title )
    : wxDialog(parent, id, title, wxDefaultPosition,
wxDefaultSize,
                wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER)
{
    wxBoxSizer *topsizer = new wxBoxSizer( wxVERTICAL );

    // create text ctrl with minimal size 100x60
    topsizer->Add(
        new wxTextCtrl( this, -1, "My text.", wxDefaultPosition,
wxSize(100,60), wxTE_MULTILINE),
        1,           // make vertically stretchable
        wxEXPAND |   // make horizontally stretchable
        wxALL,       // and make border all around
        10 );        // set border width to 10

    wxBoxSizer *button_sizer = new wxBoxSizer( wxHORIZONTAL );
    button_sizer->Add(
        new wxButton( this, wxID_OK, "OK" ),
        0,           // make horizontally unstretchable
```

```
        wxALL,          // make border all around (implicit top alignment)
        10 );          // set border width to 10
button_sizer->Add(
    new wxButton( this, wxID_CANCEL, "Cancel" ),
    0,              // make horizontally unstretchable
    wxALL,          // make border all around (implicit top alignment)
    10 );          // set border width to 10

topSizer->Add(
    button_sizer,
    0,              // make vertically unstretchable
    wxALIGN_CENTER ); // no border and centre horizontally

SetSizer( topSizer );      // use the sizer for layout

topSizer->SetSizeHints( this ); // set size hints to honour minimum
size
}
```

Note that the new way of specifying flags to `wxSizer` is via `wxSizerFlags` (p. 1455). This class greatly eases the burden of passing flags to a `wxSizer`.

Here's how you'd do the previous example with `wxSizerFlags`:

```
// we want to get a dialog that is stretchable because it
// has a text ctrl at the top and two buttons at the bottom

MyDialog::MyDialog(wxFrame *parent, wxWindowID id, const wxString
&title )
    : wxDialog(parent, id, title, wxDefaultPosition,
wxDefaultSize,
                wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER)
{
    wxBoxSizer *topSizer = new wxBoxSizer( wxVERTICAL );

    // create text ctrl with minimal size 100x60 that is horizontally
    and
    // vertically stretchable with a border width of 10
    topSizer->Add(
        new wxTextCtrl( this, -1, "My text.", wxDefaultPosition,
wxSize(100,60), wxTE_MULTILINE),
        wxSizerFlags(1).Align().Expand().Border(wxALL, 10));

    wxBoxSizer *button_sizer = new wxBoxSizer( wxHORIZONTAL );

    //create two buttons that are horizontally unstretchable,
    // with an all-around border with a width of 10 and implicit top
    alignment
    button_sizer->Add(
        new wxButton( this, wxID_OK, "OK" ),
        wxSizerFlags(0).Align().Border(wxALL, 10));

    button_sizer->Add(
        new wxButton( this, wxID_CANCEL, "Cancel" ),
        wxSizerFlags(0).Align().Border(wxALL, 10));
}
```



```
//create a sizer with no border and centered horizontally
topSizer->Add(
    button_sizer,
    wxSizerFlags(0).Center() );

SetSizer( topSizer );      // use the sizer for layout

topSizer->SetSizeHints( this ); // set size hints to honour minimum
size
}
```

## Programming with wxGridSizer

*wxGridSizer* (p. 797) is a sizer which lays out its children in a two-dimensional table with all table fields having the same size, i.e. the width of each field is the width of the widest child, the height of each field is the height of the tallest child.

## Programming with wxFlexGridSizer

*wxFlexGridSizer* (p. 652) is a sizer which lays out its children in a two-dimensional table with all table fields in one row having the same height and all fields in one column having the same width, but all rows or all columns are not necessarily the same height or width as in the *wxGridSizer* (p. 797).

## Programming with wxStaticBoxSizer

*wxStaticBoxSizer* (p. 1532) is a sizer derived from *wxBoxSizer* but adds a static box around the sizer. Note that this static box has to be created separately.

## CreateButtonSizer

As a convenience, *CreateButtonSizer* ( long flags ) can be used to create a standard button sizer in which standard buttons are displayed. The following flags can be passed to this function:

```
wxYES_NO // Add Yes/No subpanel
wxYES    // return wxID_YES
wxNO     // return wxID_NO
wxNO_DEFAULT // make the wxNO button the default, otherwise wxYES
or wxOK button will be default

wxOK      // return wxID_OK
wxCANCEL  // return wxID_CANCEL
wxHELP    // return wxID_HELP

wxFORWARD // return wxID_FORWARD
wxBACKWARD // return wxID_BACKWARD
wxSETUP   // return wxID_SETUP
wxMORE    // return wxID_MORE
```

## XML-based resource system overview

Classes: *wxXmlResource* (p. 1878), *wxXmlResourceHandler* (p. 1883)

The XML-based resource system, known as XRC, allows user interface elements such as dialogs, menu bars and toolbars, to be stored in text files and loaded into the application at run-time. XRC files can also be compiled into binary XRS files or C++ code (the former makes it possible to store all resources in a single file and the latter is useful when you want to embed the resources into the executable).

There are several advantages to using XRC resources.

- Recompiling and linking an application is not necessary if the resources change.
- If you use a dialog designer that generates C++ code, it can be hard to reintegrate this into existing C++ code. Separation of resources and code is a more elegant solution.
- You can choose between different alternative resource files at run time, if necessary.
- The XRC format uses sizers for flexibility, allowing dialogs to be resizable and highly portable.
- The XRC format is a wxWidgets standard, and can be generated or postprocessed by any program that understands it. As it is based on the XML standard, existing XML editors can be used for simple editing purposes.

XRC was written by Vaclav Slavik.

### XRC concepts

These are the typical steps for using XRC files in your application.

- Include the appropriate headers: normally "wx/xrc/xmlres.h" will suffice;
- If you are going to use *XRS files* (p. 2107), install wxFileSystem archive handler first with `wxFileSystem::AddHandler(new wxArchiveFSHandler);`
- call `wxXmlResource::Get()->InitAllHandlers()` from your `wxApp::OnInit` function, and then call `wxXmlResource::Get()->Load("myfile.xrc")` to load the resource file;
- to create a dialog from a resource, create it using the default constructor, and then load it using for example `wxXmlResource::Get()->LoadDialog(&dlg, this, "dlg1");`
- set up event tables as usual but use the `XRCID(str)` macro to translate from XRC string names to a suitable integer identifier, for example `EVT_MENU(XRCID("quit"), MyFrame::OnQuit).`

To create an XRC file, you can use one of the following methods.

- Create the file by hand;
- use wxDesigner (<http://www.roebling.de>), a commercial dialog designer/RAD tool;
- use DialogBlocks (<http://www.anthemion.co.uk/dialogblocks>), a commercial dialog editor;
- use XRCed (<http://xrced.sf.net>), a wxPython-based dialog editor that you can find in the `wxPython/tools` subdirectory of the wxWidgets CVS archive;
- use wxGlade (<http://wxglade.sf.net>), a GUI designer written in wxPython. At the moment it can generate Python, C++ and XRC;
- convert WIN32 RC files to XRC with the tool in `contrib/utils/convertrc`.

A complete list of third-party tools that write to XRC can be found at [www.wxwidgets.org/wiki/index.php/Tools](http://www.wxwidgets.org/wiki/index.php/Tools) (<http://www.wxwidgets.org/wiki/index.php/Tools>).

It is highly recommended that you use a resource editing tool, since it's fiddly writing XRC files by hand.

You can use `wxXmlResource::Load` (p. 1881) in a number of ways. You can pass an XRC file (XML-based text resource file) or a *zip-compressed file* (p. 2107) (extension ZIP or XRS) containing other XRC.

You can also use *embedded C++ resources* (p. 2108)

## Using binary resource files

To compile binary resource files, use the command-line `wxrc` utility. It takes one or more file parameters (the input XRC files) and the following switches and options:

- `-h` (`--help`): show a help message
- `-v` (`--verbose`): show verbose logging information
- `-c` (`--cpp-code`): write C++ source rather than a XRS file
- `-e` (`--extra-cpp-code`): if used together with `-c`, generates C++ header file containing class definitions for the windows defined by the XRC file (see special subsection)
- `-u` (`--uncompressed`): do not compress XML files (C++ only)
- `-g` (`--gettext`): output underscore-wrapped strings that `poEdit` or `gettext` can scan. Outputs to `stdout`, or a file if `-o` is used
- `-n` (`--function`) `<name>`: specify C++ function name (use with `-c`)
- `-o` (`--output`) `<filename>`: specify the output file, such as `resource.xrs` or `resource.cpp`

- `-l (--list-of-handlers) <filename>`: output a list of necessary handlers to this file

For example: `% wxrc resource.xrc`  
`% wxrc resource.xrc -o resource.xrs`  
`% wxrc resource.xrc -v -c -o resource.cpp`

### Note

XRS file is essentially a renamed ZIP archive which means that you can manipulate it with standard ZIP tools. Note that if you are using XRS files, you have to initialize the *wxFileSystem* (p. 633) archive handler first! It is a simple thing to do:

```
#include <wx/filesys.h>
#include <wx/fs_arc.h>
...
wxFileSystem::AddHandler(new wxArchiveFSHandler);
```

### Using embedded resources

It is sometimes useful to embed resources in the executable itself instead of loading an external file (e.g. when your app is small and consists only of one exe file). XRC provides means to convert resources into regular C++ file that can be compiled and included in the executable.

Use the `-c` switch to `wxrc` utility to produce C++ file with embedded resources. This file will contain a function called *InitXmlResource* (unless you override this with a command line switch). Use it to load the resource:

```
extern void InitXmlResource(); // defined in generated file
...
wxXmlResource::Get()->InitAllHandlers();
InitXmlResource();
...
```

### XRC C++ sample

This is the C++ source file (`xrcdemo.cpp`) for the XRC sample.

```
#include "wx/wx.h"
#include "wx/image.h"
#include "wx/xrc/xmlres.h"

// the application icon
#ifdef __WXGTK__ || defined(__WXMOTIF__) || defined(__WXMAC__)
    #include "rc/appicon.xpm"
#endif

//
-----
// private classes
//
-----
```

```
-----

// Define a new application type, each program should derive a class
from wxApp
class MyApp : public wxApp
{
public:
    // override base class virtuals
    // -----

    // this one is called on application startup and is a good place
    for the app
    // initialization (doing it here and not in the ctor allows to have
    an error
    // return: if OnInit() returns false, the application terminates)
    virtual bool OnInit();
};

// Define a new frame type: this is going to be our main frame
class MyFrame : public wxFrame
{
public:
    // ctor(s)
    MyFrame(const wxString& title, const wxPoint& pos, const wxSize&
size);

    // event handlers (these functions should _not_ be virtual)
    void OnQuit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);
    void OnDlg1(wxCommandEvent& event);
    void OnDlg2(wxCommandEvent& event);

private:
    // any class wishing to process wxWidgets events must use this
    macro
    DECLARE_EVENT_TABLE()
};

//
-----

// event tables and other macros for wxWidgets
//
-----

BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(XRCID("menu_quit"), MyFrame::OnQuit)
    EVT_MENU(XRCID("menu_about"), MyFrame::OnAbout)
    EVT_MENU(XRCID("menu_dlg1"), MyFrame::OnDlg1)
    EVT_MENU(XRCID("menu_dlg2"), MyFrame::OnDlg2)
END_EVENT_TABLE()

IMPLEMENT_APP(MyApp)

//
-----
```

```
-----
// the application class
//
-----

// 'Main program' equivalent: the program execution "starts" here
bool MyApp::OnInit()
{
    wxImage::AddHandler(new wxGIFHandler);
    wxXmlResource::Get()->InitAllHandlers();
    wxXmlResource::Get()->Load("rc/resource.xrc");

    MyFrame *frame = new MyFrame("XML resources demo",
                                wxPoint(50, 50), wxSize(450,
340));
    frame->Show(true);
    return true;
}

//
-----

// main frame
//
-----

// frame constructor
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const
wxSize& size)
    : wxFrame((wxFrame *)NULL, -1, title, pos, size)
{
    SetIcon(wxICON(appicon));

    SetMenuBar(wxXmlResource::Get()->LoadMenuBar("mainmenu"));
    SetToolBar(wxXmlResource::Get()->LoadToolBar(this,
"toolbar"));
}

// event handlers
void MyFrame::OnQuit(wxCommandEvent& WXUNUSED(event))
{
    // true is to force the frame to close
    Close(true);
}

void MyFrame::OnAbout(wxCommandEvent& WXUNUSED(event))
{
    wxString msg;
    msg.Printf(_T("This is the about dialog of XML resources
demo.\n"))
        _T("Welcome to %s"), wxVERSION_STRING);

    wxMessageBox(msg, "About XML resources demo", wxOK |
wxICON_INFORMATION, this);
}
```

```
void MyFrame::OnDlg1(wxCommandEvent& WXUNUSED(event))
{
    wxDialog dlg;
    wxXmlResource::Get()->LoadDialog(&dlg, this, "dlg1");
    dlg.ShowModal();
}

void MyFrame::OnDlg2(wxCommandEvent& WXUNUSED(event))
{
    wxDialog dlg;
    wxXmlResource::Get()->LoadDialog(&dlg, this, "dlg2");
    dlg.ShowModal();
}
```

## XRC resource file sample

This is the XML file (resource.xrc) for the XRC sample.

```
<?xml version="1.0"?>
<resource version="2.3.0.1">
  <object class="wxMenuBar" name="mainmenu">
    <style>wxMB_DOCKABLE</style>
    <object class="wxMenu" name="menu_file">
      <label>_File</label>
      <style>wxMENU_TEAROFF</style>
      <object class="wxMenuItem" name="menu_about">
        <label>_About...</label>
        <bitmap>filesave.gif</bitmap>
      </object>
      <object class="separator"/>
      <object class="wxMenuItem" name="menu_dlg1">
        <label>Dialog 1</label>
      </object>
      <object class="wxMenuItem" name="menu_dlg2">
        <label>Dialog 2</label>
      </object>
      <object class="separator"/>
      <object class="wxMenuItem" name="menu_quit">
        <label>E_xit\tAlt-X</label>
      </object>
    </object>
  </object>
  <object class="wxToolBar" name="toolbar">
    <style>wxTB_FLAT|wxTB_DOCKABLE</style>
    <margins>2,2</margins>
    <object class="tool" name="menu_open">
      <bitmap>fileopen.gif</bitmap>
      <tooltip>Open catalog</tooltip>
    </object>
    <object class="tool" name="menu_save">
      <bitmap>filesave.gif</bitmap>
      <tooltip>Save catalog</tooltip>
    </object>
    <object class="tool" name="menu_update">
      <bitmap>update.gif</bitmap>
    </object>
  </object>
</resource>
```

---

```

        <tooltip>Update catalog - synchronize it with sources</tooltip>
    </object>
    <separator/>
    <object class="tool" name="menu_quotes">
        <bitmap>quotes.gif</bitmap>
        <toggle>1</toggle>
        <tooltip>Display quotes around the string?</tooltip>
    </object>
    <object class="separator"/>
    <object class="tool" name="menu_fuzzy">
        <bitmap>fuzzy.gif</bitmap>
        <tooltip>Toggled if selected string is fuzzy
translation</tooltip>
        <toggle>1</toggle>
    </object>
</object>
<object class="wxDialog" name="dlg1">
    <object class="wxBoxSizer">
        <object class="sizeritem">
            <object class="wxBitmapButton">
                <bitmap>fuzzy.gif</bitmap>
                <focus>fileopen.gif</focus>
            </object>
        </object>
        <object class="sizeritem">
            <object class="wxPanel">
                <object class="wxStaticText">
                    <label>fdgdfgdfgdfg</label>
                </object>
                <style>wxSUNKEN_BORDER</style>
            </object>
            <flag>wxALIGN_CENTER</flag>
        </object>
        <object class="sizeritem">
            <object class="wxButton">
                <label>Buttonek</label>
            </object>
            <border>10d</border>
            <flag>wxALL</flag>
        </object>
        <object class="sizeritem">
            <object class="wxHtmlWindow">
                <htmlcode><h1>Hi,</h1><h1>man</h1></htmlcode>
                <size>100,45d</size>
            </object>
        </object>
        <object class="sizeritem">
            <object class="wxNotebook">
                <object class="notebookpage">
                    <object class="wxPanel">
                        <object class="wxBoxSizer">
                            <object class="sizeritem">
                                <object class="wxHtmlWindow">
                                    <htmlcode>Hello, we are inside a
&lt;/u&gt;NOTEBOOK&lt;/u&gt;...</htmlcode>
                                    <size>50,50d</size>
                                </object>

```



```
        <option>1</option>
      </object>
    </object>
  </object>
  <label>Page</label>
</object>
<object class="notebookpage">
  <object class="wxPanel">
    <object class="wxBoxSizer">
      <object class="sizeritem">
        <object class="wxHtmlWindow">
          <htmlcode>Hello, we are inside a
&lt;/u&gt;NOTEBOOK&lt;/u&gt;...</htmlcode>
          <size>50,50d</size>
        </object>
      </object>
    </object>
  </object>
  <label>Page 2</label>
</object>
<usenotebooksizer>1</usenotebooksizer>
</object>
<flag>wxEXPAND</flag>
</object>
<orient>wxVERTICAL</orient>
</object>
</object>
<object class="wxDialog" name="dlg2">
  <object class="wxBoxSizer">
    <orient>wxVERTICAL</orient>
    <object class="sizeritem" name="dfgdfg">
      <object class="wxTextCtrl">
        <size>200,200d</size>
        <style>wxTE_MULTILINE|wxSUNKEN_BORDER</style>
        <value>Hello, this is an ordinary multiline\n
textctrl....</value>
      </object>
      <option>1</option>
      <flag>wxEXPAND|wxALL</flag>
      <border>10</border>
    </object>
    <object class="sizeritem">
      <object class="wxBoxSizer">
        <object class="sizeritem">
          <object class="wxButton" name="wxID_OK">
            <label>Ok</label>
            <default>1</default>
          </object>
        </object>
        <object class="sizeritem">
          <object class="wxButton" name="wxID_CANCEL">
            <label>Cancel</label>
          </object>
        </object>
      <border>10</border>
      <flag>wxLEFT</flag>
    </object>
  </object>
</object>
```

```
        <flag>wxLEFT|wxRIGHT|wxBOTTOM|wxALIGN_RIGHT</flag>
        <border>10</border>
    </object>
</object>
<title>Second testing dialog</title>
</object>
</resource>
```

## XRC file format

Please see Technical Note 14 (docs/tech/tn0014.txt) in your wxWidgets distribution.

## C++ header file generation

Using the `-e` switch together with `-c`, a C++ header file is written containing class definitions for the GUI windows defined in the XRC file. This code generation can make it easier to use XRC and automate program development. The classes can be used as basis for development, freeing the programmer from dealing with most of the XRC specifics (e.g. `XRCCTRL`).

For each top level window defined in the XRC file a C++ class definition is generated, containing as class members the named widgets of the window. A default constructor for each class is also generated. Inside the constructor all XRC loading is done and all class members representing widgets are initialized.

A simple example will help understand how the scheme works. Suppose you have a XRC file defining a top level window `TestWnd_Base`, which subclasses `wxFrame` (any other class like `wxDialog` will do also), and has subwidgets `wxTextCtrl A` and `wxButton B`. The XRC file and corresponding class definition in the header file will be something like:

```
<?xml version="1.0"?>
<resource version="2.3.0.1">
    <object class="wxFrame" name="TestWnd_Base">
        <size>-1,-1</size>
        <title>Test</title>
        <object class="wxBoxSizer">
            <orient>wxHORIZONTAL</orient>
            <object class="sizeritem">
                <object class="wxTextCtrl" name="A">
                    <label>Test label</label>
                </object>
            </object>
            <object class="sizeritem">
                <object class="wxButton" name="B">
                    <label>Test button</label>
                </object>
            </object>
        </object>
    </object>
</resource>

class TestWnd_Base : public wxFrame {
```

```
protected:
    wxTextCtrl* A;
    wxButton* B;

private:
    void InitWidgetsFromXRC(){
        wxXmlResource::Get()->LoadObject(this,NULL,"TestWnd","wxFrame");
        A = XRCCTRL(*this,"A",wxTextCtrl);
        B = XRCCTRL(*this,"B",wxButton);
    }
public:
    TestWnd::TestWnd(){
        InitWidgetsFromXRC();
    }
};
```

The generated window class can be used as basis for the full window class. The class members which represent widgets may be accessed by name instead of using `XRCCTRL` every time you wish to reference them (note that they are `protected` class members), though you must still use `XRCID` to refer to widget IDs in the event table.

Example:

```
#include "resource.h"

class TestWnd : public TestWnd_Base {
public:
    TestWnd(){
        // A, B already initialised at this point
        A->SetValue("Updated in TestWnd::TestWnd");
        B->SetValue("Nice :)");
    }
    void OnBPressed(wxEvent& event){
        Close();
    }
    DECLARE_EVENT_TABLE();
};

BEGIN_EVENT_TABLE(TestWnd,TestWnd_Base)
    EVT_BUTTON(XRCID("B"),TestWnd::OnBPressed)
END_EVENT_TABLE()
```

## Adding new resource handlers

Coming soon.

## Scrolling overview

Classes: *wxWindow* (p. 1795), *wxScrolledWindow* (p. 1414), *wxIcon* (p. 894), *wxScrollBar* (p. 1408).

Scrollbars come in various guises in *wxWidgets*. All windows have the potential to show a

vertical scrollbar and/or a horizontal scrollbar: it is a basic capability of a window. However, in practice, not all windows do make use of scrollbars, such as a single-line `wxTextCtrl`.

Because any class derived from `wxWindow` (p. 1795) may have scrollbars, there are functions to manipulate the scrollbars and event handlers to intercept scroll events. But just because a window generates a scroll event, doesn't mean that the window necessarily handles it and physically scrolls the window. The base class `wxWindow` in fact doesn't have any default functionality to handle scroll events. If you created a `wxWindow` object with scrollbars, and then clicked on the scrollbars, nothing at all would happen. This is deliberate, because the *interpretation* of scroll events varies from one window class to another.

`wxScrolledWindow` (p. 1414) (formerly `wxCanvas`) is an example of a window that adds functionality to make scrolling really work. It assumes that scrolling happens in consistent units, not different-sized jumps, and that page size is represented by the visible portion of the window. It is suited to drawing applications, but perhaps not so suitable for a sophisticated editor in which the amount scrolled may vary according to the size of text on a given line. For this, you would derive from `wxWindow` and implement scrolling yourself. `wxGrid` (p. 735) is an example of a class that implements its own scrolling, largely because columns and rows can vary in size.

### The scrollbar model

The function `wxWindow::SetScrollbar` (p. 1843) gives a clue about the way a scrollbar is modeled. This function takes the following arguments:

orientation	Which scrollbar: <code>wxVERTICAL</code> or <code>wxHORIZONTAL</code> .
position	The position of the scrollbar in scroll units.
visible	The size of the visible portion of the scrollbar, in scroll units.
range	The maximum position of the scrollbar.
refresh	Whether the scrollbar should be repainted.

*orientation* determines whether we're talking about the built-in horizontal or vertical scrollbar.

*position* is simply the position of the 'thumb' (the bit you drag to scroll around). It is given in scroll units, and so is relative to the total range of the scrollbar.

*visible* gives the number of scroll units that represents the portion of the window currently visible. Normally, a scrollbar is capable of indicating this visually by showing a different length of thumb.

*range* is the maximum value of the scrollbar, where zero is the start position. You choose the units that suit you, so if you wanted to display text that has 100 lines, you would set this to 100. Note that this doesn't have to correspond to the number of pixels scrolled - it is up to you how you actually show the contents of the window.

*refresh* just indicates whether the scrollbar should be repainted immediately or not.

### An example

Let's say you wish to display 50 lines of text, using the same font. The window is sized so that you can only see 16 lines at a time.

You would use:

```
SetScrollbar(wxVERTICAL, 0, 16, 50);
```

Note that with the window at this size, the thumb position can never go above 50 minus 16, or 34.

You can determine how many lines are currently visible by dividing the current view size by the character height in pixels.

When defining your own scrollbar behaviour, you will always need to recalculate the scrollbar settings when the window size changes. You could therefore put your scrollbar calculations and `SetScrollbar` call into a function named `AdjustScrollbars`, which can be called initially and also from your `wxSizeEvent` (p. 1443) handler function.

## Bitmaps and icons overview

Classes: *wxBitmap* (p. 123), *wxBitmapHandler* (p. 146), *wxIcon* (p. 894), *wxCursor* (p. 297).

The `wxBitmap` class encapsulates the concept of a platform-dependent bitmap, either monochrome or colour. Platform-specific methods for creating a `wxBitmap` object from an existing file are catered for, and this is an occasion where conditional compilation will sometimes be required.

A bitmap created dynamically or loaded from a file can be selected into a memory device context (instance of *wxMemoryDC* (p. 1069)). This enables the bitmap to be copied to a window or memory device context using *wxDC::Blit* (p. 457), or to be used as a drawing surface.

See *wxMemoryDC* (p. 1069) for an example of drawing onto a bitmap.

All `wxWidgets` platforms support XPMs for small bitmaps and icons. You may include the XPM inline as below, since it's C code, or you can load it at run-time.

```
#include "mondrian.xpm"
```

Sometimes you wish to use a `.ico` resource on Windows, and XPMs on other platforms (for example to take advantage of Windows' support for multiple icon resolutions). A macro, *wxICON* (p. 1946), is available which creates an icon using an XPM on the appropriate platform, or an icon resource on Windows.

```
wxIcon icon(wxICON(mondrian));

// Equivalent to:

#if defined(__WXGTK__) || defined(__WXMOTIF__)
wxIcon icon(mondrian_xpm);
```

```
#endif

#if defined(__WXMSW__)
wxIcon icon("mondrian");
#endif
```

There is also a corresponding *wxBITMAP* (p. 1944) macro which allows to create the bitmaps in much the same way as *wxICON* (p. 1946) creates icons. It assumes that bitmaps live in resources under Windows or OS2 and XPM files under all other platforms (for XPMs, the corresponding file must be included before this macro is used, of course, and the name of the bitmap should be the same as the resource name under Windows with *\_xpm*suffix). For example:

```
// an easy and portable way to create a bitmap
wxBitmap bmp(wxBITMAP(bmpname));

// which is roughly equivalent to the following
#if defined(__WXMSW__) || defined(__WXPX__)
    wxBitmap bmp("bmpname", wxBITMAP_TYPE_RESOURCE);
#else // Unix
    wxBitmap bmp(bmpname_xpm, wxBITMAP_TYPE_XPM);
#endif
```

You should always use *wxICON* and *wxBITMAP* macros because they work for any platform (unlike the code above which doesn't deal with *wxMac*, *wxX11*, ...) and are more short and clear than versions with *#ifdef*s. Even better, use the same XPMs on all platforms.

## Supported bitmap file formats

The following lists the formats handled on different platforms. Note that missing or partially-implemented formats are automatically supplemented by the *wxImage* (p. 906) to load the data, and then converting it to *wxBitmap* form. Note that using *wxImage* is the preferred way to load images in *wxWidgets*, with the exception of resources (XPM-files or native Windows resources). Writing an image format handler for *wxImage* is also far easier than writing one for *wxBitmap*, because *wxImage* has exactly one format on all platforms whereas *wxBitmap* can store pixel data very differently, depending on colour depths and platform.

### **wxBitmap**

Under Windows, *wxBitmap* may load the following formats:

- Windows bitmap resource (*wxBITMAP\_TYPE\_BMP\_RESOURCE*)
- Windows bitmap file (*wxBITMAP\_TYPE\_BMP*)
- XPM data and file (*wxBITMAP\_TYPE\_XPM*)
- All formats that are supported by the *wxImage* (p. 906) class.

Under *wxGTK*, *wxBitmap* may load the following formats:

- XPM data and file (wxBITMAP\_TYPE\_XPM)
- All formats that are supported by the *wxImage* (p. 906) class.

Under wxMotif and wxX11, wxBitmap may load the following formats:

- XBM data and file (wxBITMAP\_TYPE\_XBM)
- XPM data and file (wxBITMAP\_TYPE\_XPM)
- All formats that are supported by the *wxImage* (p. 906) class.

### **wxIcon**

Under Windows, wxIcon may load the following formats:

- Windows icon resource (wxBITMAP\_TYPE\_ICO\_RESOURCE)
- Windows icon file (wxBITMAP\_TYPE\_ICO)
- XPM data and file (wxBITMAP\_TYPE\_XPM)

Under wxGTK, wxIcon may load the following formats:

- XPM data and file (wxBITMAP\_TYPE\_XPM)
- All formats that are supported by the *wxImage* (p. 906) class.

Under wxMotif and wxX11, wxIcon may load the following formats:

- XBM data and file (wxBITMAP\_TYPE\_XBM)
- XPM data and file (wxBITMAP\_TYPE\_XPM)
- All formats that are supported by the *wxImage* (p. 906) class.

### **wxCursor**

Under Windows, wxCursor may load the following formats:

- Windows cursor resource (wxBITMAP\_TYPE\_CUR\_RESOURCE)
- Windows cursor file (wxBITMAP\_TYPE\_CUR)
- Windows icon file (wxBITMAP\_TYPE\_ICO)
- Windows bitmap file (wxBITMAP\_TYPE\_BMP)

Under wxGTK, wxCursor may load the following formats (in addition to stock cursors):

- None (stock cursors only).

Under wxMotif and wxX11, wxCursor may load the following formats:

- XBM data and file (wxBITMAP\_TYPE\_XBM)

## Bitmap format handlers

To provide extensibility, the functionality for loading and saving bitmap formats is not implemented in the `wxBitmap` class, but in a number of handler classes, derived from `wxBitmapHandler`. There is a static list of handlers which `wxBitmap` examines when a file load/save operation is requested. Some handlers are provided as standard, but if you have special requirements, you may wish to initialise the `wxBitmap` class with some extra handlers which you write yourself or receive from a third party.

To add a handler object to `wxBitmap`, your application needs to include the header which implements it, and then call the static function `wxBitmap::AddHandler` (p. 128).

**Note:** bitmap handlers are not implemented on all platforms, and new ones rarely need to be implemented since `wxImage` can be used for loading most formats, as noted earlier.

## Device context overview

Classes: `wxBufferedDC` (p. 157), `wxBufferedPaintDC` (p. 159), `wxDC` (p. 456), `wxPostScriptDC` (p. 1194), `wxMetafileDC` (p. 1109), `wxMemoryDC` (p. 1069), `wxPrinterDC` (p. 1213), `wxScreenDC` (p. 1407), `wxClientDC` (p. 193), `wxPaintDC` (p. 1164), `wxWindowDC` (p. 1855).

A `wxDC` is a *device context* onto which graphics and text can be drawn. The device context is intended to represent a number of output devices in a generic way, with the same API being used throughout.

Some device contexts are created temporarily in order to draw on a window. This is true of `wxScreenDC` (p. 1407), `wxClientDC` (p. 193), `wxPaintDC` (p. 1164), and `wxWindowDC` (p. 1855). The following describes the differences between these device contexts and when you should use them.

- **wxScreenDC.** Use this to paint on the screen, as opposed to an individual window.
- **wxClientDC.** Use this to paint on the client area of window (the part without borders and other decorations), but do not use it from within an `wxPaintEvent` (p. 1164).
- **wxPaintDC.** Use this to paint on the client area of a window, but *only* from within a `wxPaintEvent` (p. 1164).
- **wxWindowDC.** Use this to paint on the whole area of a window, including decorations. This may not be available on non-Windows platforms.

To use a client, paint or window device context, create an object on the stack with the window as argument, for example:

```
void MyWindow::OnMyCmd(wxCommandEvent& event)
{
    wxClientDC dc(window);
    DrawMyPicture(dc);
}
```



Try to write code so it is parameterised by `wxDC` - if you do this, the same piece of code may write to a number of different devices, by passing a different device context. This doesn't work for everything (for example not all device contexts support bitmap drawing) but will work most of the time.

## wxFont overview

Class: `wxFont` (p. 655), `wxFontDialog` (p. 670)

A font is an object which determines the appearance of text, primarily when drawing text to a window or device context. A font is determined by the following parameters (not all of them have to be specified, of course):

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 <code>NULL</code> , a default typeface will be chosen based on the family.
Encoding	The font encoding (see <b>wxFONTENCODING_XXX</b> constants and the <i>font overview</i> (p. 2122) for more details)

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, however it doesn't allow to choose a font precisely as the parameters above don't suffice, in general, to identify all the available fonts and this is where using the native font descriptions may be helpful - see below.

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 name with any file that might be transported to a different Windows machine or other platform.

**Note:** There is currently a difference between the appearance of fonts on the two platforms, if the mapping mode is anything other than `wxMM_TEXT`. Under X, font size is always specified in points. Under MS Windows, the unit for text is points but the text is scaled according to the current mapping mode. However, user scaling on a device context will also scale fonts under both environments.

## Native font information

An alternative way of choosing fonts is to use the native font description. This is the only acceptable solution if the user is allowed to choose the font using the *wxFontDialog* (p. 670) because the selected font cannot be described using only the family name and so, if only family name is stored permanently, the user would almost surely see a different font in the program later.

Instead, you should store the value returned by *wxFont::GetNativeFontInfoDesc* (p. 661) and pass it to *wxFont::SetNativeFontInfo* (p. 664) later to recreate exactly the same font.

Note that the contents of this string depends on the platform and shouldn't be used for any other purpose (in particular, it is not meant to be shown to the user). Also please note that although the native font information is currently implemented for Windows and Unix (GTK+ and Motif) ports only, all the methods are available for all the ports and should be used to make your program work correctly when they are implemented later.

## Font encoding overview

*wxWidgets* has support for multiple font encodings starting from release 2.2. By encoding we mean here the mapping between the character codes and the letters. Probably the most well-known encoding is (7 bit) ASCII one which is used almost universally now to represent the letters of the English alphabet and some other common characters. However, it is not enough to represent the letters of foreign alphabets and here other encodings come into play. Please note that we will only discuss 8-bit fonts here and not *Unicode* (p. 2056).

Font encoding support is ensured by several classes: *wxFont* (p. 655) itself, but also *wxFontEnumerator* (p. 672) and *wxFontMapper* (p. 674). *wxFont* encoding support is reflected by a (new) constructor parameter *encoding* which takes one of the following values (elements of enumeration type *wxFontEncoding*):

<code>wxFONTENCODING_SYSTEM</code>	The default encoding of the underlying operating system (notice that this might be a "foreign" encoding for foreign versions of Windows 9x/NT).
<code>wxFONTENCODING_DEFAULT</code>	The applications default encoding as returned by <i>wxFont::GetDefaultEncoding</i> (p. 661). On program startup, the applications default encoding is the same as <code>wxFONTENCODING_SYSTEM</code> , but may be changed to make all the fonts created later to use it (by default).
<code>wxFONTENCODING_ISO8859_1..15</code>	ISO8859 family encodings which are usually used by all non-Microsoft operating systems
<code>wxFONTENCODING_KOI8</code>	Standard Cyrillic encoding for the Internet (but see also <code>wxFONTENCODING_ISO8859_5</code> and <code>wxFONTENCODING_CP1251</code> )
<code>wxFONTENCODING_CP1250</code>	Microsoft analogue of ISO8859-2
<code>wxFONTENCODING_CP1251</code>	Microsoft analogue of ISO8859-5
<code>wxFONTENCODING_CP1252</code>	Microsoft analogue of ISO8859-1

As you may see, Microsoft's encoding partly mirror the standard ISO8859 ones, but there are (minor) differences even between ISO8859-1 (Latin1, ISO encoding for Western Europe) and CP1251 (WinLatin1, standard code page for English versions of Windows) and there are more of them for other encodings.

The situation is particularly complicated with Cyrillic encodings for which (more than) three incompatible encodings exist: KOI8 (the old standard, widely used on the Internet), ISO8859-5 (ISO standard for Cyrillic) and CP1251 (WinCyrillic).

This abundance of (incompatible) encodings should make it clear that using encodings is less easy than it might seem. The problems arise both from the fact that the standard encodings for the given language (say Russian, which is written in Cyrillic) are different on different platforms and because the fonts in the given encoding might just not be installed (this is especially a problem with Unix, or, in general, non-Win32 systems).

To clarify, the *wxFontEnumerator* (p. 672) class may be used to enumerate both all available encodings and to find the facename(s) in which the given encoding exists. If you can find the font in the correct encoding with *wxFontEnumerator* then your troubles are over, but, unfortunately, sometimes this is not enough. For example, there is no standard way (that I know of, please tell me if you do!) to find a font on a Windows system for KOI8 encoding (only for WinCyrillic one which is quite different), so *wxFontEnumerator* (p. 672) will never return one, even if the user has installed a KOI8 font on his system.

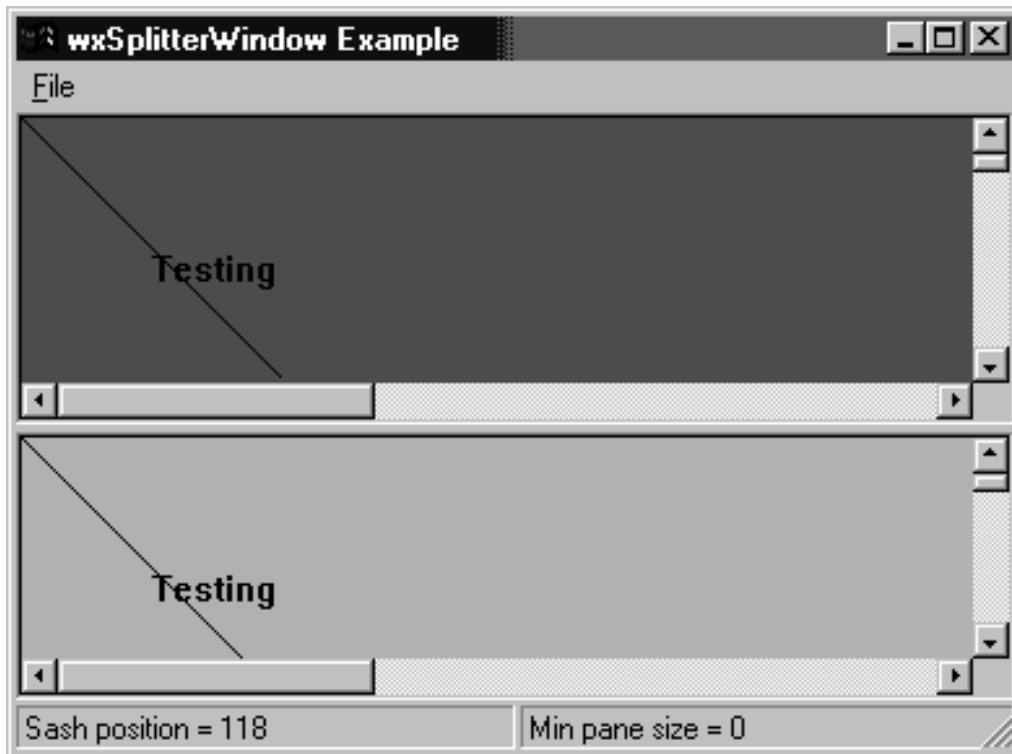
To solve this problem, a *wxFontMapper* (p. 674) class is provided. This class stores the mapping between the encodings and the font face names which support them in *wxConfig* (p. 2074) object. Of course, it would be fairly useless if it tried to determine these mappings by itself, so, instead, it (optionally) asks the user and remembers his answers so that the next time the program will automatically choose the correct font.

All these topics are illustrated by the *font sample* (p. 2035); please refer to it and the documentation of the classes mentioned here for further explanations.

## **wxSplitterWindow overview**

Classes: *wxSplitterWindow* (p. 1508)

The following screenshot shows the appearance of a splitter window with a horizontal split.



The style `wxSP_3D` has been used to show a 3D border and 3D sash.

### Example

The following fragment shows how to create a splitter window, creating two subwindows and hiding one of them.

```

    splitter = new wxSplitterWindow(this, -1, wxPoint(0, 0), wxSize(400,
400), wxSP_3D);

    leftWindow = new MyWindow(splitter);
    leftWindow->SetScrollbars(20, 20, 50, 50);

    rightWindow = new MyWindow(splitter);
    rightWindow->SetScrollbars(20, 20, 50, 50);
    rightWindow->Show(false);

    splitter->Initialize(leftWindow);

    // Set this to prevent unsplitting
    // splitter->SetMinimumPaneSize(20);

```

The next fragment shows how the splitter window can be manipulated after creation.

```

void MyFrame::OnSplitVertical(wxCommandEvent& event)
{
    if ( splitter->IsSplit() )

```

```
        splitter->Unsplit();
        leftWindow->Show(true);
        rightWindow->Show(true);
        splitter->SplitVertically( leftWindow, rightWindow );
    }

    void MyFrame::OnSplitHorizontal(wxCommandEvent& event)
    {
        if ( splitter->IsSplit() )
            splitter->Unsplit();
        leftWindow->Show(true);
        rightWindow->Show(true);
        splitter->SplitHorizontally( leftWindow, rightWindow );
    }

    void MyFrame::OnUnsplit(wxCommandEvent& event)
    {
        if ( splitter->IsSplit() )
            splitter->Unsplit();
    }
}
```

## wxTreeCtrl overview

Classes: *wxTreeCtrl* (p. 1728), *wxImageList* (p. 933)

The tree control displays its items in a tree like structure. Each item has its own (optional) icon and a label. An item may be either collapsed (meaning that its children are not visible) or expanded (meaning that its children are shown). Each item in the tree is identified by its *itemId* which is of opaque data type *wxTreeItemId*. You can test whether an item is valid by calling *wxTreeItemId::IsOk*.

The items text and image may be retrieved and changed with *GetItemText* (p. 1738)/*SetItemText* (p. 1745) and *GetItemImage* (p. 1738)/*SetItemImage* (p. 1745). In fact, an item may even have two images associated with it: the normal one and another one for selected state which is set/retrieved with *SetItemSelectedImage* (p. 1745)/*GetItemSelectedImage* (p. 1740) functions, but this functionality might be unavailable on some platforms.

Tree items have several attributes: an item may be selected or not, visible or not, bold or not. It may also be expanded or collapsed. All these attributes may be retrieved with the corresponding functions: *IsSelected* (p. 1742), *IsVisible* (p. 1742), *IsBold* (p. 1742) and *IsExpanded* (p. 1742). Only one item at a time may be selected, selecting another one (with *SelectItem* (p. 1743)) automatically unselects the previously selected one.

In addition to its icon and label, a user-specific data structure may be associated with all tree items. If you wish to do it, you should derive a class from *wxTreeItemData* which is a very simple class having only one function *GetId()* which returns the id of the item this data is associated with. This data will be freed by the control itself when the associated item is deleted (all items are deleted when the control is destroyed), so you shouldn't delete it yourself (if you do it, you should call *SetItemData(NULL)* (p. 1744) to prevent the tree from deleting the pointer second time). The associated data may be retrieved with

*GetItemData()* (p. 1737) function.

Working with trees is relatively straightforward if all the items are added to the tree at the moment of its creation. However, for large trees it may be very inefficient. To improve the performance you may want to delay adding the items to the tree until the branch containing the items is expanded: so, in the beginning, only the root item is created (with *AddRoot* (p. 1732)). Other items are added when *EVT\_TREE\_ITEM\_EXPANDING* event is received: then all items lying immediately under the item being expanded should be added, but, of course, only when this event is received for the first time for this item - otherwise, the items would be added twice if the user expands/collapses/re-expands the branch.

The tree control provides functions for enumerating its items. There are 3 groups of enumeration functions: for the children of a given item, for the sibling of the given item and for the visible items (those which are currently shown to the user: an item may be invisible either because its branch is collapsed or because it is scrolled out of view). Child enumeration functions require the caller to give them a *cookie* parameter: it is a number which is opaque to the caller but is used by the tree control itself to allow multiple enumerations to run simultaneously (this is explicitly allowed). The only thing to remember is that the *cookie* passed to *GetFirstChild* (p. 1736) and to *GetNextChild* (p. 1739) should be the same variable (and that nothing should be done with it by the user code).

Among other features of the tree control are: item sorting with *SortChildren* (p. 1746) which uses the user-defined comparison function *OnCompareItems* (p. 1743) (by default the comparison is the alphabetic comparison of tree labels), hit testing (determining to which portion of the control the given point belongs, useful for implementing drag-and-drop in the tree) with *HitTest* (p. 1741) and editing of the tree item labels in place (see *EditLabel* (p. 1734)).

Finally, the tree control has a keyboard interface: the cursor navigation (arrow) keys may be used to change the current selection. <HOME> and <END> are used to go to the first/last sibling of the current item. '+', '-' and '\*' expand, collapse and toggle the current branch. Note, however, that <DEL> and <INS> keys do nothing by default, but it is common to associate them with deleting an item from a tree and inserting a new one into it.

## **wxListCtrl overview**

Classes: *wxListCtrl* (p. 980), *wxImageList* (p. 933)

Sorry, this topic has yet to be written.

## **wxImageList overview**

Classes: *wxImageList* (p. 933)

An image list is a list of images that may have transparent areas. The class helps an application organise a collection of images so that they can be referenced by integer index instead of by pointer.

Image lists are used in *wxNotebook* (p. 1135), *wxListCtrl* (p. 980), *wxTreeCtrl* (p. 980) and some other control classes.

## wxBookCtrl overview

Classes: *wxNotebook* (p. 1135), *wxListbook* (p. 974), *wxChoicebook* (p. 189), *wxTreebook* (p. 1721), *wxToolbook* (p. 1711)

### Introduction

A book control is a convenient way of displaying multiple pages of information, displayed one page at a time. wxWidgets has five variants of this control:

- *wxNotebook* (p. 1135): uses a row of tabs
- *wxListbook* (p. 974): controlled by a *wxListCtrl* (p. 980)
- *wxChoicebook* (p. 189): controlled by a *wxChoice* (p. 186)
- *wxTreebook* (p. 1721): controlled by a *wxTreeCtrl* (p. 1728)
- *wxToolbook* (p. 1711): controlled by a *wxToolBar* (p. 1694)

See *Notebook sample* (p. 2036) for an example of wxBookCtrl usage.

### Best book

wxBookCtrl is mapped to the class best suited for a given platform. Currently it provides *wxChoicebook* (p. 189) for smartphones equipped with WinCE, and *wxNotebook* (p. 1135) for all other platforms. The mapping consists of:

wxBookCtrl	wxChoicebook or wxNotebook
wxBookCtrlEvent	wxChoicebookEvent or wxNotebookEvent
wxEVT_COMMAND_BOOKCTRL_PAGE_CHANGED	wxEVT_COMMAND_CHOICEBOOK_PAGE_CHANGED or wxEVT_COMMAND_NOTEBOOK_PAGE_CHANGED
wxEVT_COMMAND_BOOKCTRL_PAGE_CHANGING	wxEVT_COMMAND_CHOICEBOOK_PAGE_CHANGING or wxEVT_COMMAND_NOTEBOOK_PAGE_CHANGING
EVT_BOOKCTRL_PAGE_CHANGED(id, fn)	EVT_CHOICEBOOK_PAGE_CHANGED(id, fn) or EVT_NOTEBOOK_PAGE_CHANGED(id, fn)
EVT_BOOKCTRL_PAGE_CHANGING(id, fn)	EVT_CHOICEBOOK_PAGE_CHANGING(id, fn) or EVT_NOTEBOOK_PAGE_CHANGING(id, fn)

For orientation of the book controller, use following flags in style:**wxBK\_TOP**  
controller above pages

**wxBK\_BOTTOM** controller below pages

**wxBK\_LEFT** controller on the left

**wxBK\_RIGHT** controller on the right

**wxBK\_DEFAULT** native controller placement

## Common dialogs overview

Classes: *wxColourDialog* (p. 221), *wxFontDialog* (p. 670), *wxPrintDialog* (p. 1206), *wxFileDialog* (p. 600), *wxDirDialog* (p. 513), *wxTextEntryDialog* (p. 1655), *wxPasswordEntryDialog* (p. 1173), *wxMessageDialog* (p. 1106), *wxSingleChoiceDialog* (p. 1435), *wxMultiChoiceDialog* (p. 1129)

Common dialog classes and functions encapsulate commonly-needed dialog box requirements. They are all 'modal', grabbing the flow of control until the user dismisses the dialog, to make them easy to use within an application.

Some dialogs have both platform-dependent and platform-independent implementations, so that if underlying windowing systems 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.

### wxColourDialog overview

Classes: *wxColourDialog* (p. 221), *wxColourData* (p. 218)

The *wxColourDialog* presents a colour selector to the user, and returns with colour information.

#### The MS Windows colour selector

Under Windows, the native colour selector common dialog is used. This presents a dialog box with three main regions: at the top left, a palette of 48 commonly-used colours is shown. Under this, there is a palette of 16 'custom colours' which can be set by the application if desired. Additionally, the user may open up the dialog box to show a right-hand panel containing controls to select a precise colour, and add it to the custom colour palette.

#### The generic colour selector

Under non-MS Windows platforms, the colour selector is a simulation of most of the features of the MS Windows selector. Two palettes of 48 standard and 16 custom colours are presented, with the right-hand area containing three sliders for the user to select a colour from red, green and blue components. This colour may be added to the custom



colour palette, and will replace either the currently selected custom colour, or the first one in the palette if none is selected. The RGB colour sliders are not optional in the generic colour selector. The generic colour selector is also available under MS Windows; use the name `wxGenericColourDialog`.

### Example

In the `samples/dialogs` directory, there is an example of using the `wxColourDialog` class. Here is an excerpt, which sets various parameters of a `wxColourData` object, including a grey scale for the custom colours. If the user did not cancel the dialog, the application retrieves the selected colour and uses it to set the background of a window.

```
wxColourData data;
data.SetChooseFull(true);
for (int i = 0; i < 16; i++)
{
    wxColour colour(i*16, i*16, i*16);
    data.SetCustomColour(i, colour);
}

wxColourDialog dialog(this, &data);
if (dialog.ShowModal() == wxID_OK)
{
    wxColourData retData = dialog.GetColourData();
    wxColour col = retData.GetColour();
    wxBrush brush(col, wxSOLID);
    myWindow->SetBackground(brush);
    myWindow->Clear();
    myWindow->Refresh();
}
```

### wxFontDialog overview

Classes: *wxFontDialog* (p. 670), *wxFontData* (p. 668)

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 `wxWidgets` 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 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`.

### Example

In the `samples/dialogs` directory, there is an example of using the `wxFontDialog` class. The application uses the returned font and colour for drawing text on a canvas. Here is an excerpt:

```
wxFontData data;
data.SetInitialFont(canvasFont);
data.SetColour(canvasTextColour);

wxFontDialog dialog(this, &data);
if (dialog.ShowModal() == wxID_OK)
{
    wxFontData retData = dialog.GetFontData();
    canvasFont = retData.GetChosenFont();
    canvasTextColour = retData.GetColour();
    myWindow->Refresh();
}
```

### wxPrintDialog overview

Classes: *wxPrintDialog* (p. 1206), *wxPrintData* (p. 1200)

This class represents the print and print setup common dialogs. You may obtain a *wxPrinterDC* (p. 1213) device context from a successfully dismissed print dialog.

The `samples/printing` example shows how to use it: see *Printing overview* (p. 2145) for an excerpt from this example.

### wxFileDialog overview

Classes: *wxFileDialog* (p. 600)

Pops up a file selector box. In Windows and GTK2.4+, this is the common file selector dialog. In X, this is a file selector box with somewhat less functionality. The path and filename are distinct elements of a full file pathname. If path is "", the current directory will be used. If filename is "", no default filename will be supplied. The wildcard determines what files are displayed in the file selector, and file extension supplies a type extension for the required filename. Flags may be a combination of `wxFD_OPEN`, `wxFD_SAVE`, `wxFD_OVERWRITE_PROMPT`, `wxFD_FILE_MUST_EXIST`, `wxFD_MULTIPLE`, `wxFD_CHANGE_DIR` or 0.

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.

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"
```

**wxDirDialog overview**

Classes: *wxDirDialog* (p. 513)

This dialog shows a directory selector dialog, allowing the user to select a single directory.

**wxTextEntryDialog overview**

Classes: *wxTextEntryDialog* (p. 1655)

This is a dialog with a text entry field. The value that the user entered is obtained using *wxTextEntryDialog::GetValue* (p. 1657).

**wxPasswordEntryDialog overview**

Classes: *wxPasswordEntryDialog* (p. 1173)

This is a dialog with a password entry field. The value that the user entered is obtained using *wxTextEntryDialog::GetValue* (p. 1657).

**wxMessageDialog overview**

Classes: *wxMessageDialog* (p. 1106)

This dialog shows a message, plus buttons that can be chosen from OK, Cancel, Yes, and No. Under Windows, an optional icon can be shown, such as an exclamation mark or question mark.

The return value of *wxMessageDialog::ShowModal* (p. 1107) indicates which button the user pressed.

**wxSingleChoiceDialog overview**

Classes: *wxSingleChoiceDialog* (p. 1435)

This dialog shows a list of choices, plus OK and (optionally) Cancel. The user can select one of them. The selection can be obtained from the dialog as an index, a string or client data.

**wxMultiChoiceDialog overview**

Classes: *wxMultiChoiceDialog* (p. 1129)

This dialog shows a list of choices, plus OK and (optionally) Cancel. The user can select one or more of them.

**Document/view overview**

Classes: *wxDocument* (p. 545), *wxView* (p. 1780), *wxDocTemplate* (p. 540),

*wxDocManager* (p. 527), *wxDocParentFrame* (p. 538), *wxDocChildFrame* (p. 525), *wxDocMDIParentFrame* (p. 537), *wxDocMDIChildFrame* (p. 535), *wxCommand* (p. 249), *wxCommandProcessor* (p. 255)

The document/view framework is found in most application frameworks, because it can dramatically simplify the code required to build many kinds of application.

The idea is that you can model your application primarily in terms of *documents* to store data and provide interface-independent operations upon it, and *views* to visualise and manipulate the data. Documents know how to do input and output given stream objects, and views are responsible for taking input from physical windows and performing the manipulation on the document data. If a document's data changes, all views should be updated to reflect the change.

The framework can provide many user-interface elements based on this model. Once you have defined your own classes and the relationships between them, the framework takes care of popping up file selectors, opening and closing files, asking the user to save modifications, routing menu commands to appropriate (possibly default) code, even some default print/preview functionality and support for command undo/redo. The framework is highly modular, allowing overriding and replacement of functionality and objects to achieve more than the default behaviour.

These are the overall steps involved in creating an application based on the document/view framework:

1. Define your own document and view classes, overriding a minimal set of member functions e.g. for input/output, drawing and initialization.
2. Define any subwindows (such as a scrolled window) that are needed for the view(s). You may need to route some events to views or documents, for example `OnPaint` needs to be routed to `wxView::OnDraw`.
3. Decide what style of interface you will use: Microsoft's MDI (multiple document child frames surrounded by an overall frame), SDI (a separate, unconstrained frame for each document), or single-window (one document open at a time, as in Windows Write).
4. Use the appropriate `wxDocParentFrame` and `wxDocChildFrame` classes. Construct an instance of `wxDocParentFrame` in your `wxApp::OnInit`, and a `wxDocChildFrame` (if not single-window) when you initialize a view. Create menus using standard menu ids (such as `wxID_OPEN`, `wxID_PRINT`).
5. Construct a single `wxDocManager` instance at the beginning of your `wxApp::OnInit`, and then as many `wxDocTemplate` instances as necessary to define relationships between documents and views. For a simple application, there will be just one `wxDocTemplate`.

If you wish to implement Undo/Redo, you need to derive your own class(es) from `wxCommand` and use `wxCommandProcessor::Submit` instead of directly executing code. The framework will take care of calling `Undo` and `Do` functions as appropriate, so long as the `wxID_UNDO` and `wxID_REDO` menu items are defined in the view menu.

Here are a few examples of the tailoring you can do to go beyond the default framework

behaviour:

- Override `wxDocument::OnCreateCommandProcessor` to define a different Do/Undo strategy, or a command history editor.
- Override `wxView::OnCreatePrintout` to create an instance of a derived *wxPrintout* (p. 1214) 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 *wxWidgets predefined command identifiers* (p. 2137) in your menus.

**wxPerl note:** The document/view framework is available in wxPerl. To use it, you will need the following statements in your application code:

```
use Wx::DocView;  
use Wx ':docview'; # import constants (optional)
```

## wxDocument overview

*Document/view framework overview* (p. 2131)

Class: *wxDocument* (p. 545)

The *wxDocument* class can be used to model an application's file-based data. It is part of the document/view framework supported by *wxWidgets*, and cooperates with the *wxView* (p. 1780), *wxDocTemplate* (p. 540) and *wxDocManager* (p. 527) classes.

Using this framework can save a lot of routine user-interface programming, since a range of menu commands -- such as open, save, save as -- are supported automatically. The programmer just needs to define a minimal set of classes and member functions for the framework to call when necessary. Data, and the means to view and edit the data, are explicitly separated out in this model, and the concept of multiple *views* onto the same data is supported.

Note that the document/view model will suit many but not all styles of application. For example, it would be overkill for a simple file conversion utility, where there may be no call for *views* on *documents* or the ability to open, edit and save files. But probably the majority of applications are document-based.

See the example application in `samples/docview`.

To use the abstract *wxDocument* class, you need to derive a new class and override at least the member functions `SaveObject` and `LoadObject`. `SaveObject` and `LoadObject` will be called by the framework when the document needs to be saved or loaded.

Use the macros `DECLARE_DYNAMIC_CLASS` and `IMPLEMENT_DYNAMIC_CLASS` in

order to allow the framework to create document objects on demand. When you create a *wxDocTemplate* (p. 540) 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 *wxWidgets* method of creating document objects dynamically, you must override *wxDocTemplate::CreateDocument* to return an instance of the appropriate class.

## **wxView overview**

*Document/view framework overview* (p. 2131)

Class: *wxView* (p. 1780)

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 *wxWidgets*, and cooperates with the *wxDocument* (p. 545), *wxDocTemplate* (p. 540) and *wxDocManager* (p. 527) 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 will probably want to respond to menu commands from the frame containing the view.

Use the macros `DECLARE_DYNAMIC_CLASS` and `IMPLEMENT_DYNAMIC_CLASS` in order to allow the framework to create view objects on demand. When you create a *wxDocTemplate* (p. 540) 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 *wxWidgets* method of creating view objects dynamically, you must override *wxDocTemplate::CreateView* to return an instance of the appropriate class.

## **wxDocTemplate overview**

*Document/view framework overview* (p. 2131)

Class: *wxDocTemplate* (p. 540)

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.

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 `wxWidgets`, and cooperates with the `wxView` (p. 1780), `wxDocument` (p. 545) and `wxDocManager` (p. 527) 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 `wxWidgets` 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.

## **wxDocManager overview**

*Document/view framework overview* (p. 2131)

Class: `wxDocManager` (p. 527)

The `wxDocManager` class is part of the document/view framework supported by `wxWidgets`, and cooperates with the `wxView` (p. 1780), `wxDocument` (p. 545) and `wxDocTemplate` (p. 540) 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`.

## **wxCommand overview**

*Document/view framework overview* (p. 2131)

Classes: `wxCommand` (p. 249), `wxCommandProcessor` (p. 255)

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

Instead of the application functionality being scattered around switch statements and functions in a way that may be hard to read and maintain, the functionality for a command is explicitly represented as an object which can be manipulated by a framework or application. When a user interface event occurs, the application *submits* a command to a `wxCommandProcessor` (p. 2136) object to execute and store.

The `wxWidgets` 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.

### **wxCommandProcessor overview**

*Document/view framework overview* (p. 2131)

Classes: `wxCommandProcessor` (p. 255), `wxCommand` (p. 249)

`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.

### **wxFileHistory overview**

*Document/view framework overview* (p. 2131)

Classes: `wxFileHistory` (p. 605), `wxDocManager` (p. 527)

`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::UseMenu()` (p. 608) you can associate a file menu with the file history. The menu will then be used for appending filenames that are added to the history. Please notice that currently if the history already contained filenames when `UseMenu()` is called (e.g. when initializing a second MDI child frame), the menu is not automatically initialized with the existing filenames in the history and so you need to call `AddFilesToMenu()` (p. 607) after `UseMenu()` explicitly in order to initialize the menu with the existing list of MRU files. (otherwise an assertion failure is raised in debug builds). The filenames are appended using menu identifiers in the range `wxID_FILE1` to `wxID_FILE9`.

In order to respond to a file load command from one of these identifiers, you need to



handle them using an event handler, for example:

```
BEGIN_EVENT_TABLE(wxDocParentFrame, wxFrame)
    EVT_MENU(wxID_EXIT, wxDocParentFrame::OnExit)
    EVT_MENU_RANGE(wxID_FILE1, wxID_FILE9,
wxDocParentFrame::OnMRUFile)
END_EVENT_TABLE()

void wxDocParentFrame::OnExit(wxCommandEvent& WXUNUSED(event))
{
    Close();
}

void wxDocParentFrame::OnMRUFile(wxCommandEvent& event)
{
    wxString f(m_docManager->GetHistoryFile(event.GetId() -
wxID_FILE1));
    if (!f.empty())
        (void)m_docManager->CreateDocument(f, wxDOC_SILENT);
}
```

### **wxWidgets 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.

- 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)

## Toolbar overview

Classes: *wxToolBar* (p. 1694)

The toolbar family of classes allows an application to use toolbars in a variety of configurations and styles.

The toolbar is a popular user interface component and contains a set of bitmap buttons or toggles. A toolbar gives faster access to an application's facilities than menus, which have to be popped up and selected rather laboriously.

Instead of supplying one toolbar class with a number of different implementations depending on platform, wxWidgets separates out the classes. This is because there are a number of different toolbar styles that you may wish to use simultaneously, and also, future toolbar implementations will emerge which cannot all be shoe-horned into the one class.

For each platform, the symbol **wxToolBar** is defined to be one of the specific toolbar classes.

The following is a summary of the toolbar classes and their differences.

- **wxToolBarBase.** This is a base class with pure virtual functions, and should not be used directly.
- **wxToolBarSimple.** A simple toolbar class written entirely with generic wxWidgets functionality. A simple 3D effect for buttons is possible, but it is not consistent with the Windows look and feel. This toolbar can scroll, and you can have arbitrary numbers of rows and columns.
- **wxToolBarMSW.** This class implements an old-style Windows toolbar, only on Windows. There are small, three-dimensional buttons, which do not (currently) reflect the current Windows colour settings: the buttons are grey. This is the default wxToolBar on 16-bit windows.
- **wxToolBar95.** Uses the native Windows 95 toolbar class. It dynamically adjusts its background and button colours according to user colour settings. **CreateTools** must be called after the tools have been added. No absolute positioning is supported but you can specify the number of rows, and add tool separators with **AddSeparator**. Tooltips are supported. **OnRightClick** is not supported. This is the default wxToolBar on Windows 95, Windows NT 4 and above. With the style **wxTB\_FLAT**, the flat toolbar look is used, with a border that is highlighted when the cursor moves over the buttons.

A toolbar might appear as a single row of images under the menubar, or it might be in a separate frame layout in several rows and columns. The class handles the layout of the images, unless explicit positioning is requested.

A tool is a bitmap which can either be a button (there is no 'state', it just generates an event when clicked) or it can be a toggle. If a toggle, a second bitmap can be provided to depict the 'on' state; if the second bitmap is omitted, either the inverse of the first bitmap will be used (for monochrome displays) or a thick border is drawn around the bitmap (for colour

displays where inverting will not have the desired result).

The Windows-specific toolbar classes expect 16-colour bitmaps that are 16 pixels wide and 15 pixels high. If you want to use a different size, call **SetToolBitmapSize** as the demo shows, before adding tools to the button bar. Don't supply more than one bitmap for each tool, because the toolbar generates all three images (normal, depressed and checked) from the single bitmap you give it.

## Using the toolbar library

Include "wx/toolbar.h", or if using a class directly, one of:

- "wx/msw/tbarmsw.h for wxToolBarMSW
- "wx/msw/tbar95.h for wxToolBar95
- "wx/tbarsmpl.h for wxToolBarSimple

Example of toolbar use are given in the sample program "toolbar". The source is given below. In fact it is out of date because recommended practise is to use event handlers (using EVT\_MENU or EVT\_TOOL) instead of overriding OnLeftClick.

```
////////////////////////////////////
////////////////////////////////////
// Name:          test.cpp
// Purpose:       wxToolBar sample
// Author:        Julian Smart
// Modified by:
// Created:       04/01/98
// RCS-ID:       $Id: ttoolbar.tex 32309 2005-02-22 15:09:56Z ABX $
// Copyright:    (c) Julian Smart
// License:       wxWindows license
////////////////////////////////////
////////////////////////////////////

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

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifdef WX_PRECOMP
#include "wx/wx.h"
#endif

#include "wx/toolbar.h"
#include <wx/log.h>

#include "test.h"

#ifdef __WXGTK__ || defined(__WXMOTIF__)
#include "mondrian.xpm"
#include "bitmaps/new.xpm"
#include "bitmaps/open.xpm"
```

```
#include "bitmaps/save.xpm"
#include "bitmaps/copy.xpm"
#include "bitmaps/cut.xpm"
#include "bitmaps/print.xpm"
#include "bitmaps/preview.xpm"
#include "bitmaps/help.xpm"
#endif

IMPLEMENT_APP(MyApp)

// The 'main program' equivalent, creating the windows and returning
the
// main frame
bool MyApp::OnInit(void)
{
    // Create the main frame window
    MyFrame* frame = new MyFrame((wxFrame *) NULL, -1, (const wxString)
    "wxToolBar Sample",
        wxPoint(100, 100), wxSize(450, 300));

    // Give it a status line
    frame->CreateStatusBar();

    // Give it an icon
    frame->SetIcon(wxICON(mondrian));

    // Make a menubar
    wxMenu *fileMenu = new wxMenu;
    fileMenu->Append(wxID_EXIT, "E&xit", "Quit toolbar sample" );

    wxMenu *helpMenu = new wxMenu;
    helpMenu->Append(wxID_HELP, "&About", "About toolbar sample");

    wxMenuBar* menuBar = new wxMenuBar;

    menuBar->Append(fileMenu, "&File");
    menuBar->Append(helpMenu, "&Help");

    // Associate the menu bar with the frame
    frame->SetMenuBar(menuBar);

    // Create the toolbar
    frame->CreateToolBar(wxNO_BORDER|wxHORIZONTAL|wxTB_FLAT,
    ID_TOOLBAR);

    frame->GetToolBar()->SetMargins( 2, 2 );

    InitToolbar(frame->GetToolBar());

    // Force a resize. This should probably be replaced by a call to a
    wxFrame
    // function that lays out default decorations and the remaining
    content window.
    wxSizeEvent event(wxSize(-1, -1), frame->GetId());
    frame->OnSize(event);
    frame->Show(true);
```

```
    frame->SetStatusText("Hello, wxWidgets");

    SetTopWindow(frame);

    return true;
}

bool MyApp::InitToolBar(wxToolBar* toolBar)
{
    // Set up toolbar
    wxBitmap* toolBarBitmaps[8];

#ifdef __WXMSW__
    toolBarBitmaps[0] = new wxBitmap("icon1");
    toolBarBitmaps[1] = new wxBitmap("icon2");
    toolBarBitmaps[2] = new wxBitmap("icon3");
    toolBarBitmaps[3] = new wxBitmap("icon4");
    toolBarBitmaps[4] = new wxBitmap("icon5");
    toolBarBitmaps[5] = new wxBitmap("icon6");
    toolBarBitmaps[6] = new wxBitmap("icon7");
    toolBarBitmaps[7] = new wxBitmap("icon8");
#else
    toolBarBitmaps[0] = new wxBitmap( new_xpm );
    toolBarBitmaps[1] = new wxBitmap( open_xpm );
    toolBarBitmaps[2] = new wxBitmap( save_xpm );
    toolBarBitmaps[3] = new wxBitmap( copy_xpm );
    toolBarBitmaps[4] = new wxBitmap( cut_xpm );
    toolBarBitmaps[5] = new wxBitmap( preview_xpm );
    toolBarBitmaps[6] = new wxBitmap( print_xpm );
    toolBarBitmaps[7] = new wxBitmap( help_xpm );
#endif

#ifdef __WXMSW__
    int width = 24;
#else
    int width = 16;
#endif
    int currentX = 5;

    toolBar->AddTool(wxID_NEW, *(toolBarBitmaps[0]), wxNullBitmap,
false, currentX, -1, (wxObject *) NULL, "New file");
    currentX += width + 5;
    toolBar->AddTool(wxID_OPEN, *(toolBarBitmaps[1]), wxNullBitmap,
false, currentX, -1, (wxObject *) NULL, "Open file");
    currentX += width + 5;
    toolBar->AddTool(wxID_SAVE, *(toolBarBitmaps[2]), wxNullBitmap,
false, currentX, -1, (wxObject *) NULL, "Save file");
    currentX += width + 5;
    toolBar->AddSeparator();
    toolBar->AddTool(wxID_COPY, *(toolBarBitmaps[3]), wxNullBitmap,
false, currentX, -1, (wxObject *) NULL, "Copy");
    currentX += width + 5;
    toolBar->AddTool(wxID_CUT, *(toolBarBitmaps[4]), wxNullBitmap,
false, currentX, -1, (wxObject *) NULL, "Cut");
    currentX += width + 5;
    toolBar->AddTool(wxID_PASTE, *(toolBarBitmaps[5]), wxNullBitmap,
false, currentX, -1, (wxObject *) NULL, "Paste");
```

```
        currentX += width + 5;
        toolBar->AddSeparator();
        toolBar->AddTool(wxID_PRINT, *(toolBarBitmaps[6]), wxNullBitmap,
false, currentX, -1, (wxObject *) NULL, "Print");
        currentX += width + 5;
        toolBar->AddSeparator();
        toolBar->AddTool(wxID_HELP, *(toolBarBitmaps[7]), wxNullBitmap,
false, currentX, -1, (wxObject *) NULL, "Help");

        toolBar->Realize();

        // Can delete the bitmaps since they're reference counted
        int i;
        for (i = 0; i < 8; i++)
            delete toolBarBitmaps[i];

        return true;
    }

    // wxID_HELP will be processed for the 'About' menu and the toolbar
    help button.

BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(wxID_EXIT, MyFrame::OnQuit)
    EVT_MENU(wxID_HELP, MyFrame::OnAbout)
    EVT_CLOSE(MyFrame::OnCloseWindow)
    EVT_TOOL_RANGE(wxID_OPEN, wxID_PASTE, MyFrame::OnToolLeftClick)
    EVT_TOOL_ENTER(wxID_OPEN, MyFrame::OnToolEnter)
END_EVENT_TABLE()

// Define my frame constructor
MyFrame::MyFrame(wxFrame* parent, wxWindowID id, const wxString&
title, const wxPoint& pos,
                const wxSize& size, long style):
    wxFrame(parent, id, title, pos, size, style)
{
    m_textWindow = new wxTextCtrl(this, -1, "", wxPoint(0, 0), wxSize(-1,
-1), wxTE_MULTILINE);
}

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

void MyFrame::OnAbout(wxCommandEvent& WXUNUSED(event))
{
    (void)wxMessageBox("wxWidgets toolbar sample", "About
wxToolBar");
}

// Define the behaviour for the frame closing
// - must delete all frames except for the main one.
void MyFrame::OnCloseWindow(wxCloseEvent& WXUNUSED(event))
{
    Destroy();
}
```

```
void MyFrame::OnToolLeftClick(wxCommandEvent& event)
{
    wxString str;
    str.Printf("Clicked on tool %d", event.GetId());
    SetStatusText(str);
}

void MyFrame::OnToolEnter(wxCommandEvent& event)
{
    if (event.GetSelection() > -1)
    {
        wxString str;
        str.Printf("This is tool number %d", event.GetSelection());
        SetStatusText(str);
    }
    else
        SetStatusText("");
}
```

## wxGrid classes overview

Classes: *wxGrid* (p. 735)

### Introduction

*wxGrid* and its related classes are used for displaying and editing tabular data.

### Getting started: a simple example

For simple applications you need only refer to the *wxGrid* class in your code. This example shows how you might create a grid in a frame or dialog constructor and illustrates some of the formatting functions.

```
// Create a wxGrid object

grid = new wxGrid( this,
                  -1,
                  wxPoint( 0, 0 ),
                  wxSize( 400, 300 ) );

// Then we call CreateGrid to set the dimensions of the grid
// (100 rows and 10 columns in this example)
grid->CreateGrid( 100, 10 );

// We can set the sizes of individual rows and columns
// in pixels
grid->SetRowSize( 0, 60 );
grid->SetColSize( 0, 120 );

// And set grid cell contents as strings
```

```
grid->SetCellValue( 0, 0, "wxGrid is good" );

// We can specify that some cells are read-only
grid->SetCellValue( 0, 3, "This is read-only" );
grid->SetReadOnly( 0, 3 );

// Colours can be specified for grid cell contents
grid->SetCellValue(3, 3, "green on grey");
grid->SetCellTextColour(3, 3, *wxGREEN);
grid->SetCellBackgroundColour(3, 3, *wxLIGHT_GREY);

// We can specify the some cells will store numeric
// values rather than strings. Here we set grid column 5
// to hold floating point values displayed with width of 6
// and precision of 2
grid->SetColFormatFloat(5, 6, 2);
grid->SetCellValue(0, 6, "3.1415");
```

## A more complex example

Yet to be written

## How the wxGrid classes relate to each other

Yet to be written

## Keyboard and mouse actions

Yet to be written

## wxTipProvider overview

Many "modern" Windows programs have a feature (some would say annoyance) of presenting the user tips at program startup. While this is probably useless to the advanced users of the program, the experience shows that the tips may be quite helpful for the novices and so more and more programs now do this.

For a wxWidgets programmer, implementing this feature is extremely easy. To show a tip, it is enough to just call *wxShowTip* (p. 1943) function like this:

```
if ( ...show tips at startup?... )
{
    wxTipProvider *tipProvider =
wxCreateFileTipProvider("tips.txt", 0);
    wxShowTip(windowParent, tipProvider);
    delete tipProvider;
}
```

Of course, you need to get the text of the tips from somewhere - in the example above, the text is supposed to be in the file *tips.txt* from where it is read by the *tip provider*. The tip provider is just an object of a class deriving from *wxTipProvider* (p. 1689). It has to



implement one pure virtual function of the base class: *GetTip* (p. 1690). In the case of the tip provider created by *wxCreateFileTipProvider* (p. 1935), the tips are just the lines of the text file.

If you want to implement your own tip provider (for example, if you wish to hardcode the tips inside your program), you just have to derive another class from *wxTipProvider* and pass a pointer to the object of this class to *wxShowTip* - then you don't need *wxCreateFileTipProvider* at all.

You will probably want to save somewhere the index of the tip last shown - so that the program doesn't always show the same tip on startup. As you also need to remember whether to show tips or not (you shouldn't do it if the user unchecked "Show tips on startup" checkbox in the dialog), you will probably want to store both the index of the last shown tip (as returned by *wxTipProvider::GetCurrentTip* (p. 1690) and the flag telling whether to show the tips at startup at all.

In a *tips.txt* file, lines that begin with a *#* character are considered comments and are automatically skipped. Blank lines and lines only having spaces are also skipped.

You can easily add runtime-translation capacity by placing each line of the *tips.txt* file inside the usual translation macro. For example, your *tips.txt* file would look like this:

```
_( "This is my first tip" )
_( "This is my second tip" )
```

Now add your *tips.txt* file into the list of files that *gettext* searches for translatable strings. The tips will thus get included into your generated *.po* file catalog and be translated at runtime along with the rest of your application's translatable strings. Note1: Each line in the *tips.txt* file needs to strictly begin with exactly the 3 characters of underscore-parenthesis-doublequote, and end with doublequote-parenthesis, as shown above. Note2: Remember to escape any doublequote characters within the tip string with a backslash-doublequote.

See the *dialogs* program in your *samples* folder for a working example inside a program.

## Printing overview

Classes: *wxPrintout* (p. 1214), *wxPrinter* (p. 1211), *wxPrintPreview* (p. 1221), *wxPrinterDC* (p. 1213), *wxPostScriptDC* (p. 1194), *wxPrintDialog* (p. 1206), *wxPrintData* (p. 1200), *wxPrintDialogData* (p. 1207), *wxPageSetupDialog* (p. 1158), *wxPageSetupDialogData* (p. 1159)

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 *wxWidgets* 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.

In most cases, the only class you will need to derive from is *wxPrintout* (p. 1214); all others will be used as-is.

A brief description of each class's role and how they work together follows.

For the special case of printing under Unix, where various different printing backends have to be offered, please have a look at the *Unix printing overview* (p. 2149).

### ***wxPrintout* (p. 1214)**

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 *wxWidgets*. For more details on how *MyPrintout* is defined, please look at the *printout sample code*.

```
case WXPRINT_PRINT:
{
    wxPrinter printer;
    MyPrintout printout("My printout");
    printer.Print(this, &printout, true);
    break;
}
case WXPRINT_PREVIEW:
{
    // Pass two printout objects: for preview, and possible
    printing.
    wxPrintPreview *preview = new wxPrintPreview(new MyPrintout,
    new MyPrintout);
    wxPreviewFrame *frame = new wxPreviewFrame(preview, this, "Demo
    Print Preview", wxPoint(100, 100), wxSize(600, 650));
    frame->Centre(wxBOTH);
    frame->Initialize();
    frame->Show(true);
    break;
}
```

Class *wxPrintout* (p. 1214) assembles the printed page and (using your subclass's overrides) writes requested pages to a *wxDC* (p. 456) that is passed to it. This *wxDC* could be a *wxMemoryDC* (p. 1069) (for displaying the preview image on-screen), a *wxPrinterDC* (p. 1213) (for printing under MSW and Mac), or a *wxPostScriptDC* (p. 1194) (for printing under GTK or generating PostScript output).

The *document/view framework* (p. 2131) creates a default *wxPrintout* object for every view, calling *wxView::OnDraw* to achieve a prepackaged print/preview facility.

If your window classes have a *Draw(wxDC \*dc)* routine to do screen rendering, your *wxPrintout* subclass will typically call those routines to create portions of the image on your printout. Your *wxPrintout* subclass can also make its own calls to its *wxDC* to draw headers, footers, page numbers, etc.

The scaling of the drawn image typically differs from the screen to the preview and printed images. This class provides a set of routines named *FitThisSizeToXXX()*, *MapScreenSizeToXXX()*, and *GetLogicalXXXRect*, which can be used to set the user

scale and origin of the wxPrintout's DC so that your class can easily map your image to the printout without getting into the details of screen and printer PPI and scaling. See the printing sample for examples of how these routines are used.

### **wxPrinter (p. 1211)**

Class wxPrinter encapsulates the platform-dependent print function with a common interface. In most cases, you will not need to derive a class from wxPrinter; simply create a wxPrinter object in your Print function as in the example above.

### **wxPrintPreview (p. 1221)**

Class wxPrintPreview manages the print preview process. Among other things, it constructs the wxDCs that get passed to your wxPrintout subclass for printing and manages the display of multiple pages, a zoomable preview image, and so forth. In most cases you will use this class as-is, but you can create your own subclass, for example, to change the layout or contents of the preview window.

### **wxPrinterDC (p. 1213)**

Class wxPrinterDC is the wxDC that represents the actual printed page under MSW and Mac. During printing, an object of this class will be passed to your derived wxPrintout object to draw upon. The size of the wxPrinterDC will depend on the paper orientation and the resolution of the printer.

There are two important rectangles in printing: the *page rectangle* defines the printable area seen by the application, and under MSW and Mac, it is the printable area specified by the printer. (For PostScript printing, the page rectangle is the entire page.) The inherited function `wxDC::GetSize` (p. 468) returns the page size in device pixels. The point (0,0) on the wxPrinterDC represents the top left corner of the page rectangle; that is, the page rect is given by `wxRect(0, 0, w, h)`, where (w,h) are the values returned by `GetSize`.

The *paper rectangle*, on the other hand, represents the entire paper area including the non-printable border. Thus, the coordinates of the top left corner of the paper rectangle will have small negative values, while the width and height will be somewhat larger than that of the page rectangle. The wxPrinterDC-specific function `wxPrinterDC::GetPaperRect` (p. 1214) returns the paper rectangle of the given wxPrinterDC.

### **wxPostScriptDC (p. 1194)**

Class wxPostScriptDC is the wxDC that represents the actual printed page under GTK and other PostScript printing. During printing, an object of this class will be passed to your derived wxPrintout object to draw upon. The size of the wxPostScriptDC will depend upon the `wxPrintData` (p. 1200) used to construct it.

Unlike a wxPrinterDC, there is no distinction between the page rectangle and the paper rectangle in a wxPostScriptDC; both rectangles are taken to represent the entire sheet of paper.

### **wxPrintDialog (p. 1206)**

Class `wxPrintDialog` puts up the standard print dialog, which allows you to select the page range for printing (as well as many other print settings, which may vary from platform to platform). You provide an object of type `wxPrintDialogData` (p. 1207) to the `wxPrintDialog` at construction, which is used to populate the dialog.

### **`wxPrintData` (p. 1200)**

Class `wxPrintData` is a subset of `wxPrintDialogData` that is used (internally) to initialize a `wxPrinterDC` or `wxPostScriptDC`. (In fact, a `wxPrintData` is a data member of a `wxPrintDialogData` and a `wxPageSetupDialogData`). Essentially, `wxPrintData` contains those bits of information from the two dialogs necessary to configure the `wxPrinterDC` or `wxPostScriptDC` (e.g., size, orientation, etc.). You might wish to create a global instance of this object to provide call-to-call persistence to your application's print settings.

### **`wxPrintDialogData` (p. 1207)**

Class `wxPrintDialogData` contains the settings entered by the user in the print dialog. It contains such things as page range, number of copies, and so forth. In most cases, you won't need to access this information; the framework takes care of asking your `wxPrintout` derived object for the pages requested by the user.

### **`wxPageSetupDialog` (p. 1158)**

Class `wxPageSetupDialog` puts up the standard page setup dialog, which allows you to specify the orientation, paper size, and related settings. You provide it with a `wxPageSetupDialogData` object at initialization, which is used to populate the dialog; when the dialog is dismissed, this object contains the settings chosen by the user, including orientation and/or page margins.

Note that on Macintosh, the native page setup dialog does not contain entries that allow you to change the page margins. You can use the Mac-specific class `wxMacPageMarginsDialog` (which, like `wxPageSetupDialog`, takes a `wxPageSetupDialogData` object in its constructor) to provide this capability; see the printing sample for an example.

### **`wxPageSetupDialogData` (p. 1159)**

Class `wxPageSetupDialogData` contains settings affecting the page size (paper size), orientation, margins, and so forth. Note that not all platforms populate all fields; for example, the MSW page setup dialog lets you set the page margins while the Mac setup dialog does not.

You will typically create a global instance of each of a `wxPrintData` and `wxPageSetupDialogData` at program initiation, which will contain the default settings provided by the system. Each time the user calls up either the `wxPrintDialog` or the `wxPageSetupDialog`, you pass these data structures to initialize the dialog values and to be updated by the dialog. The framework then queries these data structures to get information like the printed page range (from the `wxPrintDialogData`) or the paper size and/or page orientation (from the `wxPageSetupDialogData`).

## Printing under Unix (GTK+)

Printing under Unix has always been a cause of problems as Unix does not provide a standard way to display text and graphics on screen and print it to a printer using the same application programming interface - instead, displaying on screen is done via the X11 library while printing has to be done with using PostScript commands. This was particularly difficult to handle for the case of fonts with the result that only a selected number of application could offer WYSIWYG under Unix. Equally, wxWidgets offered its own printing implementation using PostScript which never really matched the screen display.

Starting with version 2.8.X, the GNOME project provides printing support through the `libgnomeprint` and `libgnomeprintui` libraries by which especially the font problem is mostly solved. Beginning with version 2.5.4, the GTK+ port of wxWidgets can make use of these libraries if wxWidgets is configured accordingly and if the libraries are present. You need to configure wxWidgets with the `configure --with-gnomeprint` switch and your application will then search for the GNOME print libraries at runtime. If they are found, printing will be done through these, otherwise the application will fall back to the old PostScript printing code. Note that the application will not require the GNOME print libraries to be installed in order to run (there will be no dependency on these libraries).

In version GTK+ 2.10, support for printing has been added to GTK+ itself. Support for this has been added to wxWidgets in the development branch and it will be available for wxWidgets 3.0.

## Multithreading overview

Classes: *wxThread* (p. 1670), *wxMutex* (p. 1131), *wxCriticalSection* (p. 294), *wxCondition* (p. 259)

wxWidgets provides a complete set of classes encapsulating objects necessary in multithreaded (MT) programs: the *thread* (p. 1670) class itself and different synchronization objects: *mutexes* (p. 1131) and *critical sections* (p. 294) with *conditions* (p. 259). The thread API in wxWidgets resembles to POSIX1.c threads API (a.k.a. pthreads), although several functions have different names and some features inspired by Win32 thread API are there as well.

These classes will hopefully make writing MT programs easier and they also provide some extra error checking (compared to the native (be it Win32 or Posix) thread API), however it is still a non-trivial undertaking especially for large projects. Before starting an MT application (or starting to add MT features to an existing one) it is worth asking oneself if there is no easier and safer way to implement the same functionality. Of course, in some situations threads really make sense (classical example is a server application which launches a new thread for each new client), but in others it might be a very poor choice (example: launching a separate thread when doing a long computation to show a progress dialog). Other implementation choices are available: for the progress dialog example it is far better to do the calculations in the *idle handler* (p. 903) or even simply do everything at once but call *wxWindow::Update()* (p. 1852) periodically to update the screen.

If you do decide to use threads in your application, it is strongly recommended that no

more than one thread calls GUI functions. The thread sample shows that it *is* possible for many different threads to call GUI functions at once (all the threads created in the sample access GUI), but it is a very poor design choice for anything except an example. The design which uses one GUI thread and several worker threads which communicate with the main one using events is much more robust and will undoubtedly save you countless problems (example: under Win32 a thread can only access GDI objects such as pens, brushes, &c created by itself and not by the other threads).

For communication between secondary threads and the main thread, you may use `wxEvtHandler::AddPendingEvent` (p. 576) or its short version `wxPostEvent` (p. 1960). These functions have a thread-safe implementation so that they can be used as they are for sending events from one thread to another. However there is no built in method to send messages to the worker threads and you will need to use the available synchronization classes to implement the solution which suits your needs yourself. In particular, please note that it is *not* enough to derive your class from `wxThread` (p. 1670) and `wxEvtHandler` (p. 576) to send messages to it: in fact, this does *not* work at all.

## Drag and drop overview

Classes: `wxDataObject` (p. 311), `wxTextDataObject` (p. 1653), `wxDropSource` (p. 558), `wxDropTarget` (p. 561), `wxTextDropTarget` (p. 1654), `wxFileDropTarget` (p. 604)

Note that `wxUSE_DRAG_AND_DROP` must be defined in `setup.h` in order to use drag and drop in `wxWidgets`.

See also: *wxDataObject overview* (p. 2151) and *DnD sample* (p. 2033)

It may be noted that data transfer to and from the clipboard is quite similar to data transfer with drag and drop and the code to implement these two types is almost the same. In particular, both data transfer mechanisms store data in some kind of `wxDataObject` (p. 311) and identify its format(s) using the `wxDataFormat` (p. 304) class.

To be a *drag source*, i.e. to provide the data which may be dragged by the user elsewhere, you should implement the following steps:

- **Preparation:** First of all, a data object must be created and initialized with the data you wish to drag. For example:

```
wxTextDataObject my_data("This text will be dragged.");
```

- **Drag start:** To start the dragging process (typically in response to a mouse click) you must call `wxDropSource::DoDragDrop` (p. 560) like this:

```
wxDropSource dragSource( this );
dragSource.SetData( my_data );
wxDragResult result = dragSource.DoDragDrop( TRUE );
```

- **Dragging:** The call to `DoDragDrop()` blocks the program until the user releases

the mouse button (unless you override the *GiveFeedback* (p. 560) function to do something special). When the mouse moves in a window of a program which understands the same drag-and-drop protocol (any program under Windows or any program supporting the XDnD protocol under X Windows), the corresponding *wxDropTarget* (p. 561) methods are called - see below.

- **Processing the result:** *DoDragDrop()* returns an *effect code* which is one of the values of *wxDragResult* enum (explained *here* (p. 561)):

```
switch (result)
{
    case wxDragCopy: /* copy the data */ break;
    case wxDragMove: /* move the data */ break;
    default:         /* do nothing */ break;
}
```

To be a *drop target*, i.e. to receive the data dropped by the user you should follow the instructions below:

- **Initialization:** For a window to be a drop target, it needs to have an associated *wxDropTarget* (p. 561) object. Normally, you will call *wxWindow::SetDropTarget* (p. 1838) during window creation associating your drop target with it. You must derive a class from *wxDropTarget* and override its pure virtual methods. Alternatively, you may derive from *wxTextDropTarget* (p. 1654) or *wxFileDropTarget* (p. 604) and override their *OnDropText()* or *OnDropFiles()* method.
- **Drop:** When the user releases the mouse over a window, *wxWidgets* asks the associated *wxDropTarget* object if it accepts the data. For this, a *wxDataObject* (p. 311) must be associated with the drop target and this data object will be responsible for the format negotiation between the drag source and the drop target. If all goes well, then *OnData* (p. 562) will get called and the *wxDataObject* belonging to the drop target can get filled with data.
- **The end:** After processing the data, *DoDragDrop()* returns either *wxDragCopy* or *wxDragMove* depending on the state of the keys <Ctrl>, <Shift> and <Alt> at the moment of the drop. There is currently no way for the drop target to change this return code.

## wxDataObject overview

Classes: *wxDataObject* (p. 311), *wxClipboard* (p. 195), *wxDataFormat* (p. 304), *wxDropSource* (p. 558), *wxDropTarget* (p. 561)

See also: *Drag and drop overview* (p. 2150) and *DnD sample* (p. 2033)

This overview discusses data transfer through clipboard or drag and drop. In *wxWidgets*, these two ways to transfer data (either between different applications or inside one and the same) are very similar which allows to implement both of them using almost the same

code - or, in other words, if you implement drag and drop support for your application, you get clipboard support for free and vice versa.

At the heart of both clipboard and drag and drop operations lies the *wxDataObject* (p. 311) class. The objects of this class (or, to be precise, classes derived from it) represent the data which is being carried by the mouse during drag and drop operation or copied to or pasted from the clipboard. *wxDataObject* is a "smart" piece of data because it knows which formats it supports (see *GetFormatCount* and *GetAllFormats*) and knows how to render itself in any of them (see *GetDataHere*). It can also receive its value from the outside in a format it supports if it implements the *SetData* method. Please see the documentation of this class for more details.

Both clipboard and drag and drop operations have two sides: the source and target, the data provider and the data receiver. These which may be in the same application and even the same window when, for example, you drag some text from one position to another in a word processor. Let us describe what each of them should do.

### The data provider (source) duties

The data provider is responsible for creating a *wxDataObject* (p. 311) containing the data to be transferred. Then it should either pass it to the clipboard using *SetData* (p. 198) function or to *wxDropSource* (p. 558) and call *DoDragDrop* (p. 560) function.

The only (but important) difference is that the object for the clipboard transfer must always be created on the heap (i.e. using *new*) and it will be freed by the clipboard when it is no longer needed (indeed, it is not known in advance when, if ever, the data will be pasted from the clipboard). On the other hand, the object for drag and drop operation must only exist while *DoDragDrop* (p. 560) executes and may be safely deleted afterwards and so can be created either on heap or on stack (i.e. as a local variable).

Another small difference is that in the case of clipboard operation, the application usually knows in advance whether it copies or cuts (i.e. copies and deletes) data - in fact, this usually depends on which menu item the user chose. But for drag and drop it can only know it after *DoDragDrop* (p. 560) returns (from its return value).

### The data receiver (target) duties

To receive (paste in usual terminology) data from the clipboard, you should create a *wxDataObject* (p. 311) derived class which supports the data formats you need and pass it as argument to *wxClipboard::GetData* (p. 197). If it returns *false*, no data in (any of) the supported format(s) is available. If it returns *true*, the data has been successfully transferred to *wxDataObject*.

For drag and drop case, the *wxDropTarget::OnData* (p. 562) virtual function will be called when a data object is dropped, from which the data itself may be requested by calling *wxDropTarget::GetData* (p. 562) method which fills the data object.

## Database classes overview

Following is a detailed overview of how to use the wxWidgets ODBC classes - *wxDb* (p.



375) and *wxDbTable* (p. 415) and their associated functions. These are the ODBC classes donated by Remstar International, and are collectively referred to herein as the wxODBC classes.

### **wxDb/wxDbTable wxODBC Overview**

Classes: *wxDb* (p. 375), *wxDbTable* (p. 415)

The wxODBC classes were designed for database independence. Although SQL and ODBC both have standards which define the minimum requirements they must support to be in compliance with specifications, different database vendors may implement things slightly differently. One example of this is that Oracle requires all user names for the datasources to be supplied in uppercase characters. In situations like this, the wxODBC classes have been written to make this transparent to the programmer when using functions that require database-specific syntax.

Currently several major databases, along with other widely used databases, have been tested and supported through the wxODBC classes. The list of supported databases is certain to grow as more users start implementing software with these classes, but at the time of the writing of this document, users have successfully used the classes with the following datasources:

- DB2
- DBase (IV, V)\*\*
- Firebird
- INFORMIX
- Interbase
- MS SQL Server (v7 - minimal testing)
- MS Access (97, 2000, 2002, and 2003)
- MySQL (2.x and 3.5 - use the 2.5x drivers though)
- Oracle (v7, v8, v8i)
- Pervasive SQL
- PostgreSQL
- Sybase (ASA and ASE)
- XBase Sequiter
- VIRTUOSO

An up-to-date list can be obtained by looking in the comments of the function *wxDb::Dbms* (p. 386) in *db.cpp*, or in the enumerated type *wxDBMS* (p. 376) in *db.h*.

**\*\*dBase is not truly an ODBC datasource, but there are drivers which can emulate much of the functionality of an ODBC connection to a dBase table. See the *wxODBC Known Issues* (p. 2167) section of this overview for details.**

## **wxODBC Where To Start**

First, if you are not familiar with SQL and ODBC, go to your local bookstore and pick up a good book on each. This documentation is not meant to teach you many details about SQL or ODBC, though you may learn some just from immersion in the subject.

If you have worked with non-SQL/ODBC datasources before, there are some things you will need to un-learn. First some terminology as these phrases will be used heavily in this section of the manual.

Datasource	(usually a database) that contains the data that will be accessed by the wxODBC classes.
Data table	The section of the datasource that contains the rows and columns of data.
ODBC driver	The middle-ware software that interprets the ODBC commands sent by your application and converts them to the SQL format expected by the target datasource.
Datasource connection	An open pipe between your application and the ODBC driver which in turn has a connection to the target datasource. Datasource connections can have a virtually unlimited number of wxDbTable instances using the same connect (dependent on the ODBC driver). A separate connection is not needed for each table (the exception is for isolating commits/rollbacks on different tables from affecting more than the desired table. See the class documentation on <i>wxDb::CommitTrans</i> (p. 384) and <i>wxDb::RollbackTrans</i> (p. 401).)
Rows	Similar to records in old relational databases, a row is a collection of one instance of each column of the data table that are all associated with each other.
Columns	Individual fields associated with each row of a data table.
Query	Request from the client to the datasource asking for the data that matches the requirements specified in the users request. When a query is performed, the datasource performs the lookup of the rows with satisfy the query, and creates a result set.
Result set	The data which matches the requirements specified in a query sent to the datasource. Dependent on drivers, a result set typically remains at the datasource (no data is transmitted to the ODBC driver) until the client actually instructs the ODBC driver to retrieve it.

Cursor	A logical pointer into the result set that a query generates, indicating the next record that will be returned to the client when a request for the next record is made.
Scrolling cursors	Scrolling refers to the movement of cursors through the result set. Cursors can always scroll forward sequentially in the result set (FORWARD ONLY scrolling cursors). With Forward only scrolling cursors, once a row in the result set has been returned to the ODBC driver and on to the client, there is no way to have the cursor move backward in the result set to look at the row that is previous to the current row in the result set. If BACKWARD scrolling cursors are supported by both the ODBC driver and the datasource that are being used, then backward scrolling cursor functions may be used ( <i>wxDbTable::GetPrev</i> (p. 432), <i>wxDbTable::GetFirst</i> (p. 430), and <i>wxDbTable::GetLast</i> (p. 431)). If the datasource or the ODBC driver only support forward scrolling cursors, your program and logic must take this in to account.
Commit/Rollback	Commit will physically save insertions/deletions/updates, while rollback basically does an undo of everything done against the datasource connection that has not been previously committed. Note that Commit and Rollbacks are done on a connection, not on individual tables. All tables which use a shared connection to the datasource are all committed/rolled back at the same time when a call to <i>wxDb::CommitTrans</i> (p. 384) or <i>wxDb::RollbackTrans</i> (p. 401) is made.
Index	Indexes are datasource-maintained lookup structures that allow the datasource to quickly locate data rows based on the values of certain columns. Without indexes, the datasource would need to do a sequential search of a table every time a query request is made. Proper unique key index construction can make datasource queries nearly instantaneous.

Before you are able to read data from a data table in a datasource, you must have a connection to the datasource. Each datasource connection may be used to open multiple tables all on the same connection (number of tables open are dependent on the driver, datasource configuration and the amount of memory on the client workstation). Multiple connections can be opened to the same datasource by the same client (number of concurrent connections is dependent on the driver and datasource configuration).

When a query is performed, the client passes the query to the ODBC driver, and the driver then translates it and passes it along to the datasource. The database engine (in most cases - exceptions are text and dBase files) running on the machine hosting the database does all the work of performing the search for the requested data. The client simply waits for a status to come back through the ODBC driver from the datasource.

Depending on the ODBC driver, the result set either remains "queued" on the database

server side, or is transferred to the machine that the driver is queued on. The client does not receive this data. The client must request some or all of the result set to be returned before any data rows are returned to the client application.

Result sets do not need to include all columns of every row matching the query. In fact, result sets can actually be joinings of columns from two or more data tables, may have derived column values, or calculated values returned.

For each result set, a cursor is maintained (typically by the database) which keeps track of where in the result set the user currently is. Depending on the database, ODBC driver, and how you configured the wxWidgets ODBC settings in `setup.h` (see *wxODBC - Compiling* (p. 2157)), cursors can be either forward or backward scrolling. At a minimum, cursors must scroll forward. For example, if a query resulted in a result set with 100 rows, as the data is read by the client application, it will read row 1, then 2, then 3, etc. With forward only cursors, once the cursor has moved to the next row, the previous row cannot be accessed again without re-querying the datasource for the result set over again. Backward scrolling cursors allow you to request the previous row from the result set, actually scrolling the cursor backward.

Backward scrolling cursors are not supported on all database/driver combinations. For this reason, forward-only cursors are the default in the wxODBC classes. If your datasource does support backward scrolling cursors and you wish to use them, make the appropriate changes in `setup.h` to enable them (see *wxODBC - Compiling* (p. 2157)). For greatest portability between datasources, writing your program in such a way that it only requires forward scrolling cursors is your best bet. On the other hand, if you are focusing on using only datasources that support backward scrolling cursors, potentially large performance benefits can be gained from using them.

There is a limit to the number of cursors that can be open on each connection to the datasource, and usually a maximum number of cursors for the datasource itself. This is all dependent on the database. Each connection that is opened (each instance of a `wxDb`) opens a minimum of 5 cursors on creation that are required for things such as updates/deletions/rollbacks/queries. Cursors are a limited resource, so use care in creating large numbers of cursors.

Additional cursors can be created if necessary with the `wxDbTable::GetNewCursor` (p. 431) function. One example use for additional cursors is to track multiple scroll points in result sets. By creating a new cursor, a program could request a second result set from the datasource while still maintaining the original cursor position in the first result set.

Different than non-SQL/ODBC datasources, when a program performs an insertion, deletion, or update (or other SQL functions like altering tables, etc) through ODBC, the program must issue a "commit" to the datasource to tell the datasource that the action(s) it has been told to perform are to be recorded as permanent. Until a commit is performed, any other programs that query the datasource will not see the changes that have been made (although there are databases that can be configured to auto-commit). NOTE: With most datasources, until the commit is performed, any cursor that is open on that same datasource connection will be able to see the changes that are uncommitted. Check your database's documentation/configuration to verify this before relying on it though.

A rollback is basically an UNDO command on the datasource connection. When a rollback is issued, the datasource will flush all commands it has been told to do since the

last commit that was performed.

NOTE: Commits/Rollbacks are done on datasource connections (wxDb instances) not on the wxDbTable instances. This means that if more than one table shares the same connection, and a commit or rollback is done on that connection, all pending changes for ALL tables using that connection are committed/rolled back.

### **wxODBC - Configuring your system for ODBC use**

Before you are able to access a datasource, you must have installed and configured an ODBC driver. Doing this is system specific, so it will not be covered in detail here. But here are a few details to get you started.

Most database vendors provide at least a minimal ODBC driver with their database product. In practice, many of these drivers have proven to be slow and/or incomplete. Rumour has it that this is because the vendors do not want you using the ODBC interface to their products; they want you to use their applications to access the data.

Whatever the reason, for database-intensive applications, you may want to consider using a third-party ODBC driver for your needs. One example of a third-party set of ODBC drivers that has been heavily tested and used is Rogue Wave's drivers. Rogue Wave has drivers available for many different platforms and databases. Under Microsoft Windows, install the ODBC driver you are planning to use. You will then use the ODBC Administrator in the Control Panel to configure an instance of the driver for your intended datasource. Note that with all flavors of NT, this configuration can be set up as a System or User DSN (datasource name). Configuring it as a system resource will make it available to all users (if you are logged in as 'administrator'), otherwise the datasource will only be available to the user who configured the DSN.

Under Unix, iODBC is used for implementation of the ODBC API. To compile the wxODBC classes, you must first obtain iODBC from <http://www.iodbc.org> ([www.iodbc.org](http://www.iodbc.org)) and install it. (Note: wxWidgets currently includes a version of iODBC.) Then you must create the file "~/.odbc.ini" (or optionally create "/etc/odbc.ini" for access for all users on the system). This file contains the settings for your system/datasource. Below is an example section of a odbc.ini file for use with the "samples/db" sample program using MySQL:

```
[contacts]
Trace      = Off
TraceFile= stderr
Driver     = /usr/local/lib/libmyodbc.so
DSN        = contacts
SERVER     = 192.168.1.13
USER       = get
PASSWORD   =
PORT       = 3306
```

### **wxODBC - Compiling**

The wxWidgets setup.h file has several settings in it pertaining to compiling the wxODBC classes.

- wxUSE\_ODBC** This must be set to 1 in order for the compiler to compile the wxODBC classes. Without setting this to 1, there will be no access to any of the wxODBC classes. The default is 0.
- wxODBC\_FWD\_ONLY\_CURSORS** When a new database connection is requested, this setting controls the default of whether the connection allows only forward scrolling cursors, or forward and backward scrolling cursors (see the section in "WHERE TO START" on cursors for more information on cursors). This default can be overridden by passing a second parameter to either the *wxDbGetConnection* (p. 380) or *wxDb constructor* (p. 382). The default is 1.
- wxODBC\_BACKWARD\_COMPATABILITY** Between v2.0 and 2.2, massive renaming efforts were done to the ODBC classes to get naming conventions similar to those used throughout wxWidgets, as well as to preface all wxODBC classes names and functions with a wxDb preface. Because this renaming would affect applications written using the v2.0 names, this compile-time directive was added to allow those programs written for v2.0 to still compile using the old naming conventions. These deprecated names are all #define'd to their corresponding new function names at the end of the db.cpp/dbtable.cpp source files. These deprecated class/function names should not be used in future development, as at some point in the future they will be removed. The default is 0.

#### *Under MS Windows*

You are required to include the "odbc32.lib" provided by your compiler vendor in the list of external libraries to be linked in. If using the makefiles supplied with wxWidgets, this library should already be included for use with makefile.b32, makefile.vc, and makefile.g95.

MORE TO COME
--------------

#### *Under Unix--with-odbc flag for configure*

MORE TO COME
--------------

### **wxODBC - Basic Step-By-Step Guide**

To use the classes in an application, there are eight basic steps:

- Define datasource connection information
- Get a datasource connection
- Create table definition
- Open the table

- Use the table
- Close the table
- Close the datasource connection
- Release the ODBC environment handle

Following each of these steps is detailed to explain the step, and to hopefully mention as many of the pitfalls that beginning users fall in to when first starting to use the classes. Throughout the steps, small snippets of code are provided to show the syntax of performing the step. A complete code snippet is provided at the end of this overview that shows a complete working flow of all these steps (see *wxODBC - Sample Code* (p. 2169)).

### **Define datasource connection information**

To be able to connect to a datasource through the ODBC driver, a program must supply a minimum of three pieces of information: Datasource name, User ID, and Authorization string (password). A fourth piece of information, a default directory indicating where the data file is stored, is required for Text and dBase drivers for ODBC.

The wxWidgets data class wxDbConnectInf exists for holding all of these values, plus some others that may be desired.

The 'Henv' member is the environment handle used to access memory for use by the ODBC driver. Use of this member is described below in the "Getting a Connection to the Datasource" section.

The 'Dsn' must exactly match the datasource name used to configure the ODBC datasource (in the ODBC Administrator (MSW only) or in the .odbc.ini file).

The 'Uid' is the User ID that is to be used to log in to the datasource. This User ID must already have been created and assigned rights within the datasource to which you are connecting. The user that the connection is establish by will determine what rights and privileges the datasource connection will allow the program to have when using the connection that this connection information was used to establish. Some datasources are case sensitive for User IDs, and though the wxODBC classes attempt to hide this from you by manipulating whatever data you pass in to match the datasource's needs, it is always best to pass the 'Uid' in the case that the datasource requires.

The 'AuthStr' is the password for the User ID specified in the 'Uid' member. As with the 'Uid', some datasources are case sensitive (in fact most are). The wxODBC classes do NOT try to manage the case of the 'AuthStr' at all. It is passed verbatim to the datasource, so you must use the case that the datasource is expecting.

The 'defaultDir' member is used with file based datasources (i.e. dBase, FoxPro, text files). It contains a full path to the location where the data table or file is located. When setting this value, use forward slashes '/' rather than backslashes ' avoid compatibility differences between ODBC drivers.

The other fields are currently unused. The intent of these fields are that they will be used to write our own ODBC Administrator type program that will work on both MSW and Un\*x

systems, regardless of the datasource. Very little work has been done on this to date.

### Get a Datasource Connection

There are two methods of establishing a connection to a datasource. You may either manually create your own wxDb instance and open the connection, or you may use the caching functions provided with the wxODBC classes to create/maintain/delete the connections.

Regardless of which method you use, you must first have a fully populated wxDbConnectInf object. In the wxDbConnectInf instance, provide a valid Dns, Uid, and AuthStr (along with a 'defaultDir' if necessary). Before using this though, you must allocate an environment handle to the 'Henv' member.

```
wxDbConnectInf DbConnectInf;  
DbConnectInf.SetDsn("MyDSN");  
DbConnectInf.SetUserID("MyUserName");  
DbConnectInf.SetPassword("MyPassword");  
DbConnectInf.SetDefaultDir("");
```

To allocate an environment handle for the ODBC connection to use, the wxDbConnectInf class has a datasource independent method for creating the necessary handle:

```
if (DbConnectInf.AllocHenv())  
{  
    wxMessageBox("Unable to allocate an ODBC environment handle",  
        "DB CONNECTION ERROR", wxOK |  
wxICON_EXCLAMATION);  
    return;  
}
```

When the wxDbConnectInf::AllocHenv() function is called successfully, a value of true will be returned. A value of false means allocation failed, and the handle will be undefined.

A shorter form of doing the above steps is encapsulated into the long form of the constructor for wxDbConnectInf.

```
wxDbConnectInf *DbConnectInf;  
  
DbConnectInf = new wxDbConnectInf(NULL, "MyDSN",  
    "MyUserName",  
                                "MyPassword", "");
```

This shorthand form of initializing the constructor passes a NULL for the SQL environment handle, telling the constructor to allocate a handle during construction. This handle is also managed for the life of wxDbConnectInf instance, and is freed automatically upon destruction of the instance.

Once the wxDbConnectInf instance is initialized, you are ready to connect to the datasource.

To manually create datasource connections, you must create a wxDb instance, and then open it.

```
wxDb *db = new wxDb(DbConnectInf->GetHenv());
```



```
opened = db->Open(DbConnectInf);
```

The first line does the house keeping needed to initialize all the members of the *wxDb* class. The second line actually sends the request to the ODBC driver to open a connection to its associated datasource using the parameters supplied in the call to *wxDb::Open* (p. 399).

A more advanced form of opening a connection is to use the connection caching functions that are included with the *wxODBC* classes. The caching mechanisms perform the same functions as the manual approach to opening a connection, but they also manage each connection they have created, re-using them and cleaning them up when they are closed, without you needing to do the coding.

To use the caching function *wxDbGetConnection* (p. 380) to get a connection to a datasource, simply call it with a single parameter of the type *wxDbConnectInf*:

```
db = wxDbGetConnection(DbConnectInf);
```

The *wxDb* pointer that is returned is both initialized and opened. If something failed in creating or opening the connection, the return value from *wxDbGetConnection* (p. 380) will be *NULL*.

The connection that is returned is either a new connection, or it is a "free" connection from the cache of connections that the class maintains that was no longer in use. Any *wxDb* instance created with a call to *wxDbGetConnection* (p. 380) is recorded in a linked list of established connections. When a program is finished with a connection, a call to *wxDbFreeConnection* (p. 380) is made, and the datasource connection will then be tagged as *FREE*, making it available for the next call to *wxDbGetConnection* (p. 380) that needs a connection using the same connection information (*Dsn*, *Uid*, *AuthStr*). The cached connections remain cached until a call to *wxDbCloseConnections* (p. 380) is made, at which time all cached connections are closed and deleted.

Besides the obvious advantage of using the single command caching routine to obtain a datasource connection, using cached connections can be quite a performance boost as well. Each time that a new connection is created (not retrieved from the cache of free connections), the *wxODBC* classes perform many queries against the datasource to determine the datasource's datatypes and other fundamental behaviours. Depending on the hardware, network bandwidth, and datasource speed, this can in some cases take a few seconds to establish the new connection (with well-balanced systems, it should only be a fraction of a second). Re-using already established datasource connections rather than creating/deleting, creating/deleting connections can be quite a time-saver.

Another time-saver is the "copy connection" features of both *wxDb::Open* (p. 399) and *wxDbGetConnection* (p. 380). If manually creating a *wxDb* instance and opening it, you must pass an existing connection to the *wxDb::Open* (p. 399) function yourself to gain the performance benefit of copying existing connection settings. The *wxDbGetConnection* (p. 380) function automatically does this for you, checking the *Dsn*, *Uid*, and *AuthStr* parameters when you request a connection for any existing connections that use those same settings. If one is found, *wxDbGetConnection* (p. 380) copies the datasource settings for datatypes and other datasource specific information that was previously queried, rather than re-querying the datasource for all those same settings.

One final note on creating a connection. When a connection is created, it will default to only allowing cursor scrolling to be either forward only, or both backward and forward scrolling. The default behavior is determined by the setting

`wxODBC_FWD_ONLY_CURSORS` in `setup.h` when you compile the `wxWidgets` library. The library default is to only support forward scrolling cursors only, though this can be overridden by parameters for `wxDb()` constructor or the `wxDbGetConnection` (p. 380) function. All datasources and ODBC drivers must support forward scrolling cursors. Many datasources support backward scrolling cursors, and many ODBC drivers support backward scrolling cursors. Before planning on using backward scrolling cursors, you must be certain that both your datasource and ODBC driver fully support backward scrolling cursors. See the small blurb about "Scrolling cursors" in the definitions at the beginning of this overview, or other details of setting the cursor behavior in the `wxDb` class documentation.

### Create Table Definition

Data can be accessed in a datasource's tables directly through various functions of the `wxDb` class (see `wxDb::GetData` (p. 392)). But to make life much simpler, the `wxDbTable` class encapsulates all of the SQL specific API calls that would be necessary to do this, wrapping it in an intuitive class of APIs.

The first step in accessing data in a datasource's tables via the `wxDbTable` class is to create a `wxDbTable` instance.

```
table = new wxDbTable(db, tableName, numTableColumns, "",
                     !wxDB_QUERY_ONLY, "");
```

When you create the instance, you indicate the previously established datasource connection to be used to access the table, the name of the primary table that is to be accessed with the datasource's tables, how many columns of each row are going to be returned, the name of the view of the table that will actually be used to query against (works with Oracle only at this time), whether the data returned is for query purposes only, and finally the path to the table, if different than the path specified when connecting to the datasource.

Each of the above parameters are described in detail in the `wxDbTable` class' description, but one special note here about the fifth parameter - the `queryOnly` setting. If a `wxDbTable` instance is created as `wxDB_QUERY_ONLY`, then no inserts/deletes/updates can be performed using this instance of the `wxDbTable`. Any calls to `wxDb::CommitTrans` (p. 384) or `wxDb::RollbackTrans` (p. 401) against the datasource connection used by this `wxDbTable` instance are ignored by this instance. If the `wxDbTable` instance is created with `!wxDB_QUERY_ONLY` as shown above, then all the cursors and other overhead associated with being able to insert/update/delete data in the table are created, and thereby those operations can then be performed against the associated table with this `wxDbTable` instance.

If a table is to be accessed via a `wxDbTable` instance, and the table will only be read from, not written to, there is a performance benefit (not as many cursors need to be maintained/updated, hence speeding up access times), as well as a resource savings due to fewer cursors being created for the `wxDbTable` instance. Also, with some datasources, the number of simultaneous cursors is limited.

When defining the columns to be retrievable by the `wxDbTable` instance, you can specify anywhere from one column up to all columns in the table.

```
table->SetColDefs(0, "FIRST_NAME", DB_DATA_TYPE_VARCHAR,
FirstName,
                SQL_C_WXCHAR, sizeof(FirstName), true, true);
table->SetColDefs(1, "LAST_NAME", DB_DATA_TYPE_VARCHAR,
LastName,
                SQL_C_WXCHAR, sizeof(LastName), true, true);
```

Notice that column definitions start at index 0 and go up to one less than the number of columns specified when the `wxDbTable` instance was created (in this example, two columns - one with index 0, one with index 1).

The above lines of code "bind" the datasource columns specified to the memory variables in the client application. So when the application makes a call to `wxDbTable::GetNext` (p. 432) (or any other function that retrieves data from the result set), the variables that are bound to the columns will have the column value stored into them. See the `wxDbTable::SetColDefs` (p. 443) class documentation for more details on all the parameters for this function.

The bound memory variables have undefined data in them until a call to a function that retrieves data from a result set is made (e.g. `wxDbTable::GetNext` (p. 432), `wxDbTable::GetPrev` (p. 432), etc). The variables are not initialized to any data by the `wxODBC` classes, and they still contain undefined data after a call to `wxDbTable::Query` (p. 437). Only after a successful call to one of the `::GetXxxx()` functions is made do the variables contain valid data.

It is not necessary to define column definitions for columns whose data is not going to be returned to the client. For example, if you want to query the datasource for all users with a first name of 'GEORGE', but you only want the list of last names associated with those rows (why return the `FIRST_NAME` column every time when you already know it is 'GEORGE'), you would only have needed to define one column above.

You may have as many `wxDbTable` instances accessing the same table using the same `wxDb` instance as you desire. There is no limit imposed by the classes on this. All datasources supported (so far) also have no limitations on this.

### Open the table

Opening the table is not technically doing anything with the datasource itself. Calling `wxDbTable::Open` (p. 436) simply does all the housekeeping of checking that the specified table exists, that the current connected user has at least `SELECT` privileges for accessing the table, setting up the requisite cursors, binding columns and cursors, and constructing the default `INSERT` statement that is used when a new row is inserted into the table (non-`wxDB_QUERY_ONLY` tables only).

```
if (!table->Open())
{
    // An error occurred opening (setting up) the table
}
```

The only reason that a call to `wxDbTable::Open` (p. 436) is likely to fail is if the user has

insufficient privileges to even SELECT the table. Other problems could occur, such as being unable to bind columns, but these other reason point to some lack of resource (like memory). Any errors generated internally in the *wxDbTable::Open* (p. 436) function are logged to the error log if SQL logging is turned on for the classes.

### Use the table

To use the table and the definitions that are now set up, we must first define what data we want the datasource to collect in to a result set, tell it where to get the data from, and in which sequence we want the data returned.

```
// the WHERE clause limits/specifies which rows in the table
// are to be returned in the result set
table->SetWhereClause("FIRST_NAME = 'GEORGE'");

// Result set will be sorted in ascending alphabetical
// order on the data in the 'LAST_NAME' column of each row
// If the same last name is in the table for two rows,
// sub-sort on the 'AGE' column
table->SetOrderByClause("LAST_NAME, AGE");

// No other tables (joins) are used for this query
table->SetFromClause("");
```

The above lines will be used to tell the datasource to return in the result all the rows in the table whose column "FIRST\_NAME" contains the name 'GEORGE' (note the required use of the single quote around the string literal) and that the result set will return the rows sorted by ascending last names (ascending is the default, and can be overridden with the "DESC" keyword for datasources that support it - "LAST\_NAME DESC").

Specifying a blank WHERE clause will result in the result set containing all rows in the datasource.

Specifying a blank ORDERBY clause means that the datasource will return the result set in whatever sequence it encounters rows which match the selection criteria. What this sequence is can be hard to determine. Typically it depends on the index that the datasource used to find the rows which match the WHERE criteria. BEWARE - relying on the datasource to return data in a certain sequence when you have not provided an ORDERBY clause will eventually cause a problem for your program. Databases can be tuned to be COST-based, SPEED-based, or some other basis for how it gets your result set. In short, if you need your result set returned in a specific sequence, ask for it that way by providing an ORDERBY clause.

Using an ORDERBY clause can be a performance hit, as the database must sort the items before making the result set available to the client. Creating efficient indexes that cause the data to be "found" in the correct ORDERBY sequence can be a big performance benefit. Also, in the large majority of cases, the database will be able to sort the records faster than your application can read all the records in (unsorted) and then sort them. Let the database do the work for you!

Notice in the example above, a column that is not included in the bound data columns ('AGE') will be used to sub-sort the result set.

The FROM clause in this example is blanked, as we are not going to be performing any

table joins with this simple query. When the FROM clause is blank, it is assumed that all columns referenced are coming from the default table for the `wxDbTable` instance.

After the selection criteria have been specified, the program can now ask the datasource to perform the search and create a result set that can be retrieved:

```
// Instruct the datasource to perform a query based on the
// criteria specified above in the where/orderby/from clauses.
if (!table->Query())
{
    // An error occurred performing the query
}
```

Typically, when an error occurs when calling `wxDbTable::Query` (p. 437), it is a syntax problem in the WHERE clause that was specified. The exact SQL (datasource-specific) reason for what caused the failure of `wxDbTable::Query` (p. 437) (and all other operations against the datasource can be found by parsing the table's database connection's "errorList[]" array member for the stored text of the error.

When the `wxDbTable::Query` (p. 437) returns true, the database was able to successfully complete the requested query using the provided criteria. This does not mean that there are any rows in the result set, it just mean that the query was successful.

**IMPORTANT:** The result created by the call to `wxDbTable::Query` (p. 437) can take one of two forms. It is either a snapshot of the data at the exact moment that the database determined the record matched the search criteria, or it is a pointer to the row that matched the selection criteria. Which form of behavior is datasource dependent. If it is a snapshot, the data may have changed since the result set was constructed, so beware if your datasource uses snapshots and call `wxDbTable::Refresh` (p. 442). Most larger brand databases do not use snapshots, but it is important to mention so that your application can handle it properly if your datasource does.

To retrieve the data, one of the data fetching routines must be used to request a row from the result set, and to store the data from the result set into the bound memory variables. After `wxDbTable::Query` (p. 437) has completed successfully, the default/current cursor is placed so it is pointing just before the first record in the result set. If the result set is empty (no rows matched the criteria), then any calls to retrieve data from the result set will return false.

```
wxString msg;

while (table->GetNext())
{
    msg.Printf("Row #%lu -- First Name : %s Last Name is %s",
               table->GetRowNum(), FirstName, LastName);
    wxMessageBox(msg, "Data", wxOK | wxICON_INFORMATION, NULL);
}
```

The sample code above will read the next record in the result set repeatedly until the end of the result set has been reached. The first time that `wxDbTable::GetNext` (p. 432) is called right after the successful call to `wxDbTable::Query` (p. 437), it actually returns the first record in the result set.

When `wxDbTable::GetNext` (p. 432) is called and there are no rows remaining in the result

set after the current cursor position, *wxDATABASE::GetNext* (p. 432) (as well as all the other *wxDATABASE::GetXxxx()* functions) will return false.

### Close the table

When the program is done using a *wxDATABASE* instance, it is as simple as deleting the table pointer (or if declared statically, letting the variable go out of scope). Typically the default destructor will take care of all that is required for cleaning up the *wxDATABASE* instance.

```
if (table)
{
    delete table;
    table = NULL;
}
```

Deleting a *wxDATABASE* instance releases all of its cursors, deletes the column definitions and frees the SQL environment handles used by the table (but not the environment handle used by the datasource connection that the *wxDATABASE* instance was using).

### Close the datasource connection

After all tables that have been using a datasource connection have been closed (this can be verified by calling *wxDATABASE::GetTableCount* (p. 395) and checking that it returns 0), then you may close the datasource connection. The method of doing this is dependent on whether the non-caching or caching method was used to obtain the datasource connection.

If the datasource connection was created manually (non-cached), closing the connection is done like this:

```
if (db)
{
    db->Close();
    delete db;
    db = NULL;
}
```

If the program used the *wxDATABASEGetConnection* (p. 380) function to get a datasource connection, the following is the code that should be used to free the connection(s):

```
if (db)
{
    wxDbFreeConnection(db);
    db = NULL;
}
```

Note that the above code just frees the connection so that it can be re-used on the next call the *wxDATABASEGetConnection* (p. 380). To actually dispose of the connection, releasing all of its resources (other than the environment handle), do the following:

```
wxDATABASECloseConnections();
```

### Release the ODBC environment handle

Once all of the connections that used the ODBC environment handle (in this example it was stored in "DbConnectInf.Henv") have been closed, then it is safe to release the environment handle:

```
DbConnectInf->FreeHenv();
```

Or, if the long form of the constructor was used and the constructor was allowed to allocate its own SQL environment handle, leaving scope or destruction of the wxDbConnectInf will free the handle automatically.

```
delete DbConnectInf;
```

Remember to never release this environment handle if there are any connections still using the handle.

## **wxODBC - Known Issues**

As with creating wxWidgets, writing the wxODBC classes was not the simple task of writing an application to run on a single type of computer system. The classes need to be cross-platform for different operating systems, and they also needed to take in to account different database manufacturers and different ODBC driver manufacturers. Because of all the possible combinations of OS/database/drivers, it is impossible to say that these classes will work perfectly with datasource ABC, ODBC driver XYZ, on platform LMN. You may run into some incompatibilities or unsupported features when moving your application from one environment to another. But that is what makes cross-platform programming fun. It also pinpoints one of the great things about open source software. It can evolve!

The most common difference between different database/ODBC driver manufacturers in regards to these wxODBC classes is the lack of standard error codes being returned to the calling program. Sometimes manufacturers have even changed the error codes between versions of their databases/drivers.

In all the tested databases, every effort has been made to determine the correct error codes and handle them in the class members that need to check for specific error codes (such as TABLE DOES NOT EXIST when you try to open a table that has not been created yet). Adding support for additional databases in the future requires adding an entry for the database in the wxDb::Dbms (p. 386) function, and then handling any error codes returned by the datasource that do not match the expected values.

## **Databases**

Following is a list of known issues and incompatibilities that the wxODBC classes have between different datasources. An up to date listing of known issues can be seen in the comments of the source for wxDb::Dbms (p. 386).

### **ORACLE**

- Currently the only database supported by the wxODBC classes to support VIEWS

### **DBASE**

NOTE: dBase is not a true ODBC datasource. You only have access to as much functionality as the driver can emulate.

- Does not support the SQL\_TIMESTAMP structure
- Supports only one cursor and one connect (apparently? with Microsoft driver only?)
- Does not automatically create the primary index if the 'keyField' param of SetColDef is true. The user must create ALL indexes from their program with calls to *wxDbTable::CreateIndex* (p. 423)
- Table names can only be 8 characters long
- Column names can only be 10 characters long
- Currently cannot CREATE a dBase table - bug or limitation of the drivers used??
- Currently cannot insert rows that have integer columns - bug??

#### *SYBASE (all)*

- To lock a record during QUERY functions, the reserved word 'HOLDLOCK' must be added after every table name involved in the query/join if that table's matching record(s) are to be locked
- Ignores the keywords 'FOR UPDATE'. Use the HOLDLOCK functionality described above

#### *SYBASE (Enterprise)*

- If a column is part of the Primary Key, the column cannot be NULL
- Maximum row size is somewhere in the neighborhood of 1920 bytes

#### *mySQL*

- If a column is part of the Primary Key, the column cannot be NULL.
- Cannot support selecting for update [*wxDbTable::CanSelectForUpdate* (p. 420)]. Always returns false.
- Columns that are part of primary or secondary keys must be defined as being NOT NULL when they are created. Some code is added in *wxDbTable::CreateIndex* (p. 423) to try to adjust the column definition if it is not defined correctly, but it is experimental (as of wxWidgets v2.2.1)
- Does not support sub-queries in SQL statements

#### *POSTGRES*

- Does not support the keywords 'ASC' or 'DESC' as of release v6.5.0



- Does not support sub-queries in SQL statements

### *DB2*

- Columns which are part of a primary key must be declared as NOT NULL

### **UNICODE with wxODBC classes**

As of v2.6 of wxWidgets, the wxODBC classes now fully support the compilation and use of the classes in a Unicode build of wxWidgets, assuming the compiler and OS on which the program will be compiled/run is Unicode capable.

The one major difference in writing code that can be compiled in either unicode or non-unicode builds that is specific to the wxODBC classes is to use the SQL\_C\_WXCHAR datatype for string columns rather than SQL\_C\_CHAR or SQL\_C\_WCHAR.

### **wxODBC - Sample Code**

Simplest example of establishing/opening a connection to an ODBC datasource, binding variables to the columns for read/write usage, opening an existing table in the datasource, inserting a record, setting query parameters (where/orderBy/from), querying the datasource, reading each row of the result set, deleting a record, releasing the connection, then cleaning up.

NOTE: Very basic error handling is shown here, to reduce the size of the code and to make it more easily readable. The HandleError() function uses the wxDbLogExtendedErrorMsg() function for retrieving database error messages.

```
//
-----
// HEADERS
//
-----
#include "wx/log.h"          // #included to enable output of messages
                             only
#include "wx/dbtable.h"

//
-----
// FUNCTION USED FOR HANDLING/DISPLAYING ERRORS
//
-----
// Very generic error handling function.
// If a connection to the database is passed in, then we retrieve all
the
// database errors for the connection and add them to the displayed
message
int HandleError(wxString errmsg, wxDb *pDb=NULL)
{
```

```
    // Retrieve all the error message for the errors that occurred
    wxString allErrors;
    if (!pDb == NULL)
        // Get the database errors and append them to the error message
        allErrors = wxDbLogExtendedErrorMsg(errmsg.c_str(), pDb, 0,
0);
    else
        allErrors = errmsg;

    // Do whatever you wish with the error message here
    // wxLogDebug() is called inside wxDbLogExtendedErrorMsg() so
this
    // console program will show the errors in the console window,
    // but these lines will show the errors in RELEASE builds also
    wxFprintf(stderr, wxT("\n%s\n"), allErrors.c_str());
    fflush(stderr);

    return 1;
}

//
-----
// entry point
//
-----

int main(int argc, char **argv)
{
    wxDbConnectInf *DbConnectInf = NULL; // DB connection information

    wxDb          *db          = NULL; // Database connection

    wxDbTable      *table      = NULL; // Data table to access
    const wxChar   tableName[] = wxT("USERS"); // Name of database
    table

    const UWORD    numTableColumns = 2; // Number table columns
    wxChar         FirstName[50+1]; // column data: "FIRST_NAME"
    wxChar         LastName[50+1]; // column data: "LAST_NAME"

    wxString       msg; // Used for display messages

    //
    -----
    -----
    // DEFINE THE CONNECTION HANDLE FOR THE DATABASE
    //
    -----
    -----
    DbConnectInf = new wxDbConnectInf(NULL,
                                     wxT("CONTACTS-SqlServer"),
                                     wxT("sa"),
                                     wxT("abk"));

    // Error checking....
    if (!DbConnectInf || !DbConnectInf->GetHenv())
```

```
{
    return HandleError(wxT("DB ENV ERROR: Cannot allocate ODBC env
handle"));
}

//
-----
// GET A DATABASE CONNECTION
//
-----
db = wxDbGetConnection(DbConnectInf);

if (!db)
{
    return HandleError(wxT("CONNECTION ERROR - Cannot get DB
connection"));
}

//
-----
// DEFINE THE TABLE, AND THE COLUMNS THAT WILL BE ACCESSED
//
-----
table = new wxDbTable(db, tableName, numTableColumns, wxT(""),
                      !wxDB_QUERY_ONLY, wxT(""));
//
// Bind the columns that you wish to retrieve. Note that there must
// be
// 'numTableColumns' calls to SetColDefs(), to match the wxDbTable def
//
// Not all columns need to be bound, only columns whose values are to
// be
// returned back to the client.
//
table->SetColDefs(0, wxT("FIRST_NAME"), DB_DATA_TYPE_VARCHAR,
FirstName,
                SQL_C_WXCHAR, sizeof(FirstName), true, true);
table->SetColDefs(1, wxT("LAST_NAME"), DB_DATA_TYPE_VARCHAR,
LastName,
                SQL_C_WXCHAR, sizeof(LastName), true, true);

//
-----
// CREATE (or RECREATE) THE TABLE IN THE DATABASE
//
-----
if (!table->CreateTable(true)) //NOTE: No CommitTrans is required
{
```

```
        return HandleError(wxT("TABLE CREATION ERROR: "),
table->GetDb());
    }

//
-----
// OPEN THE TABLE FOR ACCESS
//
-----
if (!table->Open())
{
    return HandleError(wxT("TABLE OPEN ERROR: "), table->GetDb());
}

//
-----
// INSERT A NEW ROW INTO THE TABLE
//
-----
wxStrcpy(FirstName, wxT("JULIAN"));
wxStrcpy(LastName, wxT("SMART"));
if (!table->Insert())
{
    return HandleError(wxT("INSERTION ERROR: "), table->GetDb());
}

// Must commit the insert to write the data to the DB
table->GetDb()->CommitTrans();

//
-----
// RETRIEVE ROWS FROM THE TABLE BASED ON SUPPLIED CRITERIA
//
-----
// Set the WHERE clause to limit the result set to return
// all rows that have a value of 'JULIAN' in the FIRST_NAME
// column of the table.
table->SetWhereClause(wxT("FIRST_NAME = 'JULIAN'"));

// Result set will be sorted in ascending alphabetical
// order on the data in the 'LAST_NAME' column of each row
table->SetOrderByClause(wxT("LAST_NAME"));

// No other tables (joins) are used for this query
table->SetFromClause(wxT(""));

// Instruct the datasource to perform a query based on the
// criteria specified above in the where/orderby/from clauses.
```

```
if (!table->Query())
{
    return HandleError(wxT("QUERY ERROR: "), table->GetDb());
}

// Loop through all rows matching the query criteria until
// there are no more records to read
while (table->GetNext())
{
    msg.Printf(wxT("Row #%lu -- First Name : %s Last Name is %s"),
        table->GetRowNum(), FirstName, LastName);

    // Code to display 'msg' here
    wxLogMessage(wxT("\n%s\n"), msg.c_str());
}

//
-----
// DELETE A ROW FROM THE TABLE
//
-----
// Select the row which has FIRST_NAME of 'JULIAN' and LAST_NAME
// of 'SMART', then delete the retrieved row
//
if (!table->DeleteWhere(wxT("FIRST_NAME = 'JULIAN' and LAST_NAME =
'SMART'")))
{
    return HandleError(wxT("DELETION ERROR: "), table->GetDb());
}

// Must commit the deletion to the database
table->GetDb()->CommitTrans();

//
-----
// TAKE CARE OF THE ODBC CLASS INSTANCES THAT WERE BEING USED
//
-----
// If the wxDbTable instance was successfully created
// then delete it as we are done with it now.
wxDELETE(table);

// Free the cached connection
// (meaning release it back in to the cache of datasource
// connections) for the next time a call to wxDbGetConnection()
// is made.
wxDbFreeConnection(db);
db = NULL;

//
```

---

```
-----  
-----  
// CLEANUP BEFORE EXITING APP  
//  
-----  
-----  
// The program is now ending, so we need to close  
// any cached connections that are still being  
// maintained.  
wxDbCloseConnections();  
  
// Release the environment handle that was created  
// for use with the ODBC datasource connections  
wxDELETE(DbConnectInf);  
  
wxUnusedVar(argc); // Here just to prevent compiler warnings  
wxUnusedVar(argv); // Here just to prevent compiler warnings  
  
return 0;  
}
```

## A selection of SQL commands

The following is a very brief description of some common SQL commands, with examples.

### See also

*Database classes overview* (p. 2152)

### 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)  
)
```

### Insert

Inserts records into a table.

Example:

```
INSERT INTO Book  
(BookNumber, CategoryCode, Title)  
VALUES(5, 'HR', 'The Lark Ascending')
```

## 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.

## 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'.

## Interprocess communication overview

Classes: *wxServer* (p. 1431), *wxConnection* (p. 478), *wxClient* (p. 191) *wxWidgets* has a number of different classes to help with interprocess communication and network programming. This section only discusses one family of classes -- the DDE-like protocol --

but here's a list of other useful classes:

- *wxSocketEvent* (p. 1490), *wxSocketBase* (p. 1472), *wxSocketClient* (p. 1488), *wxSocketServer* (p. 1492): classes for the low-level TCP/IP API.
- *wxProtocol* (p. 1235), *wxURL* (p. 1763), *wxFTP* (p. 694), *wxHTTP* (p. 889): classes for programming popular Internet protocols.

wxWidgets' DDE-like protocol is a high-level protocol based on Windows DDE. There are two implementations of this DDE-like protocol: one using real DDE running on Windows only, and another using TCP/IP (sockets) that runs on most platforms. Since the API and virtually all of the behaviour is the same apart from the names of the classes, you should find it easy to switch between the two implementations.

Notice that by including `<wx/ipc.h>` you may define convenient synonyms for the IPC classes: *wxServer* for either *wxDDEServer* or *wxTCPServer* depending on whether DDE-based or socket-based implementation is used and the same thing for *wxClient* and *wxConnection*.

By default, the DDE implementation is used under Windows. DDE works within one computer only. If you want to use IPC between different workstations you should define `wxUSE_DDE_FOR_IPC` as 0 before including this header -- this will force using TCP/IP implementation even under Windows.

The following description refers to wx... but remember that the equivalent wxTCP... and wxDDE... classes can be used in much the same way.

Three classes are central to the DDE-like API:

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 client to the server - both the client and the server use an instance of this class, one per connection. 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 uses *wxClient::MakeConnection* to send a message to the server 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 may be either an integer port identifier in which case an Internet domain socket will be used for the communications or a valid file name (which shouldn't exist and will be deleted afterwards) in which case a Unix domain socket is created.

**SECURITY NOTE:** Using Internet domain sockets is extremely insecure for IPC as there is absolutely no access control for them, use Unix domain sockets whenever possible!

The server then responds and either vetoes the connection or allows it. If allowed, both the



server and client objects create `wxConnection` objects which persist until the connection is closed. The connection object is then used for sending and receiving subsequent messages between client and server - overriding virtual functions in your class derived from `wxConnection` allows you to handle the DDE messages.

To create a working server, the programmer must:

1. Derive a class from `wxConnection`, providing handlers for various messages sent to the server side of a `wxConnection` (e.g. `OnExecute`, `OnRequest`, `OnPoke`). Only the handlers actually required by the application need to be overridden.
2. Derive a class from `wxServer`, overriding `OnAcceptConnection` to accept or reject a connection on the basis of the topic argument. This member must create and return an instance of the derived connection class if the connection is accepted.
3. Create an instance of your server object and call `Create` to activate it, giving it a service name.

To create a working client, the programmer must:

1. Derive a class from `wxConnection`, providing handlers for various messages sent to the client side of a `wxConnection` (e.g. `OnAdvise`). Only the handlers actually required by the application need to be overridden.
2. Derive a class from `wxClient`, overriding `OnMakeConnection` to create and return an instance of the derived connection class.
3. Create an instance of your client object.
4. When appropriate, create a new connection using `wxClient::MakeConnection` (p. 192), with arguments host name (processed in Unix only, use 'localhost' for local computer), service name, and topic name for this connection. The client object will call `OnMakeConnection` (p. 478) to create a connection object of the derived class if the connection is successful.
5. Use the `wxConnection` member functions to send messages to the server.

## Data transfer

These are the ways that data can be transferred from one application to another. These are methods of `wxConnection`.

- **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.

## 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.

## More DDE details

A `wxClient` object initiates the client part of a client-server DDE-like (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 receive communications in a 'conversation' with a server. and the custom `wxServer` is required so that a user-overridden `wxClient::OnMakeConnection` (p. 478) member can return a `wxConnection` of the required class, when a connection is made.

For example:

```
class MyConnection: public wxConnection {
public:
    MyConnection(void)::wxConnection() {}
    ~MyConnection(void) {}
    bool OnAdvise(const wxString& topic, const wxString& item, char
    *data, int size, wxIPCFormat format)
    { wxMessageBox(topic, data); }
};

class MyClient: public wxClient {
public:
    MyClient(void) {}
    wxConnectionBase *OnMakeConnection(void) { return new
    MyConnection; }
};
```

Here, **MyConnection** will respond to `OnAdvise` (p. 480) messages sent by the server by displaying a message box.

When the client application starts, it must create 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` (p. 478) 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 (an Advise loop is where the server calls the client when data has changed).

```
wxString server = "4242";
wxString hostName;
wxGetHostName(hostName);

// Create a new client
MyClient *client = new MyClient;
connection = (MyConnection *)client->MakeConnection(hostName,
server, "IPC TEST");

if (!connection)
{
    wxMessageBox("Failed to make connection to server", "Client Demo
Error");
    return NULL;
}
connection->StartAdvise("Item");
```

## wxHTML overview

This topic was written by Vaclav Slavik, the author of the wxHTML library.

The wxHTML library provides classes for parsing and displaying HTML.

It is not intended to be a high-end HTML browser. If you are looking for something like that try <http://www.mozilla.org> (<http://www.mozilla.org>).

wxHTML can be used as a generic rich text viewer - for example to display a nice About Box (like those of GNOME apps) or to display the result of database searching. There is a *wxFileSystem* (p. 633) class which allows you to use your own virtual file systems.

wxHtmlWindow supports tag handlers. This means that you can easily extend wxHtml library with new, unsupported tags. Not only that, you can even use your own application-specific tags! See `src/html/m_*.cpp` files for details.

There is a generic wxHtmlParser class, independent of wxHtmlWindow.

## wxHTML quick start

### Displaying HTML

First of all, you must include `<wx/wxhtml.h>`.

Class *wxHtmlWindow* (p. 872) (derived from *wxScrolledWindow*) is used to display HTML documents. It has two important methods: *LoadPage* (p. 875) and *SetPage* (p. 880). *LoadPage* loads and displays HTML file while *SetPage* displays directly the passed **string**. See the example:

```
mywin -> LoadPage("test.htm");
```

```
mywin -> SetPage("<html><body>"
                 "<h1>Error</h1>"
                 "Some error occurred :-H)"
                 "</body></html>");
```

I think the difference is quite clear.

### Displaying Help

See *wxHtmlHelpController* (p. 838).

### Setting up wxHtmlWindow

Because *wxHtmlWindow* is derived from *wxScrolledWindow* and not from *wxFrame*, it doesn't have visible frame. But the user usually wants to see the title of HTML page displayed somewhere and the frame's titlebar is the ideal place for it.

*wxHtmlWindow* provides 2 methods in order to handle this: *SetRelatedFrame* (p. 880) and *SetRelatedStatusBar* (p. 880). See the example:

```
html = new wxHtmlWindow(this);
html -> SetRelatedFrame(this, "HTML : %s");
html -> SetRelatedStatusBar(0);
```

The first command associates the HTML object with its parent frame (this points to *wxFrame* object there) and sets the format of the title. Page title "Hello, world!" will be displayed as "HTML : Hello, world!" in this example.

The second command sets which frame's status bar should be used to display browser's messages (such as "Loading..." or "Done" or hypertext links).

### Customizing wxHtmlWindow

You can customize *wxHtmlWindow* by setting font size, font face and borders (space between border of window and displayed HTML). Related functions:

- *SetFont*s (p. 879)
- *SetBorder*s (p. 879)
- *ReadCustomization* (p. 877)
- *WriteCustomization* (p. 881)

The last two functions are used to store user customization info *wxConfig* stuff (for example in the registry under Windows, or in a dotfile under Unix).

### HTML Printing

The *wxHTML* library provides printing facilities with several levels of complexity.

The easiest way to print an HTML document is to use *wxHtmlEasyPrinting* class (p. 833). It lets you print HTML documents with only one command and you don't have to worry about deriving from the *wxPrintout* class at all. It is only a simple wrapper around the

*wxHtmlPrintout* (p. 863), normal *wxWidgets* printout class.

And finally there is the low level class *wxHtmlDCRenderer* (p. 831) which you can use to render HTML into a rectangular area on any DC. It supports rendering into multiple rectangles with the same width. (The most common use of this is placing one rectangle on each page or printing into two columns.)

## Help Files Format

wxHTML library uses a reduced version of MS HTML Workshop format. Tex2RTF can produce these files when generating HTML, if you set **htmlWorkshopFiles** to **true** in your *tex2rtf.ini* file.

(See *wxHtmlHelpController* (p. 838) for help controller description.)

A **book** consists of three files: header file, contents file and index file. You can make a regular zip archive of these files, plus the HTML and any image files, for wxHTML (or *helpview*) to read; and the .zip file can optionally be renamed to .htb.

### Header file (.hhp)

Header file must contain these lines (and may contain additional lines which are ignored) :

```
Contents file=<filename.hhc>
Index file=<filename.hhk>
Title=<title of your book>
Default topic=<default page to be displayed.htm>
```

All filenames (including the Default topic) are relative to the location of .hhp file.

**Localization note:** In addition, .hhp file may contain line

```
Charset=<rfc_charset>
```

which specifies what charset (e.g. "iso8859\_1") was used in contents and index files. Please note that this line is incompatible with MS HTML Help Workshop and it would either silently remove it or complain with some error. See also *Writing non-English applications* (p. 2063).

### Contents file (.hhc)

Contents file has HTML syntax and it can be parsed by regular HTML parser. It contains exactly one list (<ul>....</ul> statement):

```
<ul>

  <li> <object type="text/sitemap">
    <param name="Name" value="@topic name@">
    <param name="ID" value="@numeric_id@">
    <param name="Local" value="@filename.htm@">
  </object>
  <li> <object type="text/sitemap">
    <param name="Name" value="@topic name@">
    <param name="ID" value="@numeric_id@">
```

```
        <param name="Local" value="@filename.htm@">
    </object>
    ...
</ul>
```

You can modify value attributes of param tags. *topic name* is name of chapter/topic as is displayed in contents, *filename.htm* is HTML page name (relative to .hhp file) and *numeric\_id* is optional - it is used only when you use *wxHtmlHelpController::Display(int)* (p. 841)

Items in the list may be nested - one `<li>` statement may contain a `<ul>` sub-statement:

```
<ul>
  <li> <object type="text/sitemap">
    <param name="Name" value="Top node">
    <param name="Local" value="top.htm">
  </object>
  <ul>
    <li> <object type="text/sitemap">
      <param name="Name" value="subnode in topnode">
      <param name="Local" value="subnode1.htm">
    </object>
    ...
  </ul>
  <li> <object type="text/sitemap">
    <param name="Name" value="Another Top">
    <param name="Local" value="top2.htm">
  </object>
  ...
</ul>
```

### Index file (.hhk)

Index files have same format as contents file except that ID params are ignored and sublists are **not** allowed.

### Input Filters

The wxHTML library provides a mechanism for reading and displaying files of many different file formats.

*wxHtmlWindow::LoadPage* (p. 875) can load not only HTML files but any known file. To make a file type known to *wxHtmlWindow* you must create a *wxHtmlFilter* (p. 837) filter and register it using *wxHtmlWindow::AddFilter* (p. 873).

### Cells and Containers

This article describes mechanism used by *wxHtmlWinParser* (p. 883) and *wxHtmlWindow* (p. 872) to parse and display HTML documents.

## Cells

You can divide any text (or HTML) into small fragments. Let's call these fragments **cells**. Cell is for example one word, horizontal line, image or any other part of document. Each cell has width and height (except special "magic" cells with zero dimensions - e.g. colour changers or font changers).

See *wxHtmlCell* (p. 819).

## Containers

Container is kind of cell that may contain sub-cells. Its size depends on number and sizes of its sub-cells (and also depends on width of window).

See *wxHtmlContainerCell* (p. 826), *wxHtmlCell::Layout* (p. 823).

## Using Containers in Tag Handler

*wxHtmlWinParser* (p. 883) provides a user-friendly way of managing containers. It is based on the idea of opening and closing containers.

Use *OpenContainer* (p. 886) to open new a container *within an already opened container*. This new container is a *sub-container* of the old one. (If you want to create a new container with the same depth level you can call `CloseContainer(); OpenContainer();`)

Use *CloseContainer* (p. 884) to close the container. This doesn't create a new container with same depth level but it returns "control" to the parent container.

It is clear there must be same number of calls to `OpenContainer` as to `CloseContainer`...

## Example

This code creates a new paragraph (container at same depth level) with "Hello, world!":

```
m_WParser -> CloseContainer();
c = m_WParser -> OpenContainer();

m_WParser -> AddWord("Hello, ");
m_WParser -> AddWord("world!");

m_WParser -> CloseContainer();
m_WParser -> OpenContainer();
```

You can see that there was opened container before running the code. We closed it, created our own container, then closed our container and opened new container. The result was that we had *same depth level* after executing. This is general rule that should be followed by tag handlers: leave depth level of containers unmodified (in other words, number of `OpenContainer` and `CloseContainer` calls should be same within *HandleTag* (p. 869)'s body).

## Tag Handlers

The wxHTML library provides architecture of pluggable *tag handlers*. Tag handler is class that understands particular HTML tag (or tags) and is able to interpret it.

*wxHtmlWinParser* (p. 883) has static table of **modules**. Each module contains one or more tag handlers. Each time a new *wxHtmlWinParser* object is constructed all modules are scanned and handlers are added to *wxHtmlParser*'s list of available handlers (note: *wxHtmlParser*'s list is non-static).

### How it works

Common tag handler's *HandleTag* (p. 869) method works in four steps:

1. Save state of parent parser into local variables
2. Change parser state according to tag's params
3. Parse text between the tag and paired ending tag (if present)
4. Restore original parser state

See *wxHtmlWinParser* (p. 883) for methods for modifying parser's state. In general you can do things like opening/closing containers, changing colors, fonts etc.

### Providing own tag handlers

You should create new .cpp file and place following lines into it:

```
#include <mod_tmpl.h>
#include <forcelink.h>
FORCE_LINK_ME(yourmodulefilenamewithoutcpp)
```

Then you must define handlers and one module.

### Tag handlers

The handler is derived from *wxHtmlWinTagHandler* (p. 888)(or directly from *wxHtmlTagHandler* (p. 869))

You can use set of macros to define the handler (see *src/html/m\_\*.cpp* files for details). Handler definition must start with **TAG\_HANDLER\_BEGIN** macro and end with **TAG\_HANDLER\_END** macro. I strongly recommend to have a look at *include/wxhtml/mod\_tmpl.h* file. Otherwise you won't understand the structure of macros. See macros reference:

#### **TAG\_HANDLER\_BEGIN**(*name*, *tags*)

Starts handler definition. *name* is handler identifier (in fact part of class name), *tags* is string containing list of tags supported by this handler (in uppercase). This macro derives new class from *wxHtmlWinTagHandler* and implements it is *GetSupportedTags* (p. 869) method.

Example: **TAG\_HANDLER\_BEGIN**(FONTS, "B,I,U,T")

#### **TAG\_HANDLER\_VARS**

This macro starts block of variables definitions. (Variables are identical to class attributes.) Example:



```
    TAG_HANDLER_BEGIN(VARS_ONLY, "CRAZYTAG")
        TAG_HANDLER_VARS
            int my_int_var;
            wxString something_else;
    TAG_HANDLER_END(VARS_ONLY)
```

This macro is used only in rare cases.

### **TAG\_HANDLER\_CONSTR(*name*)**

This macro supplies object constructor. *name* is same name as the one from TAG\_HANDLER\_BEGIN macro. Body of constructor follow after this macro (you must use and ). Example:

```
    TAG_HANDLER_BEGIN(VARS2, "CRAZYTAG")
        TAG_HANDLER_VARS
            int my_int_var;
        TAG_HANDLER_CONSTR(vars2)
            { // !!!!!
                my_int_var = 666;
            } // !!!!!
    TAG_HANDLER_END(VARS2)
```

Never used in wxHTML :-)

### **TAG\_HANDLER\_PROC(*varib*)**

This is very important macro. It defines *HandleTag* (p. 869) method. *varib* is name of parameter passed to the method, usually *tag*. Body of method follows after this macro. Note than you must use and ! Example:

```
    TAG_HANDLER_BEGIN(TITLE, "TITLE")
        TAG_HANDLER_PROC(tag)
            {
                printf("TITLE found...\n");
            }
    TAG_HANDLER_END(TITLE)
```

### **TAG\_HANDLER\_END(*name*)**

Ends definition of tag handler *name*.

## **Tags Modules**

You can use set of 3 macros TAGS\_MODULE\_BEGIN, TAGS\_MODULE\_ADD and TAGS\_MODULE\_END to inherit new module from *wxHtmlTagsModule* (p. 870) and to create instance of it. See macros reference:

### **TAGS\_MODULE\_BEGIN(*modname*)**

Begins module definition. *modname* is part of class name and must be unique.

### **TAGS\_MODULE\_ADD(*name*)**

Adds the handler to this module. *name* is the identifier from TAG\_HANDLER\_BEGIN.

**TAGS\_MODULE\_END(modname)**

Ends the definition of module.

**Example:**

```
TAGS_MODULE_BEGIN(Examples)
    TAGS_MODULE_ADD(VARS_ONLY)
    TAGS_MODULE_ADD(VARS2)
    TAGS_MODULE_ADD(TITLE)
TAGS_MODULE_END(Examples)
```

**Tags supported by wxHTML**

wxHTML is not full implementation of HTML standard. Instead, it supports most common tags so that it is possible to display *simple* HTML documents with it. (For example it works fine with pages created in Netscape Composer or generated by tex2rtf).

Following tables list all tags known to wxHTML, together with supported parameters. A tag has general form of <tagname param\_1 param\_2 ... param\_n> where param\_i is either paramname="paramvalue" or paramname=paramvalue - these two are equivalent. Unless stated otherwise, wxHTML is case-insensitive.

**Table of common parameter values**

We will use these substitutions in tags descriptions:

[alignment]	CENTER LEFT RIGHT JUSTIFY
[v_alignment]	TOP BOTTOM CENTER
[color]	HTML 4.0-compliant colour specification
[fontsize]	-2 -1 +0 +1 +2 +3 +4 1 2 3 4 5 6 7
[pixels]	integer value that represents dimension in pixels

[percent]	i% where i is integer
[url]	an URL
[string]	text string
[coords]	c(1),c(2),c(3),...,c(n) where c(i) is integer

### List of supported tags

A	NAME=[string] HREF=[url] TARGET=[target window spec]
ADDRESS	
AREA	SHAPE=POLY SHAPE=CIRCLE SHAPE=RECT COORDS=[coords] HREF=[url]
B	
BIG	
BLOCKQUOTE	
BODY	TEXT=[color] LINK=[color] BGCOLOR=[color] ALIGN=[alignment]
BR	
CENTER	
CITE	
CODE	
DD	
DIV	ALIGN=[alignment]
DL	
DT	
EM	
FONT	COLOR=[color] SIZE=[fontsize] FACE=[comma-separated list of facenames]
HR	ALIGN=[alignment] SIZE=[pixels] WIDTH=[percent pixels] NOSHAE
H1	
H2	
H3	
H4	
H5	
H6	
I	
IMG	SRC=[url] WIDTH=[pixels] HEIGHT=[pixels] ALIGN=TEXTTOP ALIGN=CENTER ALIGN=ABSCENTER

	ALIGN=BOTTOM USEMAP=[url]
KBD	
LI	
MAP	NAME=[string]
META	HTTP-EQUIV="Content-Type" CONTENT=[string]
OL	
P	ALIGN=[alignment]
PRE	
SAMP	
SMALL	
STRIKE	
STRONG	
SUB	
SUP	
TABLE	ALIGN=[alignment] WIDTH=[percent pixels] BORDER=[pixels] VALIGN=[v_alignment] BGCOLOR=[color] CELLSPACING=[pixels] CELLPADDING=[pixels]
TD	ALIGN=[alignment] VALIGN=[v_alignment] BGCOLOR=[color] WIDTH=[percent pixels] COLSPAN=[pixels] ROWSPAN=[pixels] NOWRAP
TH	ALIGN=[alignment] VALIGN=[v_alignment] BGCOLOR=[color] WIDTH=[percent pixels] COLSPAN=[pixels] ROWSPAN=[pixels]
TITLE	
TR	ALIGN=[alignment] VALIGN=[v_alignment] BGCOLOR=[color]
TT	
U	
UL	

## wxRichTextCtrl overview

**Major classes:** *wxRichTextCtrl* (p. 1316), *wxRichTextBuffer* (p. 1298), *wxRichTextEvent* (p. 1346)

**Helper classes:** *wxRichTextAttr* (p. 1281), *wxTextAttrEx* (p. 1623), *wxRichTextRange* (p. 1373)

**File handler classes:** *wxRichTextFileHandler* (p. 1350), *wxRichTextHTMLHandler* (p.

1362), *wxRichTextXMLHandler* (p. 1390)

**Style classes:** *wxRichTextCharacterStyleDefinition* (p. 1315), *wxRichTextParagraphStyleDefinition* (p. 1367), *wxRichTextListStyleDefinition* (p. 1365), *wxRichTextStyleSheet* (p. 1387)

**Additional controls:** *wxRichTextStyleComboCtrl* (p. 1378), *wxRichTextStyleListBox* (p. 1379), *wxRichTextStyleListCtrl* (p. 1381)

**Printing classes:** *wxRichTextPrinting* (p. 1368), *wxRichTextPrintout* (p. 1371), *wxRichTextHeaderFooterData* (p. 1359)

**Dialog classes:** *wxRichTextStyleOrganiserDialog* (p. 1383), *wxRichTextFormattingDialog* (p. 1353), *wxSymbolPickerDialog* (p. 1586)

*wxRichTextCtrl* provides a generic implementation of a rich text editor that can handle different character styles, paragraph formatting, and images. It's aimed at editing 'natural' language text - if you need an editor that supports code editing, *wxStyledTextCtrl* is a better choice.

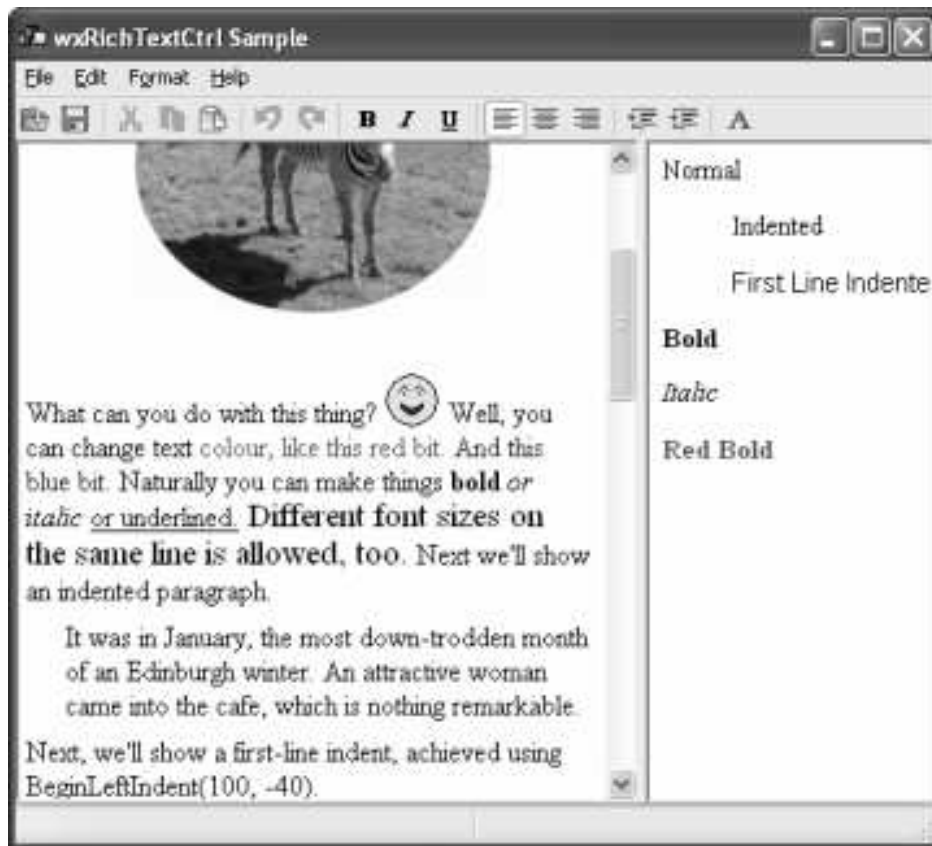
Despite its name, it cannot currently read or write RTF (rich text format) files. Instead, it uses its own XML format, and can also read and write plain text. In future we expect to provide RTF file capabilities. Custom file formats can be supported by creating additional file handlers and registering them with the control.

*wxRichTextCtrl* is largely compatible with the *wxTextCtrl* API, but extends it where necessary. The control can be used where the native rich text capabilities of *wxTextCtrl* are not adequate (this is particularly true on Windows) and where more direct access to the content representation is required. It is difficult and inefficient to read the style information in a *wxTextCtrl*, whereas this information is readily available in *wxRichTextCtrl*. Since it's written in pure *wxWidgets*, any customizations you make to *wxRichTextCtrl* will be reflected on all platforms.

*wxRichTextCtrl* supports basic printing via the easy-to-use *wxRichTextPrinting* (p. 1368) class. Creating applications with simple word processing features is simplified with the inclusion of *wxRichTextFormattingDialog* (p. 1353), a tabbed dialog allowing interactive tailoring of paragraph and character styling. Also provided is the multi-purpose dialog *wxRichTextStyleOrganiserDialog* (p. 1383) that can be used for managing style definitions, browsing styles and applying them, or selecting list styles with a renumber option.

There are a few disadvantages to using *wxRichTextCtrl*. It is not native, so does not behave exactly as a native *wxTextCtrl*, although common editing conventions are followed. Users may miss the built-in spelling correction on Mac OS X, or any special character input that may be provided by the native control. It would also be a poor choice if intended users rely on screen readers that would be not work well with non-native text input implementation. You might mitigate this by providing the choice between *wxTextCtrl* and *wxRichTextCtrl*, with fewer features in the former case.

A good way to understand *wxRichTextCtrl*'s capabilities is to compile and run the sample, *samples/richtext*, and browse the code. The following screenshot shows the sample in action:



### Example

The following code is taken from the sample, and adds text and styles to a rich text control programmatically.

```
wxRichTextCtrl* richTextCtrl = new wxRichTextCtrl(splitter,
wxID_ANY, wxEmptyString, wxDefaultPosition, wxSize(200, 200),
wxVSCROLL|wxHSCROLL|wxNO_BORDER|wxWANTS_CHARS);

wxFont textFont = wxFont(12, wxROMAN, wxNORMAL, wxNORMAL);
wxFont boldFont = wxFont(12, wxROMAN, wxNORMAL, wxBOLD);
wxFont italicFont = wxFont(12, wxROMAN, wxITALIC, wxNORMAL);

wxFont font(12, wxROMAN, wxNORMAL, wxNORMAL);

m_richTextCtrl->SetFont(font);

wxRichTextCtrl& r = richTextCtrl;

r.BeginSuppressUndo();

r.BeginParagraphSpacing(0, 20);

r.BeginAlignment(wxTEXT_ALIGNMENT_CENTRE);
r.BeginBold();

r.BeginFontSize(14);
```

```
        r.WriteText(wxT("Welcome to wxRichTextCtrl, a wxWidgets control
for editing and presenting styled text and images"));
        r.EndFontSize();
        r.Newline();

        r.BeginItalic();
        r.WriteText(wxT("by Julian Smart"));
        r.EndItalic();

        r.EndBold();

        r.Newline();
        r.WriteImage(wxBitmap(zebra_xpm));

        r.EndAlignment();

        r.Newline();
        r.Newline();

        r.WriteText(wxT("What can you do with this thing? "));
        r.WriteImage(wxBitmap(smiley_xpm));
        r.WriteText(wxT(" Well, you can change text "));

        r.BeginTextColour(wxColour(255, 0, 0));
        r.WriteText(wxT("colour, like this red bit."));
        r.EndTextColour();

        r.BeginTextColour(wxColour(0, 0, 255));
        r.WriteText(wxT(" And this blue bit."));
        r.EndTextColour();

        r.WriteText(wxT(" Naturally you can make things "));
        r.BeginBold();
        r.WriteText(wxT("bold "));
        r.EndBold();
        r.BeginItalic();
        r.WriteText(wxT("or italic "));
        r.EndItalic();
        r.BeginUnderline();
        r.WriteText(wxT("or underlined."));
        r.EndUnderline();

        r.BeginFontSize(14);
        r.WriteText(wxT(" Different font sizes on the same line is allowed,
too. "));
        r.EndFontSize();

        r.WriteText(wxT(" Next we'll show an indented paragraph. "));

        r.BeginLeftIndent(60);
        r.Newline();

        r.WriteText(wxT("Indented paragraph. "));
        r.EndLeftIndent();

        r.Newline();
```

```
    r.WriteText(wxT("Next, we'll show a first-line indent, achieved
using BeginLeftIndent(100, -40)."));

    r.BeginLeftIndent(100, -40);
    r.Newline();

    r.WriteText(wxT("It was in January, the most down-trodden month
of an Edinburgh winter."));
    r.EndLeftIndent();

    r.Newline();

    r.WriteText(wxT("Numbered bullets are possible, again using
subindents:"));

    r.BeginNumberedBullet(1, 100, 60);
    r.Newline();

    r.WriteText(wxT("This is my first item. Note that wxRichTextCtrl
doesn't automatically do numbering, but this will be added later.));
    r.EndNumberedBullet();

    r.BeginNumberedBullet(2, 100, 60);
    r.Newline();

    r.WriteText(wxT("This is my second item.));
    r.EndNumberedBullet();

    r.Newline();

    r.WriteText(wxT("The following paragraph is right-indented:"));

    r.BeginRightIndent(200);
    r.Newline();

    r.WriteText(wxT("It was in January, the most down-trodden month
of an Edinburgh winter. An attractive woman came into the cafe, which
is nothing remarkable.));
    r.EndRightIndent();

    r.Newline();

    wxArrayInt tabs;
    tabs.Add(400);
    tabs.Add(600);
    tabs.Add(800);
    tabs.Add(1000);
    wxTextAttrEx attr;
    attr.SetFlags(wxTEXT_ATTR_TABS);
    attr.SetTabs(tabs);
    r.SetDefaultStyle(attr);

    r.WriteText(wxT("This line contains tabs:\tFirst tab\tSecond
tab\tThird tab"));

    r.Newline();
    r.WriteText(wxT("Other notable features of wxRichTextCtrl
```



```
include:"));

    r.BeginSymbolBullet(wxT('*'), 100, 60);
    r.Newline();
    r.WriteText(wxT("Compatibility with wxTextCtrl API"));
    r.EndSymbolBullet();

    r.WriteText(wxT("Note: this sample content was generated
programmatically from within the MyFrame constructor in the demo. The
images were loaded from inline XPMs. Enjoy wxRichTextCtrl!"));

    r.EndSuppressUndo();
```

## Programming with wxRichTextCtrl

### Starting to use wxRichTextCtrl

You need to include `<wx/richtext/richtextctrl.h>` in your source, and link with the appropriate wxWidgets library with `richtext` suffix. Put the rich text library first in your link line to avoid unresolved symbols.

Then you can create a `wxRichTextCtrl`, with the `wxWANT_CHARS` style if you want tabs to be processed by the control rather than being used for navigation between controls.

### wxRichTextCtrl and styles

Styling attributes are represented by three classes: `wxTextAttr` (p. 1618), `wxTextAttrEx` (p. 1623) and `wxRichTextAttr` (p. 1281). `wxTextAttr` is shared across all controls that are derived from `wxTextCtrlBase` and can store basic character and paragraph attributes. `wxTextAttrEx` derives from `wxTextAttr` and adds some further attributes that are only supported by `wxRichTextCtrl`. Finally, `wxRichTextAttr` is a more efficient version of `wxTextAttrEx` that doesn't use a `wxFont` object and can be used to query styles more quickly. `wxTextAttrEx` and `wxRichTextAttr` are largely interchangeable and have suitable conversion operators between them.

When setting a style, the flags of the attribute object determine which attributes are applied. When querying a style, the passed flags are ignored except (optionally) to determine whether attributes should be retrieved from character content or from the paragraph object.

`wxRichTextCtrl` takes a layered approach to styles, so that different parts of the content may be responsible for contributing different attributes to the final style you see on the screen.

There are four main notions of style within a control:

1. **Basic style:** the fundamental style of a control, onto which any other styles are layered. It provides default attributes, and changing the basic style may immediately change the look of the content depending on what other styles the content uses. Calling `wxRichTextCtrl::SetFont` changes the font for the basic style.

The basic style is set with `wxRichTextCtrl::SetBasicStyle` (p. 1341).

2. **Paragraph style:** each paragraph has attributes that are set independently from other paragraphs and independently from the content within the paragraph. Normally, these attributes are paragraph-related, such as alignment and indentation, but it is possible to set character attributes too. The paragraph style can be set independently of its content by passing `wxRICHTEXT_SETSTYLE_PARAGRAPHS_ONLY` to `wxRichTextCtrl::SetStyleEx` (p. 1344).
3. **Character style:** characters within each paragraph can have attributes. A single character, or a run of characters, can have a particular set of attributes. The character style can be with `wxRichTextCtrl::SetStyle` (p. 1343) or `wxRichTextCtrl::SetStyleEx` (p. 1344).
4. **Default style:** this is the 'current' style that determines the style of content that is subsequently typed, pasted or programmatically inserted. The default style is set with `wxRichTextCtrl::SetDefaultStyle` (p. 1341).

What you see on the screen is the dynamically *combined* style, found by merging the first three of the above style types (the fourth is only a guide for future content insertion and therefore does not affect the currently displayed content).

To make all this more concrete, here are examples of where you might set these different styles:

1. You might set the **basic style** to have a Times Roman font in 12 point, left-aligned, with two millimetres of spacing after each paragraph.
2. You might set the **paragraph style** (for one particular paragraph) to be centred.
3. You might set the **character style** of one particular word to bold.
4. You might set the **default style** to be underlined, for subsequent inserted text.

Naturally you can do any of these things either using your own UI, or programmatically.

The basic `wxTextCtrl` doesn't make the same distinctions as `wxRichTextCtrl` regarding attribute storage. So we need finer control when setting and retrieving attributes.

`wxRichTextCtrl::SetStyleEx` (p. 1344) takes a *flags* parameter:

- `wxRICHTEXT_SETSTYLE_OPTIMIZE` specifies that the style should be changed only if the combined attributes are different from the attributes for the current object. This is important when applying styling that has been edited by the user, because he has just edited the *combined* (visible) style, and `wxRichTextCtrl` wants to leave unchanged attributes associated with their original objects instead of applying them to both paragraph and content objects.
- `wxRICHTEXT_SETSTYLE_PARAGRAPHS_ONLY` specifies that only paragraph objects within the given range should take on the attributes.
- `wxRICHTEXT_SETSTYLE_CHARACTERS_ONLY` specifies that only content objects (text or images) within the given range should take on the attributes.

- `wxRICHTEXT_SETSTYLE_WITH_UNDO` specifies that the operation should be undoable.

It's great to be able to change arbitrary attributes in a `wxRichTextCtrl`, but it can be unwieldy for the user or programmer to set attributes separately. Word processors have collections of styles that you can tailor or use as-is, and this means that you can set a heading with one click instead of marking text in bold, specifying a large font size, and applying a certain paragraph spacing and alignment for every such heading. Similarly, `wxWidgets` provides a class called `wxRichTextStyleSheet` (p. 1387) which manages style definitions (`wxRichTextParagraphStyleDefinition` (p. 1367), `wxRichTextListStyleDefinition` (p. 1365) and `wxRichTextCharacterStyleDefinition` (p. 1315)). Once you have added definitions to a style sheet and associated it with a `wxRichTextCtrl`, you can apply a named definition to a range of text. The classes `wxRichTextStyleComboCtrl` (p. 1378) and `wxRichTextStyleListBox` (p. 1379) can be used to present the user with a list of styles in a sheet, and apply them to the selected text.

You can reapply a style sheet to the contents of the control, by calling `wxRichTextCtrl::ApplyStyleSheet` (p. 1317). This is useful if the style definitions have changed, and you want the content to reflect this. It relies on the fact that when you apply a named style, the style definition name is recorded in the content. So `ApplyStyleSheet` works by finding the paragraph attributes with style names and re-applying the definition's attributes to the paragraph. Currently, this works with paragraph and list style definitions only.

## **wxRichTextCtrl dialogs**

`wxRichTextCtrl` comes with standard dialogs to make it easier to implement text editing functionality.

`wxRichTextFormattingDialog` (p. 1353) can be used for character or paragraph formatting, or a combination of both. It's a `wxPropertySheetDialog` with the following available tabs: Font, Indents & Spacing, Tabs, Bullets, Style, and List Style. You can select which pages will be shown by supplying flags to the dialog constructor. In a character formatting dialog, typically only the Font page will be shown. In a paragraph formatting dialog, you'll show the Indents & Spacing, Tabs and Bullets pages. The Style tab is useful when editing a style definition.

You can customize this dialog by providing your own `wxRichTextFormattingDialogFactory` object, which tells the formatting dialog how many pages are supported, what their identifiers are, and how to create the pages.

`wxRichTextStyleOrganiserDialog` (p. 1383) is a multi-purpose dialog that can be used for managing style definitions, browsing styles and applying them, or selecting list styles with a renumber option. See the sample for usage - it is used for the "Manage Styles" and "Bullets and Numbering" menu commands.

`wxSymbolPickerDialog` (p. 1586) lets the user insert a symbol from a specified font. It has no `wxRichTextCtrl` dependencies besides being included in the rich text library.

## **How wxRichTextCtrl is implemented**

Data representation is handled by `wxRichTextBuffer`, and a `wxRichTextCtrl` always has one such buffer.

The content is represented by a hierarchy of objects, all derived from `wxRichTextObject`. An object might be an image, a fragment of text, a paragraph, or a whole buffer. Objects store a `wxTextAttrEx` containing style information; a paragraph object can contain both paragraph and character information, but content objects such as text can only store character information. The final style displayed in the control or in a printout is a combination of base style, paragraph style and content (character) style.

The top of the hierarchy is the buffer, a kind of `wxRichTextParagraphLayoutBox`, containing further `wxRichTextParagraph` objects, each of which can include text, images and potentially other types of object.

Each object maintains a range (start and end position) measured from the start of the main parent object.

When `Layout` is called on an object, it is given a size which the object must limit itself to, or one or more flexible directions (vertical or horizontal). So, for example, a centred paragraph is given the page width to play with (minus any margins), but can extend indefinitely in the vertical direction. The implementation of `Layout` caches the calculated size and position.

When the buffer is modified, a range is invalidated (marked as requiring layout), so that only the minimum amount of layout is performed.

A paragraph of pure text with the same style contains just one further object, a `wxRichTextPlainText` object. When styling is applied to part of this object, the object is decomposed into separate objects, one object for each different character style. So each object within a paragraph always has just one `wxTextAttrEx` object to denote its character style. Of course, this can lead to fragmentation after a lot of edit operations, potentially leading to several objects with the same style where just one would do. So a `Defragment` function is called when updating the control's display, to ensure that the minimum number of objects is used.

## **wxRichTextCtrl roadmap**

### **Bugs**

This is an incomplete list of bugs.

- Moving the caret up at the beginning of a line sometimes incorrectly positions the caret.
- As the selection is expanded, the text jumps slightly due to kerning differences between drawing a single text string versus drawing several fragments separately. This could be improved by using `wxDC::GetPartialTextExtents` to calculate exactly where the separate fragments should be drawn. Note that this problem also applies to separation of text fragments due to difference in their attributes.

### **Features**

This is a list of some of the features that have yet to be implemented. Help with them will be appreciated.

- RTF input and output
- Conversion from HTML
- Open Office input and output
- Floating images, with content wrapping around them
- A ruler control
- Standard editing toolbars
- Tables
- Bitmap bullets
- Borders
- Text frames
- Justified text, in print/preview at least

There are also things that could be done to take advantage of the underlying text capabilities of the platform; higher-level text formatting APIs are available on some platforms, such as Mac OS X, and some of translation from high level to low level wxDC API is unnecessary. However this would require additions to the wxWidgets API.

## wxAUI overview

Class: *wxAuiManager* (p. 99), *wxAuiPanelInfo* (p. 110)

wxAUI stands for Advanced User Interface and the wxAUI framework aims to give its user a cutting edge interface for use with the wxWidgets based applications. The original wxAUI sources have kindly been made available under the wxWindows licence by Kirix Corp. and they have since then been integrated into wxWidgets CVS and further improved.

wxAUI attempts to encapsulate the following aspects of the user interface:

**Frame Management:**Frame management provides the means to open, move and hide common controls that are needed to interact with the document, and allow these configurations to be saved into different perspectives and loaded at a later time.

**Toolbars:**Toolbars are a specialized subset of the frame management system and should behave similarly to other docked components. However, they also require additional functionality, such as "spring-loaded" rebar support, "chevron" buttons and end-user customizability.

**Modeless Controls:**Modeless controls expose a tool palette or set of options that float

above the application content while allowing it to be accessed. Usually accessed by the toolbar, these controls disappear when an option is selected, but may also be "torn off" the toolbar into a floating frame of their own.

**Look and Feel:** Look and feel encompasses the way controls are drawn, both when shown statically as well as when they are being moved. This aspect of user interface design incorporates "special effects" such as transparent window dragging as well as frame animation.

wxAUI adheres to the following principles:

Use native floating frames to obtain a native look and feel for all platforms. Use existing wxWidgets code where possible, such as sizer implementation for frame management. Use classes included in wxCore and wxBase only. Use standard wxWidgets coding conventions.

## Environment variables

This section describes all environment variables that affect execution of wxWidgets programs.

WXTRACE	(Debug build only.) This variable can be set to a comma-separated list of trace masks used in <i>wxLogTrace</i> (p. 1974) calls; <i>wxLog::AddTraceMask</i> (p. 1021) is called for every mask in the list during wxWidgets initialization.
WXPREFIX	(Unix only.) Overrides installation prefix. Normally, the prefix is hard-coded and is the same as the value passed to <i>configure</i> via the <code>--prefix</code> switch when compiling the library (typically <code>/usr/local</code> or <code>/usr</code> ). You can set WXPREFIX if you are for example distributing a binary version of an application and you don't know in advance where it will be installed.
WXMODE	(wxMGL only.) Sets MGL video mode. The value must be in form <i>widthxheight-depth</i> . The default is <code>640x480-16</code> .
WXSTDERR	(wxMGL only.) Redirects stderr output to a file.

## wxPython overview

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

### What is wxPython?

wxPython is a blending of the wxWidgets GUI classes and the Python (<http://www.python.org/>) programming language.

### Python

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

Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. There are interfaces to many system calls and libraries, and new built-in modules are easily written in C or C++. Python is also usable as an extension language for applications that need a programmable interface.

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

### **wxPython**

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

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

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

### **Why use wxPython?**

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

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

### **Other Python GUIs**

There are other GUI solutions out there for Python.

#### **Tkinter**

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

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

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

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

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

### **PythonWin**

PythonWin is an add-on package for Python for the Win32 platform. It includes wrappers for MFC as well as much of the Win32 API. Because of its foundation, it is very familiar for programmers who have experience with MFC and the Win32 API. It is obviously not compatible with other platforms and toolkits. PythonWin is organized as separate packages and modules so you can use the pieces you need without having to use the GUI portions.

### **Others**

There are quite a few other GUI modules available for Python, some in active use, some that haven't been updated for ages. Most are simple wrappers around some C or C++ toolkit or another, and most are not cross-platform compatible. See this link (<http://www.python.org/download/Contributed.html#Graphics>) for a listing of a few of them.

## **Using wxPython**

### **First things first...**

I'm not going to try and teach the Python language here. You can do that at the Python Tutorial (<http://www.python.org/doc/tut/tut.html>). I'm also going to assume that you know a bit about wxWidgets already, enough to notice the similarities in the classes used.

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

**python DialogUnits.py**

---

```
001: ## import all of the wxPython GUI package
002: from wxPython.wx import *
003:
```

---



```
004: ## Create a new frame class, derived from the wxPython Frame.
005: class MyFrame(wxFrame):
006:
007:     def __init__(self, parent, id, title):
008:         # First, call the base class' __init__ method to create
the frame
009:         wxFrame.__init__(self, parent, id, title,
010:                           wxPoint(100, 100), wxSize(160, 100))
011:
012:         # Associate some events with methods of this class
013:         EVT_SIZE(self, self.OnSize)
014:         EVT_MOVE(self, self.OnMove)
015:
016:         # Add a panel and some controls to display the size and
position
017:         panel = wxPanel(self, -1)
018:         wxStaticText(panel, -1, "Size:",
019:                       wxDLG_PNT(panel, wxPoint(4, 4)), wxDefaultSize)
020:         wxStaticText(panel, -1, "Pos:",
021:                       wxDLG_PNT(panel, wxPoint(4, 14)), wxDefaultSize)
022:         self.sizeCtrl = wxTextCtrl(panel, -1, "",
023:                                     wxDLG_PNT(panel, wxPoint(24,
024:                                     4))),
025:                                     wxDLG_SZE(panel, wxSize(36,
-1))),
026:                                     wxTE_READONLY)
027:         self.posCtrl = wxTextCtrl(panel, -1, "",
028:                                     wxDLG_PNT(panel, wxPoint(24,
14))),
029:                                     wxDLG_SZE(panel, wxSize(36, -1)),
030:                                     wxTE_READONLY)
031:
032:         # This method is called automatically when the CLOSE event
is
033:         # sent to this window
034:         def OnCloseWindow(self, event):
035:             # tell the window to kill itself
036:             self.Destroy()
037:
038:         # This method is called by the system when the window is resized,
039:         # because of the association above.
040:         def OnSize(self, event):
041:             size = event.GetSize()
042:             self.sizeCtrl.SetValue("%s, %s" % (size.width,
size.height))
043:
044:             # tell the event system to continue looking for an event
handler,
045:             # so the default handler will get called.
046:             event.Skip()
047:
048:         # This method is called by the system when the window is moved,
049:         # because of the association above.
050:         def OnMove(self, event):
051:             pos = event.GetPosition()
052:             self.posCtrl.SetValue("%s, %s" % (pos.x, pos.y))
```

```
053:
054:
055: # Every wxWidgets application must have a class derived from wxApp
056: class MyApp(wxApp):
057:
058:     # wxWidgets calls this method to initialize the application
059:     def OnInit(self):
060:
061:         # Create an instance of our customized Frame class
062:         frame = MyFrame(NULL, -1, "This is a test")
063:         frame.Show(true)
064:
065:         # Tell wxWidgets that this is our main window
066:         self.SetTopWindow(frame)
067:
068:         # Return a success flag
069:         return true
070:
071:
072: app = MyApp(0)      # Create an instance of the application class
073: app.MainLoop()      # Tell it to start processing events
074:
```

---

### Things to notice

1. At line 2 the wxPython classes, constants, and etc. are imported into the current module's namespace. If you prefer to reduce namespace pollution you can use "from wxPython import wx" and then access all the wxPython identifiers through the wx module, for example, "wx.wxFrame".
2. At line 13 the frame's sizing and moving events are connected to methods of the class. These helper functions are intended to be like the event table macros that wxWidgets employs. But since static event tables are impossible with wxPython, we use helpers that are named the same to dynamically build the table. The only real difference is that the first argument to the event helpers is always the window that the event table entry should be added to.
3. Notice the use of wxDLG\_PNT and wxDLG\_SIZE in lines 19 - 29 to convert from dialog units to pixels. These helpers are unique to wxPython since Python can't do method overloading like C++.
4. There is an OnCloseWindow method at line 34 but no call to EVT\_CLOSE to attach the event to the method. Does it really get called? The answer is, yes it does. This is because many of the *standard* events are attached to windows that have the associated *standard* method names. I have tried to follow the lead of the C++ classes in this area to determine what is *standard* but since that changes from time to time I can make no guarantees, nor will it be fully documented. When in doubt, use an EVT\_\*\*\* function.
5. At lines 17 to 21 notice that there are no saved references to the panel or the static text items that are created. Those of you who know Python might be wondering what happens when Python deletes these objects when they go out of

scope. Do they disappear from the GUI? They don't. Remember that in wxPython the Python objects are just shadows of the corresponding C++ objects. Once the C++ windows and controls are attached to their parents, the parents manage them and delete them when necessary. For this reason, most wxPython objects do not need to have a `__del__` method that explicitly causes the C++ object to be deleted. If you ever have the need to forcibly delete a window, use the `Destroy()` method as shown on line 36.

6. Just like wxWidgets in C++, wxPython apps need to create a class derived from `wxApp` (line 56) that implements a method named `OnInit`, (line 59.) This method should create the application's main window (line 62) and use `wxApp.SetTopWindow()` (line 66) to inform wxWidgets about it.
7. And finally, at line 72 an instance of the application class is created. At this point wxPython finishes initializing itself, and calls the `OnInit` method to get things started. (The zero parameter here is a flag for functionality that isn't quite implemented yet. Just ignore it for now.) The call to `MainLoop` at line 73 starts the event loop which continues until the application terminates or all the top level windows are closed.

### **wxWidgets classes implemented in wxPython**

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

- `wxAcceleratorEntry` (p. 21)
- `wxAcceleratorTable` (p. 23)
- `wxActivateEvent` (p. 33)
- `wxBitmap` (p. 123)
- `wxBitmapButton` (p. 139)
- `wxBitmapDataObject` (p. 145)
- `wxBMPHandler`
- `wxBoxSizer` (p. 149)
- `wxBrush` (p. 150)
- `wxBusyInfo` (p. 163)
- `wxBusyCursor` (p. 162)
- `wxButton` (p. 164)
- `wxCalculateLayoutEvent` (p. 167)

- *wxCalendarCtrl* (p. 168)
- *wxCaret* (p. 177)
- *wxCheckBox* (p. 180)
- *wxCheckListBox* (p. 183)
- *wxChoice* (p. 186)
- *wxClientDC* (p. 193)
- *wxClipboard* (p. 195)
- *wxCloseEvent* (p. 200)
- *wxColourData* (p. 218)
- *wxColourDialog* (p. 221)
- *wxColour* (p. 214)
- *wxComboBox* (p. 225)
- *wxCommandEvent* (p. 250)
- *wxConfig* (p. 262)
- *wxControl* (p. 285)
- *wxCursor* (p. 297)
- *wxCustomDataObject* (p. 302)
- *wxDataFormat* (p. 304)
- *wxDataObject* (p. 311)
- *wxDataObjectComposite* (p. 334)
- *wxDataObjectSimple* (p. 335)
- *wxDateTime* (p. 348)
- *wxDateSpan* (p. 343)
- *wxDC* (p. 456)
- *wxDialog* (p. 496)
- *wxDirDialog* (p. 513)
- *wxDragImage* (p. 552)

- *wxDropFilesEvent* (p. 557)
- *wxDropSource* (p. 558)
- *wxDropTarget* (p. 561)
- *wxEraseEvent* (p. 571)
- *wxEvent* (p. 572)
- *wxEvtHandler* (p. 576)
- *wxFileConfig* (p. 598)
- *wxFileDataObject* (p. 599)
- *wxFileDialog* (p. 600)
- *wxFileDropTarget* (p. 604)
- *wxFileSystem* (p. 633)
- *wxFileSystemHandler* (p. 636)
- *wxFocusEvent* (p. 655)
- *wxFontData* (p. 668)
- *wxFontDialog* (p. 670)
- *wxFont* (p. 655)
- *wxFrame* (p. 682)
- *wxFSFile* (p. 692)
- *wxGauge* (p. 701)
- *wxGIFHandler*
- *wxGLCanvas* (p. 715)
- *wxHtmlCell* (p. 819)
- *wxHtmlContainerCell* (p. 826)
- *wxHtmlDCRenderer* (p. 831)
- *wxHtmlEasyPrinting* (p. 833)
- *wxHtmlParser* (p. 858)
- *wxHtmlTagHandler* (p. 869)

- *wxHtmlTag* (p. 865)
- *wxHtmlWinParser* (p. 883)
- *wxHtmlPrintout* (p. 863)
- *wxHtmlWinTagHandler* (p. 888)
- *wxHtmlWindow* (p. 872)
- *wxIconizeEvent* (p. 903)
- *wxIcon* (p. 894)
- *wxIdleEvent* (p. 903)
- *wxImage* (p. 906)
- *wxImageHandler* (p. 930)
- *wxImageList* (p. 933)
- *wxIndividualLayoutConstraint* (p. 938)
- *wxInitDialogEvent* (p. 941)
- *wxInputStream* (p. 941)
- *wxInternetFSHandler* (p. 2075)
- *wxJoystickEvent* (p. 954)
- *wxJPEGHandler*
- *wxKeyEvent* (p. 956)
- *wxLayoutAlgorithm* (p. 961)
- *wxLayoutConstraints* (p. 964)
- *wxListBox* (p. 974)
- *wxListCtrl* (p. 980)
- *wxListEvent* (p. 1000)
- *wxListItem* (p. 995)
- *wxMask* (p. 1036)
- *wxMaximizeEvent* (p. 1037)
- *wxMDIChildFrame* (p. 1047)

- *wxMDIClientWindow* (p. 1050)
- *wxMDIParentFrame* (p. 1051)
- *wxMemoryDC* (p. 1069)
- *wxMemoryFSHandler* (p. 1071)
- *wxMenuBar* (p. 1088)
- *wxMenuEvent* (p. 1098)
- *wxMenuItem* (p. 1099)
- *wxMenu* (p. 1074)
- *wxMessageDialog* (p. 1106)
- *wxMetaFileDC* (p. 1109)
- *wxMiniFrame* (p. 1113)
- *wxMouseEvent* (p. 1119)
- *wxMoveEvent* (p. 1128)
- *wxNotebookEvent* (p. 1144)
- *wxNotebook* (p. 1135)
- *wxPageSetupDialogData* (p. 1159)
- *wxPageSetupDialog* (p. 1158)
- *wxPaintDC* (p. 1164)
- *wxPaintEvent* (p. 1164)
- *wxPalette* (p. 1166)
- *wxPanel* (p. 1170)
- *wxPen* (p. 1175)
- *wxPNGHandler*
- *wxPoint* (p. 1193)
- *wxPostScriptDC* (p. 1194)
- *wxPreviewFrame* (p. 1199)
- *wxPrintData* (p. 1200)

- *wxPrintDialogData* (p. 1207)
- *wxPrintDialog* (p. 1206)
- *wxPrinter* (p. 1211)
- *wxPrintPreview* (p. 1221)
- *wxPrinterDC* (p. 1213)
- *wxPrintout* (p. 1214)
- *wxProcess* (p. 1224)
- *wxQueryLayoutInfoEvent* (p. 1238)
- *wxRadioBox* (p. 1241)
- *wxRadioButton* (p. 1249)
- *wxRealPoint* (p. 1251)
- *wxRect* (p. 1252)
- *wxRegionIterator* (p. 1269)
- *wxRegion* (p. 1264)
- *wxSashEvent* (p. 1392)
- *wxSashLayoutWindow* (p. 1394)
- *wxSashWindow* (p. 1397)
- *wxScreenDC* (p. 1407)
- *wxScrollBar* (p. 1408)
- *wxScrollEvent* (p. 1423)
- *wxScrolledWindow* (p. 1414)
- *wxScrollWinEvent* (p. 1426)
- *wxShowEvent*
- *wxSingleChoiceDialog* (p. 1435)
- *wxSizeEvent* (p. 1443)
- *wxSize* (p. 1440)
- *wxSizer* (p. 1444)



- *wxSizerItem* (p. 1458)
- *wxSlider* (p. 1462)
- *wxSpinButton* (p. 1496)
- *wxSpinEvent* (p. 1503)
- *wxSplitterWindow* (p. 1508)
- *wxStaticBitmap* (p. 1527)
- *wxStaticBox* (p. 1530)
- *wxStaticBoxSizer* (p. 1532)
- *wxStaticLine* (p. 1532)
- *wxStaticText* (p. 1534)
- *wxStatusBar* (p. 1536)
- *wxSysColourChangedEvent* (p. 1590)
- *wxTaskBarIcon* (p. 1607)
- *wxTextCtrl* (p. 1633)
- *wxTextDataObject* (p. 1653)
- *wxTextDropTarget* (p. 1654)
- *wxTextEntryDialog* (p. 1655)
- *wxTimer* (p. 1680)
- *wxTimerEvent* (p. 1682)
- *wxTimeSpan* (p. 1683)
- *wxTipProvider* (p. 1689)
- *wxToolBarTool*
- *wxToolBar* (p. 1694)
- *wxToolTip* (p. 1712)
- *wxTreeCtrl* (p. 1728)
- *wxTreeEvent* (p. 1748)
- *wxTreeItemData* (p. 1751)

- `wxTreeItemId`
- `wxUpdateUIEvent` (p. 1752)
- `wxValidator` (p. 1767)
- `wxWindowDC` (p. 1855)
- `wxWindow` (p. 1795)
- `wxZipFSHandler` (p. 2075)

### Where to go for help

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

<http://lists.wxwindows.org/mailman/listinfo/wxpython-users>  
(<http://lists.wxwindows.org/mailman/listinfo/wxpython-users>)

Or you can send mail directly to the list using this address:

[wxpython-users@lists.wxwindows.org](mailto:wxpython-users@lists.wxwindows.org)

## Syntax of the builtin regular expression library

A *regular expression* describes strings of characters. It's a pattern that matches certain strings and doesn't match others.

### See also

`wxRegEx` (p. 1260)

### Different Flavors of REs

*Syntax of the builtin regular expression library* (p. 2210)

Regular expressions ("RE"s), as defined by POSIX, come in two flavors: *extended* REs ("EREs") and *basic* REs ("BREs"). EREs are roughly those of the traditional *egrep*, while BREs are roughly those of the traditional *ed*. This implementation adds a third flavor, *advanced* REs ("AREs"), basically EREs with some significant extensions.

This manual page primarily describes AREs. BREs mostly exist for backward compatibility in some old programs; they will be discussed at the *end* (p. 2219). POSIX EREs are almost an exact subset of AREs. Features of AREs that are not present in EREs will be indicated.

### Regular Expression Syntax

*Syntax of the builtin regular expression library* (p. 2210)

These regular expressions are implemented using the package written by Henry Spencer, based on the 1003.2 spec and some (not quite all) of the Perl5 extensions (thanks, Henry!). Much of the description of regular expressions below is copied verbatim from his manual entry.

An ARE is one or more *branches*, separated by '|', matching anything that matches any of the branches.

A branch is zero or more *constraints* or *quantified atoms*, concatenated. It matches a match for the first, followed by a match for the second, etc; an empty branch matches the empty string.

A quantified atom is an *atom* possibly followed by a single *quantifier*. Without a quantifier, it matches a match for the atom. The quantifiers, and what a so-quantified atom matches, are:

<b>*</b>	a sequence of 0 or more matches of the atom
<b>+</b>	a sequence of 1 or more matches of the atom
<b>?</b>	a sequence of 0 or 1 matches of the atom
<b>{m}</b>	a sequence of exactly <i>m</i> matches of the atom
<b>{m,}</b>	a sequence of <i>m</i> or more matches of the atom
<b>{m,n}</b>	a sequence of <i>m</i> through <i>n</i> (inclusive) matches of the atom; <i>m</i> may not exceed <i>n</i>
<b>*? +? ?? {m}? {m,}? {m,n}?</b>	<i>non-greedy</i> quantifiers, which match the same possibilities, but prefer the smallest number rather than the largest number of matches (see <i>Matching</i> (p. 2217))

The forms using { and } are known as *bounds*. The numbers *m* and *n* are unsigned decimal integers with permissible values from 0 to 255 inclusive. An atom is one of:

<b>(re)</b>	(where <i>re</i> is any regular expression) matches a match for <i>re</i> , with the match noted for possible reporting
<b>(?:re)</b>	as previous, but does no reporting (a "non-capturing" set of parentheses)
<b>()</b>	matches an empty string, noted for possible reporting
<b>(?:)</b>	matches an empty string, without reporting
<b>[chars]</b>	a <i>bracket expression</i> , matching any one of the <i>chars</i> (see <i>Bracket Expressions</i> (p. 2212) for more detail)
<b>.</b>	matches any single character
<b>\k</b>	(where <i>k</i> is a non-alphanumeric character) matches that character taken as an ordinary character, e.g. \ matches a backslash

	character
<b>\c</b>	where <i>c</i> is alphanumeric (possibly followed by other characters), an <i>escape</i> (AREs only), see <i>Escapes</i> (p. 2214) below
<b>{</b>	when followed by a character other than a digit, matches the left-brace character '{'; when followed by a digit, it is the beginning of a <i>bound</i> (see above)
<b>x</b>	where <i>x</i> is a single character with no other significance, matches that character.

A *constraint* matches an empty string when specific conditions are met. A constraint may not be followed by a quantifier. The simple constraints are as follows; some more constraints are described later, under *Escapes* (p. 2214).

<b>^</b>	matches at the beginning of a line
<b>\$</b>	matches at the end of a line
<b>(?=re)</b>	<i>positive lookahead</i> (AREs only), matches at any point where a substring matching <i>re</i> begins
<b>(?!re)</b>	<i>negative lookahead</i> (AREs only), matches at any point where no substring matching <i>re</i> begins

The lookahead constraints may not contain back references (see later), and all parentheses within them are considered non-capturing.

An RE may not end with '\.

## Bracket Expressions

*Syntax of the builtin regular expression library* (p. 2210)

A *bracket expression* is a list of characters enclosed in '['']. It normally matches any single character from the list (but see below). If the list begins with '^', it matches any single character (but see below) *not* from the rest of the list.

If two characters in the list are separated by '-', this is shorthand for the full *range* of characters between those two (inclusive) in the collating sequence, e.g. **[0-9]** in ASCII matches any decimal digit. Two ranges may not share an endpoint, so e.g. **a-c-e** is illegal. Ranges are very collating-sequence-dependent, and portable programs should avoid relying on them.

To include a literal **]** or **-** in the list, the simplest method is to enclose it in **[.** and **.]** to make it a collating element (see below). Alternatively, make it the first character (following a possible '^'), or (AREs only) precede it with '\. Alternatively, for '-', make it the last character, or the second endpoint of a range. To use a literal **-** as the first endpoint of a range, make it a collating element or (AREs only) precede it with '\. With the exception of these, some combinations using **[** (see next paragraphs), and escapes, all other special characters lose their special significance within a bracket expression.

Within a bracket expression, a collating element (a character, a multi-character sequence that collates as if it were a single character, or a collating-sequence name for either) enclosed in `[.` and `.]` stands for the sequence of characters of that collating element.

*wxWidgets*: Currently no multi-character collating elements are defined. So in `[.X.]`, `X` can either be a single character literal or the name of a character. For example, the following are both identical `[[.0.]-[.9.]]` and `[[.zero.]-[.nine.]]` and mean the same as `[0-9]`. See *Character Names* (p. 2219).

Within a bracket expression, a collating element enclosed in `[=` and `=]` is an equivalence class, standing for the sequences of characters of all collating elements equivalent to that one, including itself. An equivalence class may not be an endpoint of a range.

*wxWidgets*: Currently no equivalence classes are defined, so `[=X=]` stands for just the single character `X`. `X` can either be a single character literal or the name of a character, see *Character Names* (p. 2219).

Within a bracket expression, the name of a *character class* enclosed in `[:` and `:]` stands for the list of all characters (not all collating elements!) belonging to that class. Standard character classes are:

<b>alpha</b>	A letter.
<b>upper</b>	An upper-case letter.
<b>lower</b>	A lower-case letter.
<b>digit</b>	A decimal digit.
<b>xdigit</b>	A hexadecimal digit.
<b>alnum</b>	An alphanumeric (letter or digit).
<b>print</b>	An alphanumeric (same as <code>alnum</code> ).
<b>blank</b>	A space or tab character.
<b>space</b>	A character producing white space in displayed text.
<b>punct</b>	A punctuation character.
<b>graph</b>	A character with a visible representation.
<b>cntrl</b>	A control character.

A character class may not be used as an endpoint of a range.

*wxWidgets*: In a non-Unicode build, these character classifications depend on the current locale, and correspond to the values return by the ANSI C 'is' functions: `isalpha`, `isupper`, etc. In Unicode mode they are based on Unicode classifications, and are not affected by the current locale.

There are two special cases of bracket expressions: the bracket expressions `[[<:]]` and `[[>:]]` are constraints, matching empty strings at the beginning and end of a word respectively. A word is defined as a sequence of word characters that is neither preceded

nor followed by word characters. A word character is an *alnum* character or an underscore (`_`). These special bracket expressions are deprecated; users of AREs should use constraint escapes instead (see *Escapes* (p. 2214) below).

## Escapes

*Syntax of the builtin regular expression library* (p. 2210)

Escapes (AREs only), which begin with a `\` followed by an alphanumeric character, come in several varieties: character entry, class shorthands, constraint escapes, and back references. A `\` followed by an alphanumeric character but not constituting a valid escape is illegal in AREs. In EREs, there are no escapes: outside a bracket expression, a `\` followed by an alphanumeric character merely stands for that character as an ordinary character, and inside a bracket expression, `\` is an ordinary character. (The latter is the one actual incompatibility between EREs and AREs.)

Character-entry escapes (AREs only) exist to make it easier to specify non-printing and otherwise inconvenient characters in REs:

<code>\a</code>	alert (bell) character, as in C
<code>\b</code>	backspace, as in C
<code>\B</code>	synonym for <code>\</code> to help reduce backslash doubling in some applications where there are multiple levels of backslash processing
<code>\cX</code>	(where <i>X</i> is any character) the character whose low-order 5 bits are the same as those of <i>X</i> , and whose other bits are all zero
<code>\e</code>	the character whose collating-sequence name is <b>'ESC'</b> , or failing that, the character with octal value 033
<code>\f</code>	formfeed, as in C
<code>\n</code>	newline, as in C
<code>\r</code>	carriage return, as in C
<code>\t</code>	horizontal tab, as in C
<code>\uwxyz</code>	(where <i>wxyz</i> is exactly four hexadecimal digits) the Unicode character <b>U+<i>wxyz</i></b> in the local byte ordering
<code>\Ustuvwxyz</code>	(where <i>stuvwxyz</i> is exactly eight hexadecimal digits) reserved for a somewhat-hypothetical Unicode extension to 32 bits
<code>\v</code>	vertical tab, as in C are all available.
<code>\xhhh</code>	(where <i>hhh</i> is any sequence of hexadecimal digits) the character whose hexadecimal value is <b>0x<i>hhh</i></b> (a single character no matter how many hexadecimal digits are used).
<code>\0</code>	the character whose value is <b>0</b>

<b>\xy</b>	(where <i>xy</i> is exactly two octal digits, and is not a <i>back reference</i> (see below)) the character whose octal value is <b>0xy</b>
<b>\xyz</b>	(where <i>xyz</i> is exactly three octal digits, and is not a <i>back reference</i> (see below)) the character whose octal value is <b>0xyz</b>

Hexadecimal digits are '0'-'9', 'a'-'f', and 'A'-'F'. Octal digits are '0'-'7'.

The character-entry escapes are always taken as ordinary characters. For example, **\135** is **]** in ASCII, but **\135** does not terminate a bracket expression. Beware, however, that some applications (e.g., C compilers) interpret such sequences themselves before the regular-expression package gets to see them, which may require doubling (quadrupling, etc.) the **\**.

Class-shorthand escapes (AREs only) provide shorthands for certain commonly-used character classes:

<b>\d</b>	<b>[[:digit:]]</b>
<b>\s</b>	<b>[[:space:]]</b>
<b>\w</b>	<b>[[:alnum:]]</b> (note underscore)
<b>\D</b>	<b>[^[:digit:]]</b>
<b>\S</b>	<b>[^[:space:]]</b>
<b>\W</b>	<b>[^[:alnum:]]</b> (note underscore)

Within bracket expressions, **\d**, **\s**, and **\w** lose their outer brackets, and **\D**, **\S**, and **\W** are illegal. (So, for example, **[a-c\d]** is equivalent to **[a-c[:digit:]]**. Also, **[a-c\D]**, which is equivalent to **[a-c^[:digit:]]**, is illegal.)

A constraint escape (AREs only) is a constraint, matching the empty string if specific conditions are met, written as an escape:

<b>\A</b>	matches only at the beginning of the string (see <i>Matching</i> (p. 2217), below, for how this differs from <b>\^</b> )
<b>\m</b>	matches only at the beginning of a word
<b>\M</b>	matches only at the end of a word
<b>\y</b>	matches only at the beginning or end of a word
<b>\Y</b>	matches only at a point that is not the beginning or end of a word
<b>\Z</b>	matches only at the end of the string (see <i>Matching</i> (p. 2217), below, for how this differs from <b>\\$</b> )
<b>\m</b>	(where <i>m</i> is a nonzero digit) a <i>back reference</i> , see below
<b>\mnn</b>	(where <i>m</i> is a nonzero digit, and <i>nn</i> is some more digits, and the decimal value <i>mnn</i> is not greater than the number of closing capturing parentheses seen so far) a <i>back reference</i> , see below

A word is defined as in the specification of `[[:<:]]` and `[[:>:]]` above. Constraint escapes are illegal within bracket expressions.

A back reference (AREs only) matches the same string matched by the parenthesized subexpression specified by the number, so that (e.g.) `([bc])\1` matches `bb` or `cc` but not `'bc'`. The subexpression must entirely precede the back reference in the RE.

Subexpressions are numbered in the order of their leading parentheses. Non-capturing parentheses do not define subexpressions.

There is an inherent historical ambiguity between octal character-entry escapes and back references, which is resolved by heuristics, as hinted at above. A leading zero always indicates an octal escape. A single non-zero digit, not followed by another digit, is always taken as a back reference. A multi-digit sequence not starting with a zero is taken as a back reference if it comes after a suitable subexpression (i.e. the number is in the legal range for a back reference), and otherwise is taken as octal.

## Metasyntax

*Syntax of the builtin regular expression library* (p. 2210)

In addition to the main syntax described above, there are some special forms and miscellaneous syntactic facilities available.

Normally the flavor of RE being used is specified by application-dependent means. However, this can be overridden by a *director*. If an RE of any flavor begins with `'***:'`, the rest of the RE is an ARE. If an RE of any flavor begins with `'***='`, the rest of the RE is taken to be a literal string, with all characters considered ordinary characters.

An ARE may begin with *embedded options*: a sequence `(?xyz)` (where `xyz` is one or more alphabetic characters) specifies options affecting the rest of the RE. These supplement, and can override, any options specified by the application. The available option letters are:

<b>b</b>	rest of RE is a BRE
<b>c</b>	case-sensitive matching (usual default)
<b>e</b>	rest of RE is an ERE
<b>i</b>	case-insensitive matching (see <i>Matching</i> (p. 2217), below)
<b>m</b>	historical synonym for <b>n</b>
<b>n</b>	newline-sensitive matching (see <i>Matching</i> (p. 2217), below)
<b>p</b>	partial newline-sensitive matching (see <i>Matching</i> (p. 2217), below)
<b>q</b>	rest of RE is a literal ("quoted") string, all ordinary characters
<b>s</b>	non-newline-sensitive matching (usual default)
<b>t</b>	tight syntax (usual default; see below)
<b>w</b>	inverse partial newline-sensitive ("weird") matching (see <i>Matching</i> (p. 2217), below)



**x** expanded syntax (see below)

Embedded options take effect at the `)` terminating the sequence. They are available only at the start of an ARE, and may not be used later within it.

In addition to the usual (*tight*) RE syntax, in which all characters are significant, there is an *expanded* syntax, available in AREs with the embedded `x` option. In the expanded syntax, white-space characters are ignored and all characters between a `#` and the following newline (or the end of the RE) are ignored, permitting paragraphing and commenting a complex RE. There are three exceptions to that basic rule:

- white-space character or `#` preceded by `\` is retained
- space or `#` within a bracket expression is retained
- space and comments are illegal within multi-character symbols like the ARE `(?:` or the BRE `\(` Expanded-syntax white-space characters are blank, tab, newline, and any character that belongs to the *space* character class.

Finally, in an ARE, outside bracket expressions, the sequence `(?#ttt)` (where *ttt* is any text not containing a `)`) is a comment, completely ignored. Again, this is not allowed between the characters of multi-character symbols like `(?:`. Such comments are more a historical artifact than a useful facility, and their use is deprecated; use the expanded syntax instead.

None of these metasyntax extensions is available if the application (or an initial `***=director`) has specified that the user's input be treated as a literal string rather than as an RE.

## Matching

*Syntax of the builtin regular expression library* (p. 2210)

In the event that an RE could match more than one substring of a given string, the RE matches the one starting earliest in the string. If the RE could match more than one substring starting at that point, its choice is determined by its *preference*: either the longest substring, or the shortest.

Most atoms, and all constraints, have no preference. A parenthesized RE has the same preference (possibly none) as the RE. A quantified atom with quantifier `{m}` or `{m}?` has the same preference (possibly none) as the atom itself. A quantified atom with other normal quantifiers (including `{m,n}` with *m* equal to *n*) prefers longest match. A quantified atom with other non-greedy quantifiers (including `{m,n}?` with *m* equal to *n*) prefers shortest match. A branch has the same preference as the first quantified atom in it which has a preference. An RE consisting of two or more branches connected by the `|` operator prefers longest match.

Subject to the constraints imposed by the rules for matching the whole RE, subexpressions also match the longest or shortest possible substrings, based on their preferences, with subexpressions starting earlier in the RE taking priority over ones starting later. Note that outer subexpressions thus take priority over their component subexpressions.

Note that the quantifiers **{1,1}** and **{1,1}?** can be used to force longest and shortest preference, respectively, on a subexpression or a whole RE.

Match lengths are measured in characters, not collating elements. An empty string is considered longer than no match at all. For example, **bb\*** matches the three middle characters of **'abbbbc'**, **(week|wee)(night|knights)** matches all ten characters of **'weeknights'**, when **(.\*)** is matched against **abc** the parenthesized subexpression matches all three characters, and when **(a\*)** is matched against **bc** both the whole RE and the parenthesized subexpression match an empty string.

If case-independent matching is specified, the effect is much as if all case distinctions had vanished from the alphabet. When an alphabetic that exists in multiple cases appears as an ordinary character outside a bracket expression, it is effectively transformed into a bracket expression containing both cases, so that **x** becomes **'[xX]'**. When it appears inside a bracket expression, all case counterparts of it are added to the bracket expression, so that **[x]** becomes **[xX]** and **[^x]** becomes **[^xX]**.

If newline-sensitive matching is specified, **.** and bracket expressions using **^** will never match the newline character (so that matches will never cross newlines unless the RE explicitly arranges it) and **^** and **\$** will match the empty string after and before a newline respectively, in addition to matching at beginning and end of string respectively. ARE **\A** and **\Z** continue to match beginning or end of string *only*.

If partial newline-sensitive matching is specified, this affects **.** and bracket expressions as with newline-sensitive matching, but not **^** and **\$**.

If inverse partial newline-sensitive matching is specified, this affects **^** and **\$** as with newline-sensitive matching, but not **.** and bracket expressions. This isn't very useful but is provided for symmetry.

## Limits And Compatibility

*Syntax of the builtin regular expression library* (p. 2210)

No particular limit is imposed on the length of REs. Programs intended to be highly portable should not employ REs longer than 256 bytes, as a POSIX-compliant implementation can refuse to accept such REs.

The only feature of AREs that is actually incompatible with POSIX EREs is that **\** does not lose its special significance inside bracket expressions. All other ARE features use syntax which is illegal or has undefined or unspecified effects in POSIX EREs; the **\*\*\*** syntax of directors likewise is outside the POSIX syntax for both BREs and EREs.

Many of the ARE extensions are borrowed from Perl, but some have been changed to clean them up, and a few Perl extensions are not present. Incompatibilities of note include **\b**, **\B**, the lack of special treatment for a trailing newline, the addition of complemented bracket expressions to the things affected by newline-sensitive matching, the restrictions on parentheses and back references in lookahead constraints, and the longest/shortest-match (rather than first-match) matching semantics.

The matching rules for REs containing both normal and non-greedy quantifiers have changed since early beta-test versions of this package. (The new rules are much simpler

and cleaner, but don't work as hard at guessing the user's real intentions.)

Henry Spencer's original 1986 *regex* package, still in widespread use, implemented an early version of today's EREs. There are four incompatibilities between *regex*'s near-EREs ('RREs' for short) and AREs. In roughly increasing order of significance:

- In AREs, `\` followed by an alphanumeric character is either an escape or an error, while in RREs, it was just another way of writing the alphanumeric. This should not be a problem because there was no reason to write such a sequence in RREs.
- `{` followed by a digit in an ARE is the beginning of a bound, while in RREs, `{` was always an ordinary character. Such sequences should be rare, and will often result in an error because following characters will not look like a valid bound.
- In AREs, `\` remains a special character within `[]`, so a literal `\` within `[]` must be written `\\`. `\\` also gives a literal `\` within `[]` in RREs, but only truly paranoid programmers routinely doubled the backslash.
- AREs report the longest/shortest match for the RE, rather than the first found in a specified search order. This may affect some RREs which were written in the expectation that the first match would be reported. (The careful crafting of RREs to optimize the search order for fast matching is obsolete (AREs examine all possible matches in parallel, and their performance is largely insensitive to their complexity) but cases where the search order was exploited to deliberately find a match which was *not* the longest/shortest will need rewriting.)

## Basic Regular Expressions

*Syntax of the builtin regular expression library* (p. 2210)

BREs differ from EREs in several respects. `|`, `+`, and `?` are ordinary characters and there is no equivalent for their functionality. The delimiters for bounds are `\{` and `\}`, with `{` and `}` by themselves ordinary characters. The parentheses for nested subexpressions are `\(` and `\)`, with `(` and `)` by themselves ordinary characters. `^` is an ordinary character except at the beginning of the RE or the beginning of a parenthesized subexpression, `$` is an ordinary character except at the end of the RE or the end of a parenthesized subexpression, and `*` is an ordinary character if it appears at the beginning of the RE or the beginning of a parenthesized subexpression (after a possible leading `^`). Finally, single-digit back references are available, and `\<` and `\>` are synonyms for `[[<:]]` and `[[>:]]` respectively; no other escapes are available.

## Regular Expression Character Names

*Syntax of the builtin regular expression library* (p. 2210)

Note that the character names are case sensitive.

NUL	'\0'
SOH	'\001'
STX	'\002'
ETX	'\003'
EOT	'\004'
ENQ	'\005'
ACK	'\006'
BEL	'\007'
alert	'\007'
BS	'\010'
backspace	'\b'
HT	'\011'
tab	'\t'
LF	'\012'
newline	'\n'
VT	'\013'
vertical-tab	'\v'
FF	'\014'
form-feed	'\f'
CR	'\015'
carriage-return	'\r'
SO	'\016'
SI	'\017'
DLE	'\020'
DC1	'\021'
DC2	'\022'
DC3	'\023'
DC4	'\024'
NAK	'\025'

SYN	'\026'
ETB	'\027'
CAN	'\030'
EM	'\031'
SUB	'\032'
ESC	'\033'
IS4	'\034'
FS	'\034'
IS3	'\035'
GS	'\035'
IS2	'\036'
RS	'\036'
IS1	'\037'
US	'\037'
space	' '
exclamation-mark	'!'
quotation-mark	'\"'
number-sign	'#'
dollar-sign	'\$'
percent-sign	'%'
ampersand	'&'
apostrophe	'\''
left-parenthesis	'('
right-parenthesis	')'
asterisk	'*'
plus-sign	'+'
comma	','
hyphen	'-'
hyphen-minus	'-'

period	'.'
full-stop	'.'
slash	'/'
solidus	'/'
zero	'0'
one	'1'
two	'2'
three	'3'
four	'4'
five	'5'
six	'6'
seven	'7'
eight	'8'
nine	'9'
colon	':'
semicolon	','
less-than-sign	'<'
equals-sign	'='
greater-than-sign	'>'
question-mark	'?'
commercial-at	'@'
left-square-bracket	'['
backslash	'\'
reverse-solidus	'\'
right-square-bracket	']'
circumflex	'^'
circumflex-accent	'^'
underscore	'_'
low-line	'_'

grave-accent	''
left-brace	'{'
left-curly-bracket	'{'
vertical-line	' '
right-brace	'}'
right-curly-bracket	'}'
tilde	'~'
DEL	'\177'

## Archive formats such as zip

The archive classes handle archive formats such as zip, tar, rar and cab. Currently *wxZip* (p. 1895) and *wxTar* (p. 1603) classes are included.

For each archive type, there are the following classes (using zip here as an example):

*wxZipInputStream* (p. 1895) Input stream

*wxZipOutputStream* (p. 1897) Output stream

*wxZipEntry* (p. 1888) Holds the meta-data for an entry (e.g. filename, timestamp, etc.)

There are also abstract *wxArchive* classes that can be used to write code that can handle any of the archive types, see '*Generic archive programming* (p. 2227)'. Also see *wxFileSystem* (p. 2075) for a higher level interface that can handle archive files in a generic way.

The classes are designed to handle archives on both seekable streams such as disk files, or non-seekable streams such as pipes and sockets (see '*Archives on non-seekable streams* (p. 2228)').

### See also

*wxFileSystem* (p. 2075)

## Creating an archive

*Archive formats such as zip* (p. 2223)

Call *PutNextEntry()* (p. 70) to create each new entry in the archive, then write the entry's data. Another call to *PutNextEntry()* closes the current entry and begins the next.

For example:

```
wxFileOutputStream out(_T("test.zip"));
wxZipOutputStream zip(out);
wxTextOutputStream txt(zip);
```

```
wxString sep(wxFileName::GetPathSeparator());

zip.PutNextEntry(_T("entry1.txt"));
txt << _T("Some text for entry1.txt\n");

zip.PutNextEntry(_T("subdir") + sep + _T("entry2.txt"));
txt << _T("Some text for subdir/entry2.txt\n");
```

The name of each entry can be a full path, which makes it possible to store entries in subdirectories.

## Extracting an archive

*Archive formats such as zip* (p. 2223)

*GetNextEntry()* (p. 65) returns a pointer to entry object containing the meta-data for the next entry in the archive (and gives away ownership). Reading from the input stream then returns the entry's data. *Eof()* becomes true after an attempt has been made to read past the end of the entry's data.

When there are no more entries, *GetNextEntry()* returns NULL and sets *Eof()*.

```
auto_ptr<wxZipEntry> entry;

wxFFFileInputStream in(_T("test.zip"));
wxZipInputStream zip(in);

while (entry.reset(zip.GetNextEntry()), entry.get() != NULL)
{
    // access meta-data
    wxString name = entry->GetName();
    // read 'zip' to access the entry's data
}
```

## Modifying an archive

*Archive formats such as zip* (p. 2223)

To modify an existing archive, write a new copy of the archive to a new file, making any necessary changes along the way and transferring any unchanged entries using *CopyEntry()* (p. 70). For archive types which compress entry data, *CopyEntry()* is likely to be much more efficient than transferring the data using *Read()* and *Write()* since it will copy them without decompressing and recompressing them.

In general modifications are not possible without rewriting the archive, though it may be possible in some limited cases. Even then, rewriting the archive is usually a better choice since a failure can be handled without losing the whole archive. *wxTempFileOutputStream* (p. 1618) can be helpful to do this.

For example to delete all entries matching the pattern `"*.txt"`:



```
    auto_ptr<wxFFFileInputStream> in(new
wxFFFileInputStream(_T("test.zip")));
    wxTempFileOutputStream out(_T("test.zip"));

    wxZipInputStream inzip(*in);
    wxZipOutputStream outzip(out);

    auto_ptr<wxZipEntry> entry;

    // transfer any meta-data for the archive as a whole (the zip
comment
    // in the case of zip)
    outzip.CopyArchiveMetaData(inzip);

    // call CopyEntry for each entry except those matching the pattern
    while (entry.reset(inzip.GetNextEntry()), entry.get() != NULL)
        if (!entry->GetName().Matches(_T("*.txt")))
            if (!outzip.CopyEntry(entry.release(), inzip))
                break;

    // close the input stream by releasing the pointer to it, do this
    // before closing the output stream so that the file can be replaced
    in.reset();

    // you can check for success as follows
    bool success = inzip.Eof() && outzip.Close() && out.Commit();
```

## Looking up an archive entry by name

*Archive formats such as zip* (p. 2223)

Also see *wxFileSystem* (p. 2075) for a higher level interface that is more convenient for accessing archive entries by name.

To open just one entry in an archive, the most efficient way is to simply search for it linearly by calling *GetNextEntry()* (p. 65) until the required entry is found. This works both for archives on seekable and non-seekable streams.

The format of filenames in the archive is likely to be different from the local filename format. For example zips and tars use unix style names, with forward slashes as the path separator, and absolute paths are not allowed. So if on Windows the file "C:\MYDIR\MYFILE.TXT" is stored, then when reading the entry back *GetName()* (p. 63) will return "MYDIR\MYFILE.TXT". The conversion into the internal format and back has lost some information.

So to avoid ambiguity when searching for an entry matching a local name, it is better to convert the local name to the archive's internal format and search for that:

```
    auto_ptr<wxZipEntry> entry;

    // convert the local name we are looking for into the internal
format
    wxString name = wxZipEntry::GetInternalName(localname);
```

```
// open the zip
wxFFFileInputStream in(_T("test.zip"));
wxZipInputStream zip(in);

// call GetNextEntry() until the required internal name is found
do {
    entry.reset(zip.GetNextEntry());
}
while (entry.get() != NULL && entry->GetInternalName() != name);

if (entry.get() != NULL) {
    // read the entry's data...
}
```

To access several entries randomly, it is most efficient to transfer the entire catalogue of entries to a container such as a `std::map` or a *wxHashMap* (p. 798) then entries looked up by name can be opened using the *OpenEntry()* (p. 65) method.

```
WX_DECLARE_STRING_HASH_MAP(wxZipEntry*, ZipCatalog);
ZipCatalog::iterator it;
wxZipEntry *entry;
ZipCatalog cat;

// open the zip
wxFFFileInputStream in(_T("test.zip"));
wxZipInputStream zip(in);

// load the zip catalog
while ((entry = zip.GetNextEntry()) != NULL) {
    wxZipEntry*& current = cat[entry->GetInternalName()];
    // some archive formats can have multiple entries with the same
name    // (e.g. tar) though it is an error in the case of zip
    delete current;
    current = entry;
}

// open an entry by name
if ((it = cat.find(wxZipEntry::GetInternalName(localname))) !=
cat.end()) {
    zip.OpenEntry(*it->second);
    // ... now read entry's data
}
```

To open more than one entry simultaneously you need more than one underlying stream on the same archive:

```
// opening another entry without closing the first requires
another
// input stream for the same file
wxFFFileInputStream in2(_T("test.zip"));
wxZipInputStream zip2(in2);
if ((it = cat.find(wxZipEntry::GetInternalName(local2))) !=
cat.end())
```

```
zip2.OpenEntry(*it->second);
```

## Generic archive programming

*Archive formats such as zip* (p. 2223)

Also see *wxFileSystem* (p. 2075) for a higher level interface that can handle archive files in a generic way.

The specific archive classes, such as the *wxZip* classes, inherit from the following abstract classes which can be used to write code that can handle any of the archive types:

*wxArchiveInputStream* (p. 64)      Input stream

*wxArchiveOutputStream* (p. 69)      Output stream

*wxArchiveEntry* (p. 62) Holds the meta-data for an entry (e.g. filename)

In order to be able to write generic code it's necessary to be able to create instances of the classes without knowing which archive type is being used. To allow this there is a class factory for each archive type, derived from *wxArchiveClassFactory* (p. 58), that can create the other classes.

For example, given *wxArchiveClassFactory\* factory*, streams and entries can be created like this:

```
// create streams without knowing their type
auto_ptr<wxArchiveInputStream> inarc(factory->NewStream(in));
auto_ptr<wxArchiveOutputStream>
outarc(factory->NewStream(out));

// create an empty entry object
auto_ptr<wxArchiveEntry> entry(factory->NewEntry());
```

For the factory itself, the static member *wxArchiveClassFactory::Find()* (p. 59). can be used to find a class factory that can handle a given file extension or mime type. For example, given *filename*:

```
const wxArchiveClassFactory *factory;
factory = wxArchiveClassFactory::Find(filename,
wxSTREAM_FILEEXT);

if (factory)
    stream = factory->NewStream(new
wxFFileInputStream(filename));
```

*Find* does not give away ownership of the returned pointer, so it does not need to be deleted.

There are similar class factories for the filter streams that handle the compression and decompression of a single stream, such as *wxGzipInputStream*. These can be found

using `wxFilterClassFactory::Find()` (p. 644).

For example, to list the contents of archive *filename*:

```
auto_ptr<wxInputStream> in(new wxFFileInputStream(filename));

if (in->IsOk())
{
    // look for a filter handler, e.g. for '.gz'
    const wxFilterClassFactory *fcf;
    fcf = wxFilterClassFactory::Find(filename,
wxSTREAM_FILEEXT);
    if (fcf) {
        in.reset(fcf->NewStream(in.release()));
        // pop the extension, so if it was '.tar.gz' it is now just
        '.tar'
        filename = fcf->PopExtension(filename);
    }

    // look for a archive handler, e.g. for '.zip' or '.tar'
    const wxArchiveClassFactory *acf;
    acf = wxArchiveClassFactory::Find(filename,
wxSTREAM_FILEEXT);
    if (acf) {
        auto_ptr<wxArchiveInputStream>
arc(acf->NewStream(in.release()));
        auto_ptr<wxArchiveEntry> entry;

        // list the contents of the archive
        while ((entry.reset(arc->GetNextEntry())),
entry.get() != NULL)
            std::wcout << entry->GetName().c_str() << "\n";
    }
    else {
        wxLogError(_T("can't handle '%s'"), filename.c_str());
    }
}
```

## Archives on non-seekable streams

*Archive formats such as zip* (p. 2223)

In general, handling archives on non-seekable streams is done in the same way as for seekable streams, with a few caveats.

The main limitation is that accessing entries randomly using `OpenEntry()` (p. 65) is not possible, the entries can only be accessed sequentially in the order they are stored within the archive.

For each archive type, there will also be other limitations which will depend on the order the entries' meta-data is stored within the archive. These are not too difficult to deal with, and are outlined below.

### PutNextEntry and the entry size

When writing archives, some archive formats store the entry size before the entry's data (tar has this limitation, zip doesn't). In this case the entry's size must be passed to *PutNextEntry()* (p. 70) or an error occurs.

This is only an issue on non-seekable streams, since otherwise the archive output stream can seek back and fix up the header once the size of the entry is known.

For generic programming, one way to handle this is to supply the size whenever it is known, and rely on the error message from the output stream when the operation is not supported.

### **GetNextEntry and the weak reference mechanism**

Some archive formats do not store all an entry's meta-data before the entry's data (zip is an example). In this case, when reading from a non-seekable stream, *GetNextEntry()* (p. 65) can only return a partially populated *wxArchiveEntry* (p. 62) object - not all the fields are set.

The input stream then keeps a weak reference to the entry object and updates it when more meta-data becomes available. A weak reference being one that does not prevent you from deleting the *wxArchiveEntry* object - the input stream only attempts to update it if it is still around.

The documentation for each archive entry type gives the details of what meta-data becomes available and when. For generic programming, when the worst case must be assumed, you can rely on all the fields of *wxArchiveEntry* being fully populated when *GetNextEntry()* returns, with the following exceptions:

*GetSize()* (p. 63) Guaranteed to be available after the entry has been read to *Eof()* (p. 942), or *CloseEntry()* (p. 65) has been called

*IsReadOnly()* (p. 64) Guaranteed to be available after the end of the archive has been reached, i.e. after *GetNextEntry()* returns NULL and *Eof()* is true

This mechanism allows *CopyEntry()* (p. 70) to always fully preserve entries' meta-data. No matter what order the meta-data occurs within the archive, the input stream will always have read it before the output stream must write it.

### **wxArchiveNotifier**

Notifier objects can be used to get a notification whenever an input stream updates a *wxArchiveEntry* (p. 62) object's data via the weak reference mechanism.

Consider the following code which renames an entry in an archive. This is the usual way to modify an entry's meta-data, simply set the required field before writing it with *CopyEntry()* (p. 70):

```
auto_ptr<wxArchiveInputStream> arc(factory->NewStream(in));
auto_ptr<wxArchiveOutputStream>
outarc(factory->NewStream(out));
auto_ptr<wxArchiveEntry> entry;

outarc->CopyArchiveMetaData(*arc);
```

```
while (entry.reset(arc->GetNextEntry()), entry.get() != NULL) {
    if (entry->GetName() == from)
        entry->SetName(to);
    if (!outarc->CopyEntry(entry.release(), *arc))
        break;
}

bool success = arc->Eof() && outarc->Close();
```

However, for non-seekable streams, this technique cannot be used for fields such as *IsReadOnly()* (p. 64), which are not necessarily set when *GetNextEntry()* (p. 65) returns. In this case a *wxArchiveNotifier* (p. 68) can be used:

```
class MyNotifier : public wxArchiveNotifier
{
public:
    void OnEntryUpdated(wxArchiveEntry& entry)
    { entry.SetIsReadOnly(false); }
};
```

The meta-data changes are done in your notifier's *OnEntryUpdated()* (p. 68) method, then *SetNotifier()* (p. 64) is called before *CopyEntry()*:

```
auto_ptr<wxArchiveInputStream> arc(factory->NewStream(in));
auto_ptr<wxArchiveOutputStream>
outarc(factory->NewStream(out));
auto_ptr<wxArchiveEntry> entry;
MyNotifier notifier;

outarc->CopyArchiveMetaData(*arc);

while (entry.reset(arc->GetNextEntry()), entry.get() != NULL) {
    entry->SetNotifier(notifier);
    if (!outarc->CopyEntry(entry.release(), *arc))
        break;
}

bool success = arc->Eof() && outarc->Close();
```

*SetNotifier()* calls *OnEntryUpdated()* immediately, then the input stream calls it again whenever it sets more fields in the entry. Since *OnEntryUpdated()* will be called at least once, this technique always works even when it is not strictly necessary to use it. For example, changing the entry name can be done this way too and it works on seekable streams as well as non-seekable.

## Backward compatibility

Many of the GUIs and platforms supported by *wxWidgets* are continuously evolving, and some of the new platforms *wxWidgets* now supports were quite unimaginable even a few years ago. In this environment *wxWidgets* must also evolve in order to support these new

features and platforms.

However the goal of wxWidgets is not only to provide a consistent programming interface across many platforms, but also to provide an interface that is reasonably stable over time, to help protect its users from some of the uncertainty of the future.

### The version numbering scheme

wxWidgets version numbers can have up to four components, with trailing zeros sometimes omitted:

```
major.minor.release.sub-release
```

A *stable* release of wxWidgets will have an even number for `minor`, e.g. 2.6.0.

Stable, in this context, means that the API is not changing. In truth, some changes are permitted, but only those that are backward compatible. For example, you can expect later 2.6.x.x releases, such as 2.6.1 and 2.6.2 to be backward compatible with their predecessor.

When it becomes necessary to make changes which are not wholly backward compatible, the stable branch is forked, creating a new *development* branch of wxWidgets. This development branch will have an odd number for `minor`, for example 2.7.x.x. Releases from this branch are known as *development snapshots*.

The stable branch and the development branch will then be developed in parallel for some time. When it is no longer useful to continue developing the stable branch, the development branch is renamed and becomes a new stable branch, for example 2.8.0. And the process begins again.

This is how the tension between keeping the interface stable, and allowing the library to evolve is managed.

You can expect the versions with the same major and *even* minor version number to be compatible, but between minor versions there will be incompatibilities. Compatibility is not broken gratuitously however, so many applications will require no changes or only small changes to work with the new version.

### Source level compatibility

Later releases from a stable branch are backward compatible with earlier releases from the same branch at the *source* level.

This means that, for example, if you develop your application using wxWidgets 2.6.0 then it should also compile fine with all later 2.6.x versions. The converse is also true providing you avoid any new features not present in the earlier version. For example if you develop using 2.6.1 your program will compile fine with wxWidgets 2.6.0 providing you don't use any 2.6.1 specific features.

For some platforms binary compatibility is also supported, see 'Library binary compatibility' below.

Between minor versions, for example between 2.2.x, 2.4.x and 2.6.x, there will be some incompatibilities. Wherever possible the old way of doing something is kept alongside the new for a time wrapped inside:

```
#if WXWIN_COMPATIBILITY_2_4
    /* deprecated feature */
    ...
#endif
```

By default the `WXWIN_COMPATIBILITY_X_X` macro is set to 1 for the previous stable branch, for example in 2.6.x `WXWIN_COMPATIBILITY_2_4 = 1`. For the next earlier stable branch the default is 0, so `WXWIN_COMPATIBILITY_2_2 = 0` for 2.6.x. Earlier than that, obsolete features are removed.

These macros can be changed in `setup.h`. Or on UNIX-like systems you can set them using the `--disable-compat24` and `--enable-compat22` options to configure.

They can be useful in two ways:

1. Changing `WXWIN_COMPATIBILITY_2_4` to 0 can be useful to find uses of deprecated features in your program.
2. Changing `WXWIN_COMPATIBILITY_2_2` to 1 can be useful to compile a program developed using 2.2.x that no longer compiles with 2.6.x.

A program requiring one of these macros to be 1 will become incompatible with some future version of wxWidgets, and you should consider updating it.

## Library binary compatibility

For some platforms, releases from a stable branch are not only source level compatible but can also be *binary compatible*.

Binary compatibility makes it possible to get the maximum benefit from using shared libraries, also known as dynamic link libraries (DLLs) on Windows or dynamic shared libraries on OS X.

For example, suppose several applications are installed on a system requiring wxWidgets 2.6.0, 2.6.1 and 2.6.2. Since 2.6.2 is backward compatible with the earlier versions, it should be enough to install just wxWidgets 2.6.2 shared libraries, and all the applications should be able to use them. If binary compatibility is not supported, then all the required versions 2.6.0, 2.6.1 and 2.6.2 must be installed side by side.

Achieving this, without the user being required to have the source code and recompile everything, places many extra constraints on the changes that can be made within the stable branch. So it is not supported for all platforms, and not for all versions of wxWidgets. To date it has mainly been supported by wxGTK for UNIX-like platforms.

Another practical consideration is that for binary compatibility to work, all the applications and libraries must have been compiled with compilers that are capable of producing compatible code; that is, they must use the same ABI (Application Binary Interface). Unfortunately most different C++ compilers do not produce code compatible with each



other, and often even different versions of the same compiler are not compatible.

### **Application binary compatibility**

The most important aspect of binary compatibility is that applications compiled with one version of wxWidgets, e.g. 2.6.1, continue to work with shared libraries of a later binary compatible version, for example 2.6.2.

The converse can also be useful however. That is, it can be useful for a developer using a later version, e.g. 2.6.2 to be able to create binary application packages that will work with all binary compatible versions of the shared library starting with, for example 2.6.0.

To do this the developer must, of course, avoid any features not available in the earlier versions. However this is not necessarily enough; in some cases an application compiled with a later version may depend on it even though the same code would compile fine against an earlier version. To help with this, a preprocessor symbol `wxABI_VERSION` can be defined during the compilation of the application (this would usually be done in the application's makefile or project settings). It should be set to the lowest version that is being targeted, as a number with two decimal digits for each component, for example `wxABI_VERSION=20600` for 2.6.0.

Setting `wxABI_VERSION` should prevent the application from implicitly depending on a later version of wxWidgets, and also disables any new features in the API, giving a compile time check that the source is compatible with the versions of wxWidgets being targeted.

Uses of `wxABI_VERSION` are stripped out of the wxWidgets sources when each new development branch is created. Therefore it is only useful to help achieve compatibility with earlier versions with the same major and *even* minor version numbers. It won't, for example, help you write code compatible with 2.4.x using wxWidgets 2.6.x.

## Platform details

wxWidgets defines a common API across platforms, but uses the native graphical user interface (GUI) on each platform, so your program will take on the native look and feel that users are familiar with. Unfortunately native toolkits and hardware do not always support the functionality that the wxWidgets API requires. This chapter collects notes about differences among supported platforms and ports.

### wxGTK port

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

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

You will need GTK+ 2.0 or higher which is available from:

<http://www.gtk.org> (<http://www.gtk.org>)

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

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

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

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

### wxMSW port

wxMSW is a port of wxWidgets for the Windows platforms including Windows 95, 98, ME, 2000, NT, XP in ANSI and Unicode mode (for Windows 95 through the MSLU extension library). wxMSW ensures native look and feel for XP as well when using wxWidgets version 2.3.3 or higher. wxMSW can be compile with a great variety of compilers including MS VC++, Borland 5.5, MinGW32, Cygwin and Watcom as well as cross-compilation with a Linux hosted MinGW32 tool chain.

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

## Themed borders on Windows

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

If you don't specify a border style for a `wxTextCtrl` in rich edit mode, wxWidgets now gives the control themed borders automatically, where previously they would take the Windows 95-style sunken border. Other native controls such as `wxTextCtrl` in non-rich edit mode, and `wxComboBox`, already paint themed borders where appropriate. To use themed borders on other windows, such as `wxPanel`, pass the `wxBORDER_THEME` style.

Note that in wxWidgets 2.9 and above, `wxBORDER_THEME` will be used on all platforms to indicate that there should definitely be a border, whose style is determined by wxWidgets for the current platform. wxWidgets 2.9 and above will also be better at determining whether there should be a themed border. Because of the requirements of binary compatibility, this automatic border capability could not be put into wxWidgets 2.8 except for built-in, native controls. In 2.8, the border must be specified for custom controls and windows. However, to make it easier to decide what style should be passed, you can use (on Windows only) the function `wxWindow::GetThemedBorderStyle()` and it will return a suitable style for the current version of Windows. On Windows CE, `wxBORDER_SIMPLE` will be returned; on Windows 95/98/NT/2K, `wxBORDER_SUNKEN` will be returned, and on XP and above where theming is turned on, `wxBORDER_THEME` will be returned. This function does not exist in wxWidgets 2.9 and is not part of the official API.

In 2.8, `wxBORDER_THEME` is normally interpreted as a `wxBORDER_SUNKEN` on platforms other than Windows.

## wxWinCE

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

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

## General issues for wxWinCE programming

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

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

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

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

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

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

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

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

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

## Testing for WinCE SDKs

Use these preprocessor symbols to test for the different types of device or SDK:

\_\_SMARTPHONE\_\_ Generic mobile devices with phone buttons and a small display

\_\_PDA\_\_ Generic mobile devices with no phone

\_\_HANDHELDPC\_\_ Generic mobile device with a keyboard

\_\_WXWINCE\_\_ Microsoft-powered Windows CE devices, whether PocketPC, Smartphone or Standard SDK

WIN32\_PLATFORM\_WFSP Microsoft-powered smartphone

\_\_POCKETPC\_\_ Microsoft-powered PocketPC devices with touch-screen

\_\_WINCE\_STANDARDSDK\_\_ Microsoft-powered Windows CE devices, for generic Windows CE applications

\_\_WINCE\_NET\_\_ Microsoft-powered Windows CE .NET devices (\_WIN32\_WCE is 400 or greater)

wxGetOsVersion will return these values:

wxWINDOWS\_POCKETPC The application is running under PocketPC.

`wxWINDOWS_SMARTPHONE`      The application is running under Smartphone.

`wxWINDOWS_CE`      The application is running under Windows CE (built with the Standard SDK).

### Window sizing in wxWinCE

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

If the screen orientation changes, the windows will automatically be resized so no further action needs to be taken (unless you want to change the layout according to the orientation, which you could detect in idle time, for example). When input panel (SIP) is shown, top level windows (frames and dialogs) resize accordingly (see *`wxTopLevelWindow::HandleSettingChange`* (p. 1714)).

### Closing top-level windows in wxWinCE

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

### Hibernation in wxWinCE

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

### Hardware buttons in wxWinCE

Special hardware buttons are sent to a window via the `wxEVT_HOTKEY` event under Smartphone and PocketPC. You should first register each required button with *`wxWindow::RegisterHotKey`* (p. 1830), and unregister the button when you're done with it. For example:

```
win->RegisterHotKey(0, wxMOD_WIN, WVK_SPECIAL1);
win->UnregisterHotKey(0);
```

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

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

### Dialogs in wxWinCE

PocketPC dialogs have an OK button on the caption, and so you should generally not

repeat an OK button on the dialog. You can add a Cancel button if necessary, but some dialogs simply don't offer you the choice (the guidelines recommend you offer an Undo facility to make up for it). When the user clicks on the OK button, your dialog will receive a `wxID_OK` event by default. If you wish to change this, call `wxDialog::SetAffirmativeId` (p. 503) with the required identifier to be used. Or, override `wxDialog::DoOK` (p. 500) (return false to have `wxWidgets` simply call `Close` to dismiss the dialog).

Smartphone dialogs do *not* have an OK button on the caption, and are closed using one of the two menu buttons. You need to assign these using `wxTopLevelWindow::SetLeftMenu` (p. 1717) and `wxTopLevelWindow::SetRightMenu` (p. 1719), for example:

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

For implementing property sheets (flat tabs), use a `wxNotebook` with `wxNB_FLAT|wxNB_BOTTOM` and have the notebook left, top and right sides overlap the dialog by about 3 pixels to eliminate spurious borders. You can do this by using a negative spacing in your `sizer Add()` call. The cross-platform property sheet dialog `wxPropertySheetDialog` (p. 1232) is provided, to show settings in the correct style on PocketPC and on other platforms.

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

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

## Menubars and toolbars in wxWinCE

### Menubars and toolbars in PocketPC

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

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

- You must create the frame's primary toolbar with `wxFrame::CreateToolBar`, because this uses the special `wxToolMenuBar` class (derived from `wxToolBar`) to implement the combined toolbar and menubar. Otherwise, you can create and manage toolbars using the `wxToolBar` class as usual, for example to implement

an optional formatting toolbar above the menubar as Pocket Word does. But don't assign a `wxToolBar` to a frame using `SetToolBar` - you should always use `CreateToolBar` for the main frame toolbar.

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

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

### Menubars and toolbars in Smartphone

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

### Closing windows in wxWinCE

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

### Context menus in wxWinCE

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

Context menus are not supported in Smartphone.

### Control differences on wxWinCE

These controls and styles are specific to `wxWinCE`:

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

These controls are missing from `wxWinCE`:

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

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

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

### Online help in wxWinCE

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

### Installing your PocketPC and Smartphone applications

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

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

Here are some links that may help.

- A setup builder that takes CABs and builds a setup program is at <http://www.eskimo.com/~scottlu/win/index.html> (<http://www.eskimo.com/~scottlu/win/index.html>).
- Sample installation files can be found in Windows CE Tools/wce420/POCKET PC 2003/Samples/Win32/AppInst.
- An installer generator using wxPython can be found at <http://ppcquicksoft.iespana.es/ppcquicksoft/myinstall.html> (<http://ppcquicksoft.iespana.es/ppcquicksoft/myinstall.html>).
- Miscellaneous Windows CE resources can be found at <http://www.orbworks.com/pcce/resources.html> (<http://www.orbworks.com/pcce/resources.html>).
- Installer creation instructions with a setup.exe for installing to PPC can be found



at <http://www.pocketpcdn.com/articles/creatingsetup.html>

(<http://www.pocketpcdn.com/articles/creatingsetup.html>).

- Microsoft instructions are at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnce30/html/appinstall30.asp?frame=true&hidetoc=true> (<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnce30/html/appinstall30.asp?frame=true&hidetoc=true>).
- Troubleshooting WinCE application installations: <http://support.microsoft.com/default.aspx?scid=KB;en-us;q181007> (<http://support.microsoft.com/default.aspx?scid=KB;en-us;q181007>)

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

## **wxFileDialog in PocketPC**

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

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

## **Embedded Visual C++ Issues**

### **Run-time type information**

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

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

Otherwise you will get linker errors similar to this:

```
wxwince26d.lib(control.obj) : error LNK2001: unresolved external  
symbol "const type_info::~`vftable'" (??_7type_info@@6B@)
```

### **Windows Mobile 5.0 emulator**

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

### **Visual Studio 2005 project files**

Unfortunately, Visual Studio 2005, required to build Windows Mobile 5.0 applications,

doesn't do a perfect job of converting the project files from eVC++ format.

When you have converted the wxWidgets workspace, edit the configuration properties for each configuration and in the Librarian, add a relative path `..\lib` to each library path. For example: `..\$(PlatformName)\$(ConfigurationName)\wx_mono.lib`.

Then, for a sample you want to compile, edit the configuration properties and make sure `..\..\lib\$(PlatformName)\$(ConfigurationName)` is in the Linker/General/Additional Library Directories property. Also change the Linker/Input/Additional Dependencies property to something like `coredll.lib wx_mono.lib wx_wxjpeg.lib wx_wxpng.lib wx_wxzlib.lib wx_wxexpat.lib commctrl.lib winsock.lib wininet.lib` (since the library names in the wxWidgets workspace were changed by VS 2005).

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

## Remaining issues

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

- **Windows Mobile 5 issues.** It is not possible to get the HMENU for the command bar on Mobile 5, so the menubar functions need to be rewritten to get the individual menus without use of a menubar handle. Also the new Mobile 5 convention of using only two menus (and no bitmap buttons) needs to be considered.
- **Sizer speed.** Particularly for dialogs containing notebooks, layout seems slow. Some analysis is required.
- **Notification boxes.** The balloon-like notification messages, and their icons, should be implemented. This will be quite straightforward.
- **SIP size.** We need to be able to get the area taken up by the SIP (input panel), and the remaining area, by calling SHSipInfo. We also may need to be able to show and hide the SIP programmatically, with SHSipPreference. See also the *Input Dialogs* topic in the *Programming Windows CE* guide for more on this, and how to have dialogs show the SIP automatically using the WC\_SIPREF control.
- **wxStaticBitmap.** The About box in the "Life!" demo shows a bitmap that is the correct size on the emulator, but too small on a VGA Pocket Loox device.
- **wxStaticLine.** Lines don't show up, and the documentation suggests that missing styles are implemented with WM\_PAINT.
- **HTML control.** PocketPC has its own HTML control which can be used for showing local pages or navigating the web. We should create a version of wxHtmlWindow that uses this control, or have a separately-named control (wxHtmlCtrl), with a syntax as close as possible to wxHtmlWindow.

- **Tooltip control.** PocketPC uses special TTBUTTON and TTSTATIC controls for adding tooltips, with the tooltip separated from the label with a double tilde. We need to support this using SetToolTip. (Unfortunately it does not seem possible to dynamically remove the tooltip, so an extra style may be required.)
- **Focus.** In the wxPropertySheetDialog demo on Smartphone, it's not possible to navigate between controls. The focus handling in wxWidgets needs investigation. See in particular src/common/containr.cpp, and note that the default OnActivate handler in src/msw/toplevel.cpp sets the focus to the first child of the dialog.
- **OK button.** We should allow the OK button on a dialog to be optional, perhaps by using wxCLOSE\_BOX to indicate when the OK button should be displayed.
- **Dynamic adaptation.** We should probably be using run-time tests more than preprocessor tests, so that the same WinCE application can run on different versions of the operating system.
- **Modeless dialogs.** When a modeless dialog is hidden with the OK button, it doesn't restore the frame's menubar. See for example the find dialog in the dialogs sample. However, the menubar is restored if pressing Cancel (the window is closed). This reflects the fact that modeless dialogs are not very useful on Windows CE; however, we could perhaps destroy/restore a modeless dialog's menubar on deactivation and activation.
- **Home screen plugins.** Figure out how to make home screen plugins for use with wxWidgets applications (see <http://www.codeproject.com/ce/CTodayWindow.asp> for inspiration). Although we can't use wxWidgets to create the plugin (too large), we could perhaps write a generic plugin that takes registry information from a given application, with options to display information in a particular way using icons and text from a specified location.
- **Further abstraction.** We should be able to abstract away more of the differences between desktop and mobile applications, in particular for sizer layout.
- **Dialog captions.** The blue, bold captions on dialogs - with optional help button - should be catered for, either by hard-wiring the capability into all dialogs and panels, or by providing a standard component and sizer.

## wxMac port

wxMac is a port of wxWidgets for the Macintosh OS platform. Currently MacOS 8.6 or higher, MacOS 9.0 or higher and MacOS X 10.0 or higher are supported, although most development effort goes into MacOS X support. wxMac can be compiled both using Apple's developer tools and MetroWerks CodeWarrior in different versions. Support for MacOS 8.X and MacOS 9.X is only available through CodeWarrior. wxMac uses the Carbon API (and optionally the Classic API under MacOS 8.X). You will need wxWidgets version 2.3.3 or higher for a stable version of wxMac.

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

## **wxPalmOS port**

wxPalmOS is a port of wxWidgets for the Palm OS 6 (Cobalt). It ensures native look and feel for Palm devices when using wxWidgets version 2.5.4 or higher. wxPalmOS can be compiled with freely distributable Palm OS Developer Studio.

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

## **wxOS2 port**

wxOS2 is a port of wxWidgets for the IBM OS/2 platform. It is currently under construction.

## **wxMGL port**

wxMGL is a port of wxWidgets using the MGL library available from SciTech as the underlying graphics backend. wxMGL draws its widgets using the wxUniversal widget set which is now part of wxWidgets. MGL itself runs on a variety of platforms including DOS, Linux hardware (similar to the Linux framebuffer) and various graphics systems such as Win32, X11 and OS/2. Note that currently MGL for Linux runs only on x86-based systems.

You will need wxWidgets 2.3.3 or higher and MGL 5.0 or higher. The latter is available from

[http://www.scitechsoft.com/products/product\\_download.html](http://www.scitechsoft.com/products/product_download.html)  
([http://www.scitechsoft.com/products/product\\_download.html](http://www.scitechsoft.com/products/product_download.html))

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

```
configure --with-mgl --with-universal
```

Under DOS, wxMGL uses a dmake based make system.

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

## **wxX11 port**

wxX11 is a port of wxWidgets using X11 (The X Window System) as the underlying graphics backend. wxX11 draws its widgets using the wxUniversal widget set which is now part of wxWidgets. wxX11 is well-suited for a number of special applications such as those running on systems with few resources (PDAs) or for applications which need to use a special themed look. You will need wxWidgets 2.3.2 or higher.

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

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

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



# Index



::copystring, 1929  
::wxAboutBox, 1934  
::wxBeginBusyCursor, 1934  
::wxBell, 1935  
::wxClientDisplayRect, 1945  
::wxClipboardOpen, 1949  
::wxCloseClipboard, 1949  
::wxColourDisplay, 1945  
::wxConcatFiles, 1922  
::wxCopyFile, 1923  
::wxCreateDynamicObject, 1968  
::wxCreateFileTipProvider, 1935  
::wxDDECleanUp, 1954  
::wxDDEInitialize, 1954  
::wxDebugMsg, 1971  
::wxDirExists, 1922  
::wxDirSelector, 1935  
::wxDisplayDepth, 1945  
::wxDisplaySize, 1945  
::wxDisplaySizeMM, 1945  
::wxDos2UnixFilename, 1920  
::wxDROP\_ICON, 1945  
::wxEmptyClipboard, 1949  
::wxEnableTopLevelWindows, 1954  
::wxEndBusyCursor, 1937  
::wxEntry, 1909  
::wxEntryCleanup, 1910  
::wxEntryStart, 1910  
::wxEnumClipboardFormats, 1949  
::wxError, 1971  
::wxExecute, 1913  
::wxExit, 1915  
::wxFatalError, 1972  
::wxFileExists, 1920  
::wxFileModificationTime, 1920  
::wxFileNameFromPath, 1920  
::wxFileSelector, 1936  
::wxFindFirstFile, 1920  
::wxFindMenuItemId, 1955  
::wxFindNextFile, 1921  
::wxFindWindowAtPoint, 1955  
::wxFindWindowAtPointer, 1956  
::wxFindWindowByLabel, 1955  
::wxFindWindowByName, 1955  
::wxGenericAboutBox, 1937  
::wxGetActiveWindow, 1956  
::wxGetApp, 1911  
::wxGetBatteryState, 1956  
::wxGetClipboardData, 1950  
::wxGetClipboardFormatName, 1950  
::wxGetColourFromUser, 1937  
::wxGetCwd, 1923

::wxGetDiskSpace, 1921  
::wxGetDisplayName, 1956  
::wxGetElapsedTime, 1977  
::wxGetEmailAddress, 1926  
::wxGetFileKind, 1921  
::wxGetFontFromUser, 1938  
::wxGetFreeMemory, 1926  
::wxGetFullHostName, 1926  
::wxGetHomeDir, 1926  
::wxGetHostName, 1927  
::wxGetKeyState, 1952  
::wxGetLocalTime, 1977  
::wxGetLocalTimeMillis, 1978  
::wxGetMousePosition, 1957  
::wxGetMouseState, 1957  
::wxGetMultipleChoice, 1940  
::wxGetMultipleChoices, 1938  
::wxGetNumberFromUser, 1939  
::wxGetOsDescription, 1927  
::wxGetOSDirectory, 1922  
::wxGetOsVersion, 1927  
::wxGetPasswordFromUser, 1939  
::wxGetPowerType, 1956  
::wxGetPrinterCommand, 1947  
::wxGetPrinterFile, 1947  
::wxGetPrinterMode, 1947  
::wxGetPrinterOptions, 1947  
::wxGetPrinterOrientation, 1947  
::wxGetPrinterPreviewCommand, 1947  
::wxGetPrinterScaling, 1948  
::wxGetPrinterTranslation, 1948  
::wxGetProcessId, 1916  
::wxGetResource, 1958  
::wxGetSingleChoice, 1940  
::wxGetSingleChoiceData, 1942  
::wxGetSingleChoiceIndex, 1941  
::wxGetStockLabel, 1958  
::wxGetTempFileName, 1923  
::wxGetTextFromUser, 1940  
::wxGetTopLevelParent, 1959  
::wxGetTranslation, 1930  
::wxGetUserHome, 1928  
::wxGetUserId, 1928  
::wxGetUserName, 1929  
::wxGetUTCTime, 1978  
::wxGetWorkingDirectory, 1923  
::wxHandleFatalExceptions, 1911  
::wxInitAllImageHandlers, 1911  
::wxInitialize, 1911  
::wxIsAbsolutePath, 1922  
::wxIsBusy, 1942  
::wxIsClipboardFormatAvailable, 1950  
::wxIsDebuggerRunning, 1983  
::wxIsEmpty, 1930  
::wxIsMainThread, 1918

- ::wxIsPlatform64Bit, 1928
  - ::wxIsPlatformLittleEndian, 1928
  - ::wxIsWild, 1923
  - ::wxKill, 1915
  - ::wxLaunchDefaultBrowser, 1959
  - ::wxLoadUserResource, 1959
  - ::wxLogDebug, 1973
  - ::wxLogError, 1972
  - ::wxLogFatalError, 1972
  - ::wxLogMessage, 1972
  - ::wxLogStatus, 1973
  - ::wxLogSysError, 1973
  - ::wxLogTrace, 1974
  - ::wxLogVerbose, 1973
  - ::wxLogWarning, 1972
  - ::wxMakeMetafilePlaceable, 1946
  - ::wxMatchWild, 1923
  - ::wxMessageBox, 1942
  - ::wxMicroSleep, 1978
  - ::wxMilliSleep, 1978
  - ::wxMkdir, 1924
  - ::wxMutexGuiEnter, 1919
  - ::wxMutexGuiLeave, 1919
  - ::wxNewId, 1953
  - ::wxNow, 1979
  - ::wxOnAssert, 1980
  - ::wxOpenClipboard, 1950
  - ::wxParseCommonDialogsFilter, 1924
  - ::wxPathOnly, 1922
  - ::wxPostDelete, 1960
  - ::wxPostEvent, 1960
  - ::wxRegisterClipboardFormat, 1950
  - ::wxRegisterId, 1954
  - ::wxRemoveFile, 1924
  - ::wxRenameFile, 1924
  - ::wxRmdir, 1924
  - ::wxSafeShowMessage, 1975
  - ::wxSafeYield, 1912
  - ::wxSetClipboardData, 1951
  - ::wxSetCursor, 1946
  - ::wxSetDisplayName, 1960
  - ::wxSetPrinterCommand, 1948
  - ::wxSetPrinterFile, 1948
  - ::wxSetPrinterMode, 1948
  - ::wxSetPrinterOptions, 1948
  - ::wxSetPrinterOrientation, 1948
  - ::wxSetPrinterPreviewCommand, 1948
  - ::wxSetPrinterScaling, 1949
  - ::wxSetPrinterTranslation, 1949
  - ::wxSetWorkingDirectory, 1924
  - ::wxShell, 1916
  - ::wxShowTip, 1943
  - ::wxShutdown, 1916
  - ::wxSleep, 1979
  - ::wxSnprintf, 1932
  - ::wxSplitPath, 1925
  - ::wxStartTimer, 1979
  - ::wxStrcmp, 1930
  - ::wxStricmp, 1931
  - ::wxStringEq, 1931
  - ::wxStringMatch, 1931
  - ::wxStringTokenize, 1931
  - ::wxStripMenuCodes, 1960
  - ::wxStrlen, 1931
  - ::wxSysErrorCode, 1975
  - ::wxSysErrorMsg, 1976
  - ::wxTrace, 1976
  - ::wxTraceLevel, 1977
  - ::wxTransferFileToStream, 1925
  - ::wxTransferStreamToFile, 1925
  - ::wxTrap, 1983
  - ::wxUninitialize, 1912
  - ::wxUnix2DosFilename, 1922
  - ::wxUsleep, 1979
  - ::wxVsnprintf, 1933
  - ::wxWakeUpIdle, 1912
  - ::wxWriteResource, 1962
  - ::wxYield, 1912
- —
- \_, 1933
- \_\_WXFUNCTION\_\_, 1963
- \_T, 1934
- ~ —
- ~wxAcceleratorTable, 24
  - ~wxAccessible, 29
  - ~wxAnimation, 40
  - ~wxApp, 46
  - ~wxArchiveOutputStream, 69
  - ~wxArray, 78
  - ~wxArrayString, 84
  - ~wxArtProvider, 91
  - ~wxAuiDockArt, 95
  - ~wxAuiManager, 101
  - ~wxAuiPanelInfo, 110
  - ~wxAutomationObject, 120
  - ~wxBitmap, 127
  - ~wxBitmapButton, 142
  - ~wxBitmapComboBox, 138
  - ~wxBitmapHandler, 146
  - ~wxBrush, 153
  - ~wxBufferedOutputStream, 162
  - ~wxBusyCursor, 163
  - ~wxBusyInfo, 164
  - ~wxButton, 166
  - ~wxCalendarCtrl, 171
  - ~wxCheckBox, 182
  - ~wxCheckListBox, 185
  - ~wxChoice, 188
  - ~wxClientData, 194
  - ~wxClientDataContainer, 195
  - ~wxClipboard, 197
  - ~wxCmdLineParser, 207
  - ~wxColourData, 218
  - ~wxColourDialog, 222
  - ~wxComboBox, 228
  - ~wxComboCtrl, 236
  - ~wxCommand, 249
  - ~wxCommandProcessor, 256
  - ~wxCondition, 261
  - ~wxConfigBase, 270

~wxContextHelp, 283  
~wxCountingOutputStream, 294  
~wxCriticalSection, 294  
~wxCriticalSectionLocker, 296  
~wxCSCConv, 297  
~wxCursor, 302  
~wxCustomDataObject, 303  
~wxDatagramSocket, 307  
~wxDataInputStream, 310  
~wxDataObject, 313  
~wxDataOutputStream, 338  
~wxDataViewColumn, 315  
~wxDataViewCtrl, 318  
~wxDataViewCustomRenderer, 333  
~wxDataViewListModel, 326  
~wxDataViewListModelNotifier, 323  
~wxDataViewModel, 325  
~wxDataViewSortedListModel, 329  
~wxDbConnectInf, 411  
~wxDbGridCollInfo, 452  
~wxDbTable, 417  
~wxDebugReport, 489  
~wxDebugReportPreview, 493  
~wxDialog, 499  
~wxDialUpManager, 507  
~wxDir, 511  
~wxDirDialog, 515  
~wxDisplay, 520  
~wxDocChildFrame, 526  
~wxDocManager, 528  
~wxDocMDIChildFrame, 536  
~wxDocMDIParentFrame, 538  
~wxDocParentFrame, 539  
~wxDocTemplate, 543  
~wxDocument, 547  
~wxDropSource, 560  
~wxDropTarget, 562  
~wxEvtHandler, 576  
~wxFFFile, 585  
~wxFFFileInputStream, 590  
~wxFFFileOutputStream, 590  
~wxFile, 593  
~wxFileDialog, 602  
~wxFileHistory, 606  
~wxFileInputStream, 609  
~wxFileOutputStream, 629  
~wxFileType, 641  
~wxFindReplaceDialog, 651  
~wxFont, 660  
~wxFontMapper, 675  
~wxFrame, 685  
~wxFTP, 696  
~wxGauge, 702  
~wxGenericDirCtrl, 711  
~wxGenericValidator, 715  
~wxGrid, 739  
~wxGridCellEditor, 778  
~wxGridTableBase, 793  
~wxHashTable, 808  
~wxHelpController, 811  
~wxHelpProvider, 818  
~wxHtmlListBox, 854  
~wxIcon, 898  
~wxIconBundle, 901  
~wxImage, 910  
~wxImageHandler, 930  
~wxInputStream, 942  
~wxJoystick, 948  
~wxLayoutAlgorithm, 963  
~wxList<T>, 968  
~wxListBox, 977  
~wxListCtrl, 983  
~wxLocale, 1012  
~wxLogChain, 1026  
~wxMask, 1037  
~wxMDIChildFrame, 1049  
~wxMDIClientWindow, 1051  
~wxMDIParentFrame, 1054  
~wxMemoryInputStream, 1073  
~wxMemoryOutputStream, 1074  
~wxMenu, 1076  
~wxMenuBar, 1089  
~wxMenuItem, 1101  
~wxMessageDialog, 1107  
~wxMetafile, 1108  
~wxMetafileDC, 1110  
~wxMimeTypesManager, 1112  
~wxMiniFrame, 1115  
~wxModule, 1117  
~wxMutex, 1133  
~wxMutexLocker, 1134  
~wxNotebook, 1138  
~wxObjArray, 79  
~wxOutputStream, 1156  
~wxOwnerDrawnComboBox, 1154  
~wxPageSetupDialog, 1159  
~wxPageSetupDialogData, 1160  
~wxPalette, 1168  
~wxPanel, 1171  
~wxPen, 1178  
~wxPreviewCanvas, 1197  
~wxPreviewControlBar, 1198  
~wxPreviewFrame, 1199  
~wxPrintData, 1201  
~wxPrintDialog, 1207  
~wxPrintDialogData, 1208  
~wxPrinter, 1222  
~wxPrintout, 1215  
~wxProcess, 1225  
~wxProgressDialog, 1232  
~wxRadioBox, 1243  
~wxRadioButton, 1251  
~wxRecursionGuard, 1259  
~wxRegEx, 1262  
~wxRegion, 1265  
~wxRendererNative, 1277  
~wxRichTextBuffer, 1299  
~wxRichTextCharacterStyleDefinition, 1316  
~wxRichTextCtrl, 1316  
~wxRichTextFormattingDialog, 1355  
~wxRichTextFormattingDialogFactory, 1357  
~wxRichTextListStyleDefinition, 1366  
~wxRichTextParagraphStyleDefinition, 1368  
~wxRichTextRange, 1374



---

~wxRichTextStyleComboCtrl, 1378  
 ~wxRichTextStyleDefinition, 1376  
 ~wxRichTextStyleListBox, 1379  
 ~wxRichTextStyleSheet, 1387  
 ~wxSashWindow, 1399  
 ~wxScopedPtr, 1405  
 ~wxScopedTiedPtr, 1407  
 ~wxScrollBar, 1412  
 ~wxScrolledWindow, 1417  
 ~wxSearchCtrl, 1434  
 ~wxSemaphore, 1428  
 ~wxSimpleHtmlListBox, 858  
 ~wxSingleInstanceChecker, 1440  
 ~wxSizer, 1445  
 ~wxSizerItem, 1459  
 ~wxSlider, 1466  
 ~wxSocketAddress, 1472  
 ~wxSocketBase, 1475  
 ~wxSocketClient, 1488  
 ~wxSocketServer, 1493  
 ~wxSortedArray, 79  
 ~wxSound, 1495  
 ~wxSpinButton, 1498  
 ~wxSplashScreen, 1505  
 ~wxSplitterWindow, 1511  
 ~wxStackWalker, 1522  
 ~wxStaticBox, 1531  
 ~wxStatusBar, 1538  
 ~wxStreamBase, 1545  
 ~wxStreamToTextRedirector, 1553  
 ~wxString, 1561  
 ~wxStringBuffer, 1580  
 ~wxStringBufferLength, 1581  
 ~wxTarOutputStream, 1605  
 ~wxTaskBarIcon, 1608  
 ~wxTempFile, 1617  
 ~wxTextCtrl, 1639  
 ~wxTextEntryDialog, 1657  
 ~wxTextFile, 1659  
 ~wxTextInputStream, 1664  
 ~wxTextOutputStream, 1666  
 ~wxThread, 1673  
 ~wxThreadHelper, 1679  
 ~wxTimer, 1681  
 ~wxToggleButton, 1694  
 ~wxToolBar, 1698  
 ~wxTreebook, 1723  
 ~wxTreeCtrl, 1732  
 ~wxTreeItemData, 1752  
 ~wxURL, 1764  
 ~wxValidator, 1768  
 ~wxVariant, 1772  
 ~wxView, 1781  
 ~wxWindow, 1799  
 ~wxWindowDisabler, 1857  
 ~wxWindowUpdateLocker, 1854  
 ~wxXmlDocument, 1868  
 ~wxXmlNode, 1872  
 ~wxXmlAttribute, 1877  
 ~wxXmlResource, 1880  
 ~wxXmlResourceHandler, 1884  
 ~wxZipOutputStream, 1898

## —A—

A more complex example, 2144  
 A selection of SQL commands, 2174  
 Abort, 696, 1236  
 Above, 939  
 Abs, 1034, 1685  
 Absolute, 939  
 Accept, 1493  
 AcceptWith, 1493  
 Access, 593  
 Accessors, 353, 1684  
 Activate, 333, 1049, 1781  
 ActivateNext, 1054  
 ActivatePrevious, 1054  
 ActivateView, 528  
 Add, 79, 85, 335, 344, 367, 772, 934, 1174, 1446, 1685  
 AddAll, 489  
 AddArc, 730  
 AddArcToPoint, 730  
 AddArtist, 18  
 AddBook, 840, 843  
 AddBookCtrl, 1234  
 AddButton, 1542  
 AddCatalog, 1013  
 AddCatalogLookupPathPrefix, 1013  
 AddCharacterStyle, 1387  
 AddCheckTool, 1700  
 AddChild, 1799, 1873  
 AddCircle, 730  
 AddCollInfo, 453  
 AddColour, 220  
 AddContext, 489  
 AddControl, 1698  
 AddCurrentContext, 489  
 AddCurrentDump, 490  
 AddCurveToPoint, 730, 731  
 AddData, 197  
 AddDependency, 1117  
 AddDeveloper, 19  
 AddDocument, 529  
 AddDocWriter, 19  
 AddDump, 490  
 AddEllipse, 731  
 AddEnvList, 1174  
 AddEventHandler, 1299  
 AddExceptionContext, 490  
 AddExceptionDump, 490  
 AddFallbacks, 1112  
 AddFile, 490, 599, 1072  
 AddFilesToMenu, 607  
 AddFileToHistory, 529, 606  
 AddFileWithMimeType, 1072  
 AddFilter, 864, 873  
 AddGrowableView, 653  
 AddGrowableView, 653  
 AddHandler, 128, 634, 910, 1299, 1880  
 AddHelp, 818  
 AddIcon, 902  
 AddImage, 1317  
 Adding items, 75

- 
- Adding new resource handlers, 2115
  - AddLanguage, 1013
  - AddLine, 1659
  - AddLineToPoint, 731
  - AddListStyle, 1387
  - AddModule, 884
  - AddNotifier, 326
  - AddOption, 208
  - AddPage, 106, 1138, 1723
  - AddPane, 101
  - AddParagraph, 1299, 1317
  - AddParagraphStyle, 1388
  - AddParam, 209
  - AddPath, 731
  - AddPendingEvent, 576
  - AddProperty, 1873
  - AddQuadCurveToPoint, 731
  - AddRadioTool, 1700
  - AddRectangle, 731
  - AddRoot, 1732
  - AddRoundedRectangle, 731
  - AddSeparator, 1698
  - AddSpacer, 1448
  - AddStretchSpacer, 1448
  - AddStyle, 1388, 1884
  - AddSubPage, 1723
  - AddSwitch, 208
  - AddTag, 859
  - AddTagHandler, 859
  - AddText, 490
  - AddTool, 1698, 1699
  - AddToolBarButtons, 845, 846, 851
  - AddTraceMask, 1021
  - AddTranslator, 19
  - AddView, 547
  - AddViewingColumn, 326
  - AddWindowStyles, 1884
  - AddWord, 860
  - AdjustPagebreak, 820
  - AdvanceSelection, 106, 1139
  - Advise, 278, 480, 1612
  - AfterFirst, 1562
  - AfterLast, 1562
  - age, 1281
  - Align, 1456
  - All date/time classes at a glance, 2053
  - Alloc, 79, 85, 303, 1561
  - AllocHenv, 411
  - Allow, 1147
  - Alpha, 216
  - AltDown, 785, 786, 788, 959, 1123
  - Animate sample, 2031
  - AnimateShow, 236
  - AnyAddress, 945, 947
  - Append, 138, 287, 968, 1076, 1077, 1089, 1562, 1772
  - AppendBitmapColumn, 318
  - AppendByte, 1069
  - AppendCheckItem, 1078
  - AppendCols, 739, 795
  - AppendColumn, 318
  - AppendData, 1069
  - AppendDateColumn, 319
  - AppendDir, 613
  - AppendItem, 1732
  - AppendProgressColumn, 319
  - AppendRadioItem, 1079
  - AppendRows, 739, 795
  - AppendSeparator, 1079
  - AppendSubMenu, 1079
  - AppendText, 1317, 1639
  - AppendTextColumn, 319
  - AppendToggleColumn, 319
  - AppendToPage, 873
  - Application binary compatibility, 2233
  - Application shutdown, 2041
  - ApplyAlignmentToSelection, 1317
  - ApplyBoldToSelection, 1317
  - ApplyItalicToSelection, 1317
  - ApplyStyle, 1317, 1355, 1379, 1385
  - ApplyStyleSheet, 1317
  - ApplyUnderlineToSelection, 1318
  - Archives on non-seekable streams, 2228
  - AreLongOptionsEnabled, 207
  - argc, 46
  - argv, 46
  - Arrange, 983
  - Arrangelcons, 1054
  - Art provider sample, 2031
  - Asls, 940
  - assign, 971
  - Assign, 613, 614, 1034
  - AssignButtonsImageList, 1732
  - AssignCwd, 614
  - AssignDbTable, 456
  - AssignDir, 614
  - AssignHomeDir, 614
  - AssignImageList, 984, 1139, 1724, 1733
  - AssignStateImageList, 1733
  - AssignTempFileName, 614
  - Associated non-class functions, 380
  - AssociateModel, 319
  - AssociateTemplate, 529
  - Astronomical/historical functions, 355
  - Attach, 585, 594
  - AttachUnknownControl, 1880
  - AutoSize, 739
  - AutoSizeColumn, 740
  - AutoSizeColumns, 740
  - AutoSizeRow, 740
  - AutoSizeRows, 740
- B—**
- back, 971
  - Background: The need for conversion, 2059
  - Background: The wxString class, 2060
  - Basic IO, 1474
  - Basic Regular Expressions, 2219
  - BatchingUndo, 1299, 1318
  - BeforeFirst, 1563
  - BeforeLast, 1563
  - begin, 801, 805, 971
  - BeginAlignment, 1299, 1318
-

BeginBatch, 740  
BeginBatchUndo, 1299, 1318  
BeginBold, 1300, 1318  
BeginCharacterStyle, 1300, 1318  
BeginContextHelp, 283  
BeginDrag, 554  
BeginEdit, 777  
BeginFind, 808  
BeginFont, 1300, 1318  
BeginFontSize, 1300, 1319  
BeginItalic, 1300, 1319  
BeginLeftIndent, 1300, 1319  
BeginLineSpacing, 1300, 1319  
BeginListStyle, 1301, 1319  
BeginNumberedBullet, 1301, 1319  
BeginParagraphSpacing, 1302, 1320  
BeginParagraphStyle, 1302, 1320  
BeginRightIndent, 1302, 1320  
BeginStandardBullet, 1302  
BeginStyle, 1302, 1321  
BeginSuppressUndo, 1302, 1321  
BeginSymbolBullet, 1302, 1321  
BeginTextColour, 1303, 1321  
BeginUnderline, 1303, 1321  
BeginURL, 1303, 1321  
Below, 940  
Best book, 2127  
BestSize, 111  
BigEndianOrdered, 310, 338  
Bitmap format handlers, 2120  
Blit, 457  
BlockToDeviceRect, 741  
Blue, 216  
Blur, 910  
BlurHorizontal, 911  
BlurVertical, 911  
border, 1519  
Border, 1456  
bottom, 965  
Bottom, 111  
BottomDockable, 111  
Bracket Expressions, 2212  
Break, 1079  
Broadcast, 261  
BuildDeleteStmt, 417  
BuildSelectStmt, 418  
BuildUnescapedURI, 1759  
BuildUpdateStmt, 419  
BuildURI, 1759  
BuildWhereClause, 419  
Button, 1123  
ButtonDClick, 1123  
ButtonDown, 954, 1123  
ButtonIsDown, 955  
ButtonUp, 955, 1123

---

## —C—

c\_str, 1563  
C++ header file generation, 2114  
CacheBestSize, 1799  
CalcBoundingBox, 459

CalcMin, 150, 772, 1448, 1459  
CalcScrolledPosition, 1417  
CalculateScaling, 1372  
CalcUnscrolledPosition, 1417  
Calendar calculations, 355  
Calendar sample, 2031  
CallMethod, 120  
CancelDialing, 508  
CanConvert, 570  
CanCopy, 228, 1321, 1639  
CanCut, 228, 1321, 1639  
CanDeleteSelection, 1322  
CanDragColMove, 741  
CanDragColSize, 741  
CanDragGridSize, 741  
CanDragRowSize, 741  
CanEnableCellControl, 741  
CanGetValueAs, 794  
CanHandle, 59, 644, 1351, 1884  
CanHandleGZip, 1900, 1902  
CanHaveAttributes, 742, 796  
CanLoad, 1351, 1391  
CanonicalizeName, 565  
CanonicalizePluginName, 565  
CanOpen, 637  
CanPaste, 228, 1322, 1640  
CanPasteFromClipboard, 1303  
CanRead, 837, 910, 942  
CanRedo, 229, 1322, 1640  
CanSave, 1351, 1391  
CanSelectForUpdate, 420  
CanSend, 904  
CanSetTransparent, 1713  
CanSetValueAs, 794  
CanUndo, 229, 249, 256, 1322, 1640  
CanUpdate, 1754  
CanUpdateByROWID, 421  
CanVeto, 201  
Caption, 111  
CaptionVisible, 111  
CaptureMouse, 1799  
Cascade, 1055  
Case conversion, 1556  
Catalog, 383  
Cells and Containers, 2182  
CellToRect, 742  
Center, 111, 1456, 1800  
CenterIn, 1253  
CenterOnParent, 1800  
CenterOnScreen, 1800  
CenterPane, 111  
Centre, 111, 499, 685, 1456, 1800  
CentreIn, 1253  
CentreOnParent, 1801  
CentreOnScreen, 1801  
CentrePane, 111  
centreX, 965  
centreY, 965  
Chain, 1866  
ChangeMode, 520  
ChangePathTo, 634  
ChangeSelection, 1144, 1726

- ChangeValue, 1651
- char\_str, 1563
- Character access, 1554
- CharsetToEncoding, 676
- ChDir, 697
- Check, 185, 483, 1079, 1089, 1101, 1754
- CheckCommand, 696
- Checked, 254
- CheckForIntersection, 773
- Checklist sample, 2031
- CheckOSVersion, 1188
- CheckToolkitVersion, 1188
- Choosing a backend, 1060
- CLASSINFO, 1964
- CleanUpHandlers, 128, 911, 1303
- clear, 801, 805, 971
- Clear, 79, 85, 197, 287, 459, 614, 795, 808, 968, 1003, 1266, 1303, 1322, 1360, 1448, 1472, 1563, 1640, 1662, 1772, 1784
- ClearAll, 984
- ClearBackground, 1801
- ClearColumnImage, 1009
- ClearColumns, 320
- ClearCommands, 256
- Cleared, 323
- ClearGrid, 742
- ClearHandlers, 1880
- ClearList, 1772
- ClearListStyle, 1303, 1322
- ClearMemberVar, 421
- ClearMemberVars, 422
- ClearSel, 1466
- ClearSelection, 320, 742
- ClearStyleStack, 1304
- ClearTemporaryImageLocations, 1363
- ClearTicks, 1467
- ClearTools, 1700
- ClearTraceMasks, 1021
- Clients, 90
- ClientToScreen, 1801, 1802
- ClientToWindowSize, 1802
- Clip, 723
- Clone, 62, 97, 322, 573, 715, 769, 778, 792, 1304, 1348, 1669, 1768, 1891
- Close, 69, 197, 384, 547, 585, 594, 1110, 1157, 1273, 1476, 1606, 1659, 1781, 1802, 1898
- CloseButton, 112
- CloseContainer, 884
- CloseCursor, 422
- CloseDocuments, 529
- CloseEntry, 65, 69, 1604, 1606, 1895, 1898
- CloseOutput, 1226
- CloseSubpath, 731
- Closing top-level windows in wxWinCE, 2237
- Closing windows in wxWinCE, 2239
- cMB2WC, 1040
- cMB2WX, 1041
- CmdDown, 959, 1124
- Cmp, 1564
- CmpNoCase, 1564
- Collapse, 213, 1733
- CollapseAll, 1733
- CollapseAllChildren, 1733
- CollapseAndReset, 1733
- CollapseNode, 1724
- CollapsePath, 712
- CollapseTree, 711
- Combine, 1284
- CombineWithParagraphStyle, 1366
- Command, 285, 1323
- Commit, 1617, 1618
- CommitTrans, 384
- Common features, 2099
- CompareTo, 1564
- CompareVersion, 1880
- Comparison, 1555
- Comparison of wxString to other string classes, 2048
- Comparison operators, 1578
- Compatibility, 2056
- Compile, 1262
- Compilers, 1988
- ComputeFittingClientSize, 1448
- ComputeFittingWindowSize, 1449
- ComputeHistogram, 911
- ComputeScaleAndOrigin, 459
- Concat, 727
- Concatenation, 1555
- ConcatTransform, 726
- Config sample, 2031
- Connect, 577, 1489
- Constraint layout: more details, 2095
- Construction, 204
- Construction and destruction, 1474
- Constructor and destructor, 264, 1111
- Constructors, 1684
- Constructors and assignment operators, 1554
- Constructors and destructors, 74
- Constructors and initialization, 738
- Constructors, assignment operators and setters, 352
- Contains, 732, 1254, 1266, 1374, 1564
- Context menus in wxWinCE, 2239
- Control differences on wxWinCE, 2239
- ControlDown, 785, 786, 788, 959, 1124
- Controls sample, 2032
- Conversion to numbers, 1556
- Convert, 570, 1772
- ConvertAlphaToMask, 912
- ConvertDialogToPixels, 1803
- Converting buffers, 2061
- Converting strings, 2061
- ConvertPixelsToDialog, 1804
- ConvertStringToArgs, 206
- ConvertTenthsMMToPixels, 1380
- ConvertToBitmap, 912, 1266
- ConvertToGreyscale, 912
- ConvertToImage, 128
- ConvertToMono, 912
- ConvertYearToBC, 356
- Copy, 229, 236, 912, 1304, 1323, 1360, 1640
- CopyArchiveMetaData, 69, 1606, 1898
- CopyEntry, 70, 1606, 1898
- CopyFromBitmap, 898

CopyFromIcon, 128  
 copystring, 1929  
 CopyTo, 1074  
 CopyToClipboard, 1304  
 count, 801, 806  
 Count, 80, 85, 422  
 CountTokens, 1585  
 Create, 43, 106, 128, 138, 142, 146, 166, 170, 178,  
 182, 188, 212, 222, 223, 228, 236, 246, 270,  
 320, 341, 482, 499, 506, 517, 538, 539, 594,  
 631, 652, 671, 680, 685, 703, 711, 721, 777,  
 845, 846, 848, 854, 858, 892, 913, 935, 977,  
 984, 1037, 1049, 1055, 1062, 1115, 1139, 1154,  
 1168, 1171, 1234, 1243, 1251, 1273, 1323,  
 1355, 1382, 1385, 1396, 1412, 1418, 1431,  
 1439, 1467, 1495, 1498, 1502, 1511, 1529,  
 1531, 1534, 1536, 1538, 1588, 1615, 1640,  
 1659, 1673, 1679, 1694, 1724, 1759, 1785,  
 1792, 1859, 2174  
 CreateAbortWindow, 1212  
 CreateBitmap, 91  
 CreateBookCtrl, 1234  
 CreateBrush, 722, 734  
 CreateButtons, 1198, 1234, 1358  
 CreateButtonSizer, 499, 2105  
 CreateCanvas, 1200  
 CreateChildren, 1884  
 CreateChildrenPrivately, 1884  
 CreateClient, 1051  
 CreateContents, 848  
 CreateContext, 733  
 CreateContextFromNativeContext, 734  
 CreateContextFromNativeWindow, 734  
 CreateControlBar, 1199  
 CreateCurrentFont, 884  
 CreateDocument, 529, 543  
 CreateFont, 723, 734, 1284  
 CreateFontMapper, 57  
 CreateFromNative, 722  
 CreateFromNativeWindow, 722  
 CreateGrid, 742  
 CreateHelpDialog, 841  
 CreateHelpFrame, 841  
 CreateHTML, 1380  
 CreateIndex, 423, 849  
 CreateInstance, 121  
 CreateLinearGradientBrush, 722, 734  
 CreateLogTarget, 46, 57  
 CreateMatrix, 723, 735  
 CreateMessageOutput, 57  
 CreateObject, 190  
 CreatePage, 1358  
 CreatePages, 1358  
 CreatePath, 723, 735  
 CreatePen, 722, 734  
 CreatePopupMenu, 1608  
 CreateRadialGradientBrush, 722, 734  
 CreateRenderer, 57  
 CreateResFromNode, 1885  
 CreateResource, 1885  
 CreateSearch, 849  
 CreateSeparatedButtonSizer, 500

CreateStatusBar, 686  
 CreateStdDialogButtonSizer, 500  
 CreateStyle, 1391  
 CreateTable, 425  
 CreateTempFileName, 615  
 CreateToolBar, 686  
 CreateTraits, 46  
 CreateView, 385, 529, 543  
 Creating a backend, 1060  
 Creating an archive, 2223  
 CrossHair, 459  
 Custom event summary, 2085  
 Customization, 204, 1020  
 Cut, 229, 236, 1323, 1640  
 cWC2MB, 1041  
 cWC2WX, 1041  
 cWX2MB, 1041  
 cWX2WC, 1042

## —D—

Data transfer, 2177  
 Database sample, 2032  
 Date arithmetics, 354, 2054  
 Date comparison, 353  
 Day, 344, 1685  
 Daylight saving time (DST), 2056  
 Days, 345, 1685  
 DB\_STATUS, 426  
 Dbms, 386  
 DebugRpt sample, 2033  
 DecBy, 1440  
 DECLARE\_ABSTRACT\_CLASS, 1965  
 DECLARE\_APP, 1965  
 DECLARE\_CLASS, 1965  
 DECLARE\_DYNAMIC\_CLASS, 1966  
 DecRef, 769, 1778  
 DecTo, 1441  
 Default constructors, 78  
 DefaultPane, 112  
 Deflate, 1254  
 delete, 1151  
 Delete, 91, 288, 426, 808, 1080, 1323, 1674, 1734,  
 1773  
 Delete entries/groups, 267  
 DeleteAll, 270  
 DeleteAllItems, 984, 1734  
 DeleteAllPages, 1139, 1724  
 DeleteAllViews, 547  
 DeleteChildren, 1734  
 DeleteCols, 742, 795  
 DeleteColumn, 320, 984  
 DeleteContents, 808, 968  
 DeleteCursor, 426  
 DeleteEntry, 270  
 DeleteGroup, 270  
 DeleteItem, 985  
 DeleteKey, 1273  
 DeleteMatching, 427  
 DeleteNode, 968  
 DeleteObject, 969  
 DeletePage, 107, 1139, 1724

- DeleteProperty, 1873
- DeleteRangeWithUndo, 1304
- DeleteRows, 743, 795
- DeleteSelectedContent, 1323
- DeleteSelection, 1323
- DeleteSelf, 1273
- DeleteStyles, 1388
- DeleteTemporaryImages, 1363, 1364
- DeleteTool, 1700
- DeleteToolByPos, 1701
- DeleteValue, 1273
- DeleteWhere, 427
- DeleteWindows, 1459
- Deselect, 977
- DeselectAll, 1785
- Destroy, 778, 913, 1080, 1476, 1804
- DestroyChildren, 1805
- DestroyClippingRegion, 459
- DestroyOnClose, 112
- Detach, 80, 565, 585, 594, 1226, 1449
- DetachOldLog, 1026
- DetachPane, 101
- DetachRoot, 1868
- DetachSizer, 1459
- DetachStream, 693
- Deviations from the RFC, 1758
- DeviceToLogicalX, 459
- DeviceToLogicalXRel, 459
- DeviceToLogicalY, 460
- DeviceToLogicalYRel, 460
- Dial, 507
- Dialogs in wxWinCE, 2237
- Dialogs sample, 2033
- Dialup sample, 2033
- Difference between wxDateSpan and wxTimeSpan, 2054
- Different Flavors of REs, 2210
- Direction, 112
- DirExists, 615
- DirName, 616
- Disable, 1805
- DisableAutoCheckOnlineStatus, 509
- DisableCellEditControl, 743
- disabled, 139
- DisableDragColMove, 743
- DisableDragColSize, 743
- DisableDragGridSize, 743
- DisableDragRowSize, 743
- DisableLongOptions, 207
- DisassociateTemplate, 530
- Discard, 1476, 1617, 1618
- DiscardEdits, 1323, 1641
- Disconnect, 278, 480, 578, 1612
- Dismiss, 246
- DispAllErrors, 386
- Dispatch, 46
- Display, 841, 849
- Display format, 738
- DisplayBlock, 811
- DisplayContents, 812, 842, 849
- DisplayContextPopup, 812
- DisplayIndex, 842, 849
- DisplaySection, 812
- DisplayTextPopup, 812
- DispNextError, 387
- DnD sample, 2033
- Do, 250
- DoAddCustomContext, 491
- DoAddExceptionInfo, 491
- DoAddLoadedModules, 491
- DoAddSystemInfo, 491
- Dock, 112
- Dockable, 112
- DockFixed, 112
- DoCreateResource, 1885
- DoDefaultAction, 29
- DoDragDrop, 560
- DoDrawImage, 555
- DoGetBestSize, 1324, 1805
- DoLoadFile, 1351, 1391
- DoLog, 1023
- DoLogString, 1023
- DoneParser, 860
- Don't poll a wxThread, 1672
- DontCreateOnDemand, 270, 1023
- DoOK, 500
- DoParsing, 860
- DoPrepareDC, 1420
- DoQuantize, 1238
- DoSaveFile, 1351, 1364, 1391
- DoSetPopupControl, 237
- DoShowPopup, 237
- DoubleBorder, 1456
- DoubleHorzBorder, 1456
- DoUpdateWindowUI, 1805
- DragAcceptFiles, 1806
- Dragging, 1124
- Draw, 792, 820, 935
- DrawArc, 460
- DrawBackground, 95, 97
- DrawBitmap, 460, 723
- DrawBorder, 95
- DrawButton, 97
- DrawCaption, 95
- DrawCheckBox, 1277
- DrawCheckMark, 460
- DrawCircle, 460
- DrawComboBoxDropButton, 1277
- DrawDropArrow, 1278
- DrawEllipse, 461, 724
- DrawEllipticArc, 461
- DrawGripper, 95
- DrawHeaderButton, 1278
- DrawIcon, 461, 724
- DrawInvisible, 821
- DrawItemSelectionRect, 1278
- DrawLabel, 461, 462
- DrawLine, 462
- DrawLines, 462, 724
- DrawPaneButton, 95
- DrawPath, 724
- DrawPoint, 463
- DrawPolygon, 462
- DrawPolyPolygon, 463

DrawPushButton, 1278  
DrawRectangle, 463, 724  
DrawRotatedText, 463  
DrawRoundedRectangle, 464, 724  
DrawSash, 95  
DrawSpline, 464  
DrawSplitterBorder, 1279  
DrawSplitterSash, 1279  
DrawTab, 97  
DrawText, 464, 724  
DrawTreeItemButton, 1279  
DrawXXX, 496  
DropIndex, 428  
DropTable, 429  
DropView, 388  
Dump, 483, 1148, 1304  
Dynamic sample, 2034

## —E—

Edges and relationships, 938  
EditLabel, 985, 1734  
Embedded Visual C++ Issues, 2241  
empty, 801, 806, 971  
Empty, 80, 85, 1564  
EmulateKeyPress, 1641  
Enable, 1081, 1090, 1101, 1243, 1712, 1754, 1806  
EnableAutoCheckOnlineStatus, 509  
EnableCellEditControl, 743  
EnableCloseButton, 1714  
EnableDragColMove, 744  
EnableDragColSize, 744  
EnableDragGridSize, 744  
EnableDragRowSize, 744  
EnableEditing, 744  
EnableEffects, 668  
EnableGridLines, 744  
EnableHelp, 1160, 1208  
EnableHolidayDisplay, 171  
EnableLongOptions, 207  
EnableMargins, 1160  
EnableMonthChange, 171  
EnableOrientation, 1160  
EnablePageNumbers, 1208  
EnablePaper, 1160  
EnablePopupAnimation, 237  
EnablePrinter, 1160  
EnablePrintToFile, 1208  
EnableScrolling, 1418  
EnableSelection, 1208  
EnableTool, 1701  
EnableTop, 1090  
EnableYearChange, 171  
end, 802, 806, 971  
EndAlignment, 1304, 1324  
EndAllStyles, 1305, 1324  
EndBatch, 744  
EndBatchUndo, 1305, 1324  
EndBold, 1305, 1324  
EndCharacterStyle, 1305, 1324  
EndContextHelp, 284  
EndDoc, 465  
EndDrag, 555  
EndDrawingOnTop, 1408  
EndEdit, 778  
EndEditLabel, 1734  
EndFont, 1305, 1324  
EndFontSize, 1305, 1324  
EndItalic, 1305, 1325  
EndLeftIndent, 1305, 1325  
EndLineSpacing, 1306, 1325  
EndListStyle, 1306, 1325  
EndModal, 500  
EndNumberedBullet, 1306, 1325  
EndPage, 465  
EndParagraphSpacing, 1306, 1325  
EndParagraphStyle, 1306, 1325  
EndRightIndent, 1306, 1325  
EndStandardBullet, 1307  
EndStyle, 1306, 1326  
EndSuppressUndo, 1306, 1326  
EndsWith, 1572  
EndSymbolBullet, 1307, 1326  
EndTextColour, 1307, 1326  
EndUnderline, 1307, 1326  
EndURL, 1307, 1326  
EnsureFileAccessible, 1175  
EnsureVisible, 985, 1735  
Enter, 295  
Entering, 1124  
Entry, 1674, 1680  
Enumerated types, 376  
EnumerateEncodings, 673  
EnumerateFacenames, 672  
Enumeration, 265  
Eof, 585, 594, 942, 1660  
Eq, 1779  
erase, 802, 806, 971  
Erase, 969  
Error, 1476  
Escapes, 2214  
EscapeSqlChars, 388  
EstimateTotalHeight, 1792  
Event macros summary, 2083  
Event sample, 2034  
Events generated by the user vs programmatically generated events, 2081  
EVT\_COMMAND(id, event, func), 2086  
EVT\_COMMAND\_RANGE(id1, id2, event, func), 2087  
EVT\_CUSTOM(event, id, func), 2086  
EVT\_CUSTOM\_RANGE(event, id1, id2, func), 2086  
EVT\_NOTIFY(event, id, func), 2087  
EVT\_NOTIFY\_RANGE(event, id1, id2, func), 2087  
Example, 2045, 2124  
Example 1: subwindow layout, 2096  
Example 2: panel item layout, 2097  
Examples, 2178  
Except(ions) sample, 2034  
Exec sample, 2034  
ExecSql, 389

Execute, 278, 480, 1612  
Exists, 271, 511, 595, 1228, 1273, 1274, 1659  
Exit, 1674  
ExitMainLoop, 47  
Expand, 213, 1457, 1735  
ExpandAll, 1735  
ExpandAllChildren, 1735  
ExpandCommand, 643  
ExpandNode, 1724  
ExpandPath, 712  
explicit wxScopedPtr, 1405  
ExportXML, 1391  
ExtendSelection, 1326  
Extracting an archive, 2224

## —F—

fd, 594  
Feature tests, 1989  
File name components, 612  
File name construction, 611  
File name format, 611  
File tests, 612  
FileExists, 616, 699  
FileHistoryAddFilesToMenu, 530  
FileHistoryLoad, 530  
FileHistoryRemoveMenu, 530  
FileHistorySave, 530  
FileHistoryUseMenu, 530  
FileName, 616  
FileNameToURL, 635  
FillBuffer, 1552  
FillHandlersTable, 871  
FillPath, 724  
FilterEvent, 47  
find, 802, 806  
Find, 59, 221, 644, 821, 969, 1565  
FindAbsoluteValidPath, 1175  
FindByld, 1701  
FindCharacterStyle, 1388  
FindClass, 190  
FindControl, 1701  
FindFileInPath, 635  
FindFirst, 511, 636, 638  
FindFirstUnusedColour, 913  
FindFocus, 1806  
FindHandler, 129, 914, 1307  
FindHandlerFilenameOrType, 1307  
FindHandlerMime, 914  
FindItem, 773, 985, 1081, 1091  
FindItemAtPoint, 773  
FindItemAtPosition, 773  
FindItemByPosition, 1082  
FindItemWithData, 773  
FindLanguageInfo, 1014  
FindLevelForIndent, 1366  
FindListStyle, 1388  
FindMenu, 1091  
FindMenuItem, 1091  
FindName, 221  
FindNext, 636, 639  
FindNextWordPosition, 1326

FindOrCreateBrush, 157  
FindOrCreateFont, 674  
FindOrCreatePen, 1183  
FindPageByld, 844  
FindPageByName, 844  
FindParagraphStyle, 1388  
FindString, 288, 1244  
FindStyle, 1388  
FindTemplateForPath, 531  
FindToolForPosition, 1702  
FindValidPath, 1175  
FindWindow, 1807  
FindWindowByld, 1807  
FindWindowByLabel, 1807  
FindWindowByName, 1808  
First, 1565  
Fit, 745, 1449, 1808  
FitInside, 1449, 1808  
FitThisSizeToPage, 1218  
FitThisSizeToPageMargins, 1218  
FitThisSizeToPaper, 1218  
FitToPage, 1860  
Fixed, 113, 1551  
FixedMinSize, 1457  
Float, 113  
Floatable, 113  
FloatingPosition, 113  
FloatingSize, 113  
FloodFill, 465  
Flush, 197, 271, 586, 595, 1023  
Flushable, 1551  
FlushActive, 1023  
FlushBuffer, 1552  
fn\_str, 1565  
focus, 140  
Focus, 1009  
Font sample, 2035  
ForceRefresh, 745  
Format, 370, 408, 1565, 1685  
FormatDate, 370  
FormatISODate, 370  
FormatISOTime, 370  
FormatTime, 370  
Formatting time spans, 1685  
FormatV, 1566  
Found, 209, 210  
fp, 585  
Free, 304  
FreeHenv, 411  
Freeze, 1327, 1808  
Freq, 1566  
From, 429  
From8BitData, 1566  
FromAscii, 1566  
FromInternal, 1374  
FromTimezone, 374  
FromUTF8, 1567  
FromWChar, 1042  
front, 972



## —G—

- General issues for wxWinCE programming, 2235
- Generic archive programming, 2227
- get, 1403, 1406
- Get, 271, 676, 727, 808, 818, 1188, 1279, 1523, 1880
- Get3StateValue, 182
- GetAbort, 1212
- GetAcceleratorTable, 1809
- GetAccessible, 1809
- GetAccessTime, 1600
- GetActive, 34
- GetActiveChild, 1056
- GetActiveTarget, 1022
- GetActualColor, 884
- GetAddress, 568, 1519
- GetAdjustedBestSize, 1809
- GetAdjustedSize, 247
- GetAffirmativeId, 501
- GetAlign, 884, 1003
- GetAlignHor, 826
- GetAlignment, 771, 1239, 1284, 1396, 1620
- GetAlignVer, 826
- GetAllEncodingNames, 676
- GetAllEquivalents, 571
- GetAllFiles, 511
- GetAllFormats, 313
- GetAllowSymbols, 669
- GetAllPages, 1209
- GetAllPanels, 101
- GetAllParams, 866
- GetAlpha, 915
- GetAltForEncoding, 676
- GetAmPmStrings, 356
- GetAnchor, 637, 693
- GetAnimation, 44
- GetAppendBuf, 1068
- GetApplyOnSelection, 1380
- GetAppName, 47, 271
- GetArch, 1189
- GetArchitecture, 1189
- GetArchName, 1189
- GetArrayString, 1773
- GetArtProvider, 101, 107
- GetAscending, 329
- GetAsDOS, 366
- GetAsString, 216
- GetAttr, 173, 796
- GetAttributes, 1355
- GetAttrProvider, 796
- GetAuthStr, 412
- GetBackground, 465
- GetBackgroundColour, 176, 771, 826, 1003, 1007, 1101, 1285, 1620, 1809
- GetBackgroundMode, 465
- GetBackgroundStyle, 1809
- GetBaseClassName1, 191
- GetBaseClassName2, 191
- GetBaseld, 607
- GetBaseStyle, 1376
- GetBasicStyle, 1308, 1327
- GetBatchCount, 745
- GetBatchedCommand, 1308
- GetBeginDST, 356
- GetBeginPos, 866
- GetBestSize, 792, 1063, 1810
- GetBestTabCtrlSize, 98
- GetBezelFace, 703
- GetBin, 1201
- GetBitmap, 92, 145, 315, 936, 1101, 1529, 1860, 1865, 1885
- GetBitmapDisabled, 142, 237
- GetBitmapFocus, 142
- GetBitmapHover, 142, 238
- GetBitmapLabel, 143
- GetBitmapNormal, 238
- GetBitmapPressed, 238
- GetBitmapSelected, 143
- GetBitmapSize, 138
- GetBlinkTime, 178
- GetBlockingFactor, 1606
- GetBlue, 915
- GetBookCtrl, 1234
- GetBookRecArray, 844
- GetBool, 1773, 1885
- GetBorder, 176, 1459
- GetBorderColour, 176
- GetBottom, 1254
- GetBottomLeft, 1255
- GetBottomRight, 1255
- GetBottomRightCoords, 786
- GetBottomRow, 787
- GetBoundingRect, 1735
- GetBox, 732, 1266
- GetBrush, 466
- GetBuffer, 1327
- GetBufferEnd, 1550
- GetBufferPos, 1551
- GetBufferStart, 1550
- GetBufSize, 1067
- GetBulletFont, 1285, 1626
- GetBulletName, 1285, 1626
- GetBulletNumber, 1285, 1626
- GetBulletStyle, 1285, 1626
- GetBulletText, 1286, 1627
- GetButton, 1124
- GetButtonChange, 955
- GetButtonsImageList, 1736
- GetButtonSize, 238
- GetButtonState, 948, 955
- GetC, 942
- GetCacheFrom, 1001
- GetCacheTo, 1001
- GetCanonicalName, 1014
- GetCanvas, 1222
- GetCap, 1179
- GetCapture, 1810
- GetCapturedWindow, 1118
- GetCaret, 1810
- GetCaretPosition, 1327
- GetCaretPositionForIndex, 1327
- GetCatalog, 389
- GetCell, 882

---

GetCellAlignment, 745  
 GetCellBackgroundColour, 745  
 GetCellEditor, 746  
 GetCellFont, 746  
 GetCellRenderer, 746  
 GetCellSize, 773  
 GetCellTextColour, 746  
 GetCellValue, 746  
 GetCentury, 357, 364  
 GetChar, 1548, 1567, 1665, 1773  
 GetCharacter, 1348  
 GetCharacterStyle, 1388  
 GetCharacterStyleCount, 1389  
 GetCharacterStyleName, 1286, 1627  
 GetCharHeight, 466, 884, 1810  
 GetCharWidth, 466, 884, 1810  
 GetChecked, 1755  
 GetCheckPrevious, 484  
 GetChild, 30  
 GetChildCount, 30  
 GetChildren, 1450, 1811, 1873  
 GetChildrenCount, 1736  
 GetChooseFull, 218  
 GetChosenFont, 669  
 GetClassDefaultAttributes, 1811  
 GetClassInfo, 1149  
 GetClassName, 47, 191  
 GetClientArea, 521  
 GetClientAreaOrigin, 687  
 GetClientData, 195, 254, 288, 579, 1477, 1491  
 GetClientObject, 195, 254, 289, 579  
 GetClientSize, 1055, 1811  
 GetClientWindow, 1056  
 GetClippingBox, 466  
 GetCol, 705, 781, 785  
 GetColAt, 746  
 GetColDefs, 429  
 GetColGridLinePen, 751  
 GetColLabelAlignment, 747  
 GetColLabelSize, 747  
 GetColLabelValue, 747, 795  
 GetCollapsed, 214  
 GetCollate, 1201, 1209  
 GetColLeft, 747  
 GetColMinimalAcceptableWidth, 747  
 GetColMinimalWidth, 747  
 GetColor, 96  
 GetColour, 96, 153, 219, 224, 225, 669, 1179, 1202, 1594, 1885  
 GetColourData, 222  
 GetColoursCount, 1168  
 GetColPos, 747  
 GetColRight, 748  
 GetCols, 797  
 GetColSize, 748  
 GetColspan, 708  
 GetColType, 327  
 GetColumn, 320, 322, 986, 1002, 1004  
 GetColumnCount, 390, 986, 1244  
 GetColumns, 188, 390, 391  
 GetColumnWidth, 986  
 GetCombinedStyle, 1366  
 GetCombinedStyleLevel, 1366  
 GetCommand, 22  
 GetCommandProcessor, 547, 1308, 1327  
 GetCommands, 257  
 GetComment, 1891, 1896  
 GetCompressedFileName, 493  
 GetCompressedSize, 1891  
 GetConfigDir, 1523  
 GetConstraints, 1812  
 GetContainer, 885  
 GetContainingSizer, 1812  
 GetContainingWindow, 1450  
 GetContent, 1873  
 GetContentsArray, 844  
 GetContentType, 1237  
 GetContext, 719  
 GetControl, 247, 781  
 GetController, 845, 847  
 GetConv, 59  
 GetCount, 80, 86, 289, 521, 607, 809, 969, 1736, 1772  
 GetCountPerPage, 986  
 GetCountry, 357  
 GetCPUCount, 1674  
 GetCrc, 1892  
 GetCreateTime, 1600  
 GetCurFileSystem, 1885  
 GetCurrentDocument, 531  
 GetCurrentId, 1674  
 GetCurrentLine, 1660  
 GetCurrentMode, 521  
 GetCurrentMonth, 357  
 GetCurrentPage, 1140, 1222, 1860  
 GetCurrentPoint, 732  
 GetCurrentSelection, 188, 229  
 GetCurrentTip, 1691  
 GetCurrentView, 531  
 GetCurrentYear, 357  
 GetCursor, 430, 1430, 1812  
 GetCustomColour, 219  
 GetCustomPaintWidth, 238  
 GetCwd, 616  
 GetDashes, 1179  
 GetData, 197, 304, 392, 562, 652, 849, 915, 1002, 1004, 1067, 1135, 1567, 1582, 1773  
 GetDatabaseName, 393  
 GetDataDir, 1523  
 GetDataHere, 314, 336  
 GetDataLeft, 1552  
 GetDataLen, 1067  
 GetDataObject, 560  
 GetDataSize, 314, 336  
 GetDataSourceName, 393  
 GetDataViewColumn, 322  
 GetDate, 171, 340  
 GetDateOnly, 363  
 GetDateTime, 63, 1773  
 GetDay, 364  
 GetDayOfYear, 365  
 GetDays, 345, 1686  
 GetDb, 430  
 GetDC, 333, 572, 885, 1215

---

GetDebugMode, 484  
GetDefault, 1279  
GetDefaultAction, 30  
GetDefaultAttributes, 1812  
GetDefaultBorder, 1457  
GetDefaultCellAlignment, 748  
GetDefaultCellBackgroundColour, 748  
GetDefaultCellFont, 748  
GetDefaultCellTextColour, 748  
GetDefaultColLabelSize, 748  
GetDefaultColSize, 748  
GetDefaultDir, 412  
GetDefaultEditor, 749  
GetDefaultEditorForCell, 749  
GetDefaultEditorForType, 749  
GetDefaultEncoding, 661  
GetDefaultExtension, 543  
GetDefaultGridLinePen, 750  
GetDefaultInfo, 1161  
GetDefaultItem, 1714  
GetDefaultMinMargins, 1160  
GetDefaultPath, 712  
GetDefaultRenderer, 733, 749  
GetDefaultRendererForCell, 749  
GetDefaultRendererForType, 749  
GetDefaultRowLabelSize, 749  
GetDefaultRowSize, 749  
GetDefaultSize, 166, 1534  
GetDefaultStyle, 1308, 1641  
GetDefaultStyleEx, 1327  
GetDelay, 40  
GetDelayedLayoutThreshold, 1327  
GetDepth, 130, 521, 899, 1873  
GetDescent, 822  
GetDescription, 30, 412, 543, 642, 1377, 1389  
GetDesktopEnvironment, 57  
GetDevMajor, 1600  
GetDevMinor, 1600  
GetDialog, 649, 1355  
GetDialogAttributes, 1355  
GetDialogStyleDefinition, 1355  
GetDimension, 1885  
GetDirCount, 616  
GetDirection, 1863  
GetDirectory, 491, 544, 602  
GetDirList, 699  
GetDirs, 616  
GetDispatchId, 39  
GetDispatchPtr, 121  
GetDllExt, 524  
GetDockSizeConstraint, 102  
GetDocument, 526, 536, 1781  
GetDocumentManager, 539, 544, 548, 1781  
GetDocumentName, 544, 548  
GetDocuments, 531  
GetDocumentsDir, 1524  
GetDocumentTemplate, 547  
GetDocumentWindow, 548  
GetDomain, 1883  
GetDouble, 1773  
GetDragRect, 1394  
GetDragStatus, 1394  
GetDropTarget, 1813  
GetDsn, 412  
GetDuplex, 1202  
GetEdge, 1394  
GetEditControl, 986, 1736  
GetEditMenu, 257  
GetEditor, 771  
GetEffectiveMinSize, 1810  
GetEmptyCellSize, 774  
GetEnabled, 1755  
GetEnableEffects, 669  
GetEnableHelp, 1161  
GetEnableMargins, 1161  
GetEnableOrientation, 1161  
GetEnablePaper, 1161  
GetEnablePrinter, 1161  
GetEncoding, 676, 1351, 1869  
GetEncodingConverter, 885  
GetEncodingDescription, 677  
GetEncodingFromName, 677  
GetEncodingName, 677  
GetEncodings, 673  
GetEnd, 1374  
GetEndDST, 357  
GetEndianness, 1189  
GetEndiannessName, 1189  
GetEndPos, 707  
GetEndPos1, 866  
GetEndPos2, 866  
GetEntryType, 271  
GetEOL, 1660  
GetError, 1236, 1764  
GetErrorStream, 1226  
GetEscapeld, 501  
GetEvent, 852  
GetEventHandler, 1813  
GetEventObject, 574  
GetEventType, 574  
GetEvtHandlerEnabled, 579  
GetExcludes, 1669  
GetExecutablePath, 1524  
GetExitCode, 1230  
GetExitOnFrameDelete, 48  
GetExt, 617  
GetExtension, 147  
GetExtension, 931  
GetExtension, 931  
GetExtension, 1351  
GetExtension, 1351  
GetExtensions, 641  
GetExternalAttributes, 1892  
GetExtra, 1892  
GetExtraLen, 1892  
GetExtraLong, 254  
GetExtraStyle, 1813  
GetExtWildcard, 1308  
GetFaceName, 661  
GetFacenames, 673  
GetFamily, 661  
GetFeatures, 238  
GetFieldRect, 1538  
GetFieldsCount, 1539

---

GetFile, 491  
 GetFileEncoding, 1869  
 GetFileFilter, 544  
 GetFileHistory, 531  
 GetFilename, 548, 602, 1328  
 GetFileName, 1520  
 GetFileNames, 600, 602  
 GetFilePath, 712  
 GetFiles, 558  
 GetFilesCount, 491  
 GetFileSize, 699  
 GetFilesList, 699  
 GetFileSystem, 854  
 GetFileType, 412  
 GetFileTypeFromExtension, 1112  
 GetFileTypeFromMimeType, 1112  
 GetFilter, 712  
 GetFilterIndex, 603, 712  
 GetFilterListCtrl, 712  
 GetFindString, 649, 650  
 GetFirst, 60, 430, 511, 644, 969  
 GetFirstChild, 822, 1736  
 GetFirstEntry, 272  
 GetFirstGroup, 271  
 GetFirstKey, 1273  
 GetFirstLine, 1660  
 GetFirstSelected, 1009, 1785  
 GetFirstValue, 1274  
 GetFirstView, 548  
 GetFirstVisibleItem, 1737  
 GetFirstVisibleLine, 1792  
 GetFirstVisiblePosition, 1328  
 GetFlag, 1459  
 GetFlags, 22, 102, 168, 544, 648, 650, 1239, 1286,  
     1349, 1351, 1385, 1477, 1622, 1880, 1892  
 GetFlexibleDirection, 653  
 GetFocus, 30  
 GetFocusedItem, 1009  
 GetFont, 96, 176, 466, 682, 771, 1004, 1007,  
     1101, 1360, 1596, 1620, 1813, 1886  
 GetFontAttributes, 1286  
 GetFontBold, 885  
 GetFontData, 671  
 GetFontFace, 885  
 GetFontFaceName, 1286  
 GetFontFixed, 885  
 GetFontItalic, 885  
 GetFontName, 1588  
 GetFontSize, 886, 1286  
 GetFontSizeMapping, 1364  
 GetFontStyle, 1286  
 GetFontUnderlined, 886, 1286  
 GetFontWeight, 1287  
 GetFooterMargin, 1360  
 GetFooterText, 1360, 1368  
 GetForbiddenChars, 617  
 GetForegroundColor, 1813  
 GetFormat, 336, 617  
 GetFormatCount, 314  
 GetFormattingDialogFactory, 1356  
 GetFragment, 1759  
 GetFrame, 41, 1031, 1222, 1781  
 GetFrameCount, 40  
 GetFrameParameters, 813  
 GetFromClause, 431  
 GetFromPage, 1209  
 GetFromPoint, 521  
 GetFromUnicode, 1588  
 GetFromWindow, 521  
 GetFS, 860  
 GetFullName, 617  
 GetFullPath, 617  
 GetGeneric, 1279  
 GetGeometry, 522  
 GetGrandParent, 1814  
 GetGreen, 915  
 GetGridCursorCol, 750  
 GetGridCursorRow, 750  
 GetGridLineColour, 750  
 GetGroupId, 1601  
 GetGroupName, 1601  
 GetH, 1270  
 GetHandle, 1814  
 GetHandlerFlags, 1328  
 GetHandlers, 130, 916, 1308  
 GetHDBC, 393  
 GetHeader, 890  
 GetHeaderButtonHeight, 1279  
 GetHeaderColourBg, 172  
 GetHeaderColourFg, 171  
 GetHeaderFooterData, 1369, 1372  
 GetHeaderMargin, 1360  
 GetHeaderText, 1360, 1369  
 GetHeaderValue, 1016  
 GetHeight, 130, 822, 899, 917, 1254, 1270, 1441  
 GetHeightForPageHeight, 107  
 GetHelp, 818, 1101  
 GetHelpController, 815  
 GetHelpString, 1082, 1092  
 GetHelpText, 30, 1814  
 GetHelpTextAtPoint, 1814  
 GetHenv, 412  
 GetHENV, 393  
 GetHGap, 797  
 GetHi, 1034  
 GetHighlightColourBg, 172  
 GetHighlightColourFg, 172  
 GetHistoryFile, 607  
 GetHistoryFilesCount, 531  
 GetHolidayColourBg, 173  
 GetHolidayColourFg, 172  
 GetHomeDir, 617  
 GetHostType, 1759  
 GetHour, 364  
 GetHours, 1686  
 GetHoverColour, 892  
 GetHref, 852  
 GetHSTMT, 394  
 GetHtmlCell, 852  
 GetHumanReadableSize, 619  
 GetIcon, 92, 642, 902, 936, 1529, 1714, 1886  
 GetIcons, 1714  
 GetId, 306, 574, 822, 1004, 1102, 1675, 1752,  
     1815

---

GetID, 1886  
GetImage, 1002, 1004  
GetImageCount, 916, 931, 936  
GetImageExtWildcard, 914  
GetImageList, 986, 1140, 1356, 1737  
GetImageRect, 556  
GetInactiveBitmap, 44  
GetIncludes, 1669  
GetIndent, 827, 1737  
GetIndentSize, 98  
GetIndentUnits, 827  
GetIndex, 1001  
GetIndexArray, 844  
GetInitialFont, 669  
GetInnerSizer, 1234  
GetInputEncoding, 886  
GetInputStream, 700, 889, 1226, 1236, 1765  
GetInputStreamBuffer, 1073  
GetInsertionPoint, 229, 239, 1328, 1641  
GetInstallPrefix, 1524  
GetInstance, 48, 121  
GetInt, 254  
GetInternalFormat, 63  
GetInternalMargin, 1184  
GetInternalName, 60, 63, 1601, 1892, 1893  
GetInternalRepresentation, 873  
GetInterval, 1681, 1683  
GetIntPosition, 1551  
GetSPNames, 507  
GetItem, 987, 1002, 1450, 1750  
GetItemBackgroundColour, 987, 1737  
GetItemBitmap, 138  
GetItemCount, 987, 1785  
GetItemData, 987, 1737  
GetItemFont, 987, 1738  
GetItemFromPoint, 1245  
GetItemHelpText, 1244  
GetItemImage, 1738  
GetItemLabel, 1102  
GetItemLabelText, 1102  
GetItemParent, 1739  
GetItemPosition, 774, 988  
GetItemRect, 988  
GetItemSelectedImage, 1740  
GetItemSpacing, 988  
GetItemSpan, 774  
GetItemState, 988  
GetItemText, 989, 1738  
GetItemTextColour, 989, 1738  
GetItemToolTip, 1244  
GetJDN, 373  
GetJoin, 1179  
GetJoystick, 955  
GetJulianDayNumber, 373  
GetKeyboardShortcut, 30  
GetKeyCode, 22, 959, 1001, 1750  
GetKeyEvent, 1750  
GetKeyFields, 394  
GetKind, 586, 595, 1102  
GetLabel, 166, 286, 1002, 1082, 1092, 1102, 1245, 1536, 1750, 1815  
GetLabelBackgroundColour, 751  
GetLabelFont, 751  
GetLabelFromText, 1103  
GetLabelText, 286, 1083, 1102  
GetLabelTextColour, 751  
GetLabelTop, 1092  
GetLanguage, 1014  
GetLanguageInfo, 1015  
GetLanguageName, 1015  
GetLast, 431, 969  
GetLastAccess, 1551  
GetLastChild, 1738  
GetLastDelimiter, 1585  
GetLastDirectory, 531  
GetLastError, 1212, 1545  
GetLastLine, 1661  
GetLastMonthDay, 372  
GetLastPosition, 229, 239, 1328, 1642  
GetLastResult, 696  
GetLastVisibleLine, 1792  
GetLastWeekDay, 372  
GetLayoutDirection, 466  
GetLeft, 1254  
GetLeftCol, 787  
GetLeftIndent, 1287, 1620  
GetLeftLocation, 638  
GetLeftSubIndent, 1287, 1620  
GetLength, 1374, 1545  
GetLevel, 484, 1520, 1899  
GetLevelAttributes, 1366  
GetLevelCount, 1367  
GetLine, 1520, 1660  
GetLineCount, 1659, 1793  
GetLineLength, 1328, 1642  
GetLineSize, 1467  
GetLineSpacing, 1287, 1627  
GetLinesPerAction, 1125  
GetLineText, 1328, 1642  
GetLineType, 1661  
GetLink, 822, 886  
GetLinkClicked, 883  
GetLinkColor, 886  
GetLinkInfo, 882  
GetLinkName, 1602  
GetListStyle, 1389  
GetListStyleCount, 1389  
GetListStyleName, 1287, 1627  
GetLo, 1034  
GetLocal, 1477  
GetLocalDataDir, 1525  
GetLocale, 1015  
GetLocalExtra, 1893  
GetLocalExtraLen, 1893  
GetLocalizedResourcesDir, 1525  
GetLocation, 31, 694  
GetLoggingOff, 201  
GetLogicalFunction, 467  
GetLogicalPageMarginsRect, 1220  
GetLogicalPageRect, 1219  
GetLogicalPaperRect, 1219  
GetLogicalPoint, 1328  
GetLogicalPosition, 1125  
GetLogLevel, 1024

GetLong, 1773, 1886  
GetLongPath, 617  
GetManagedWindow, 102  
GetManager, 102  
GetManufacturerId, 948  
GetMapMode, 467  
GetMarginBottomRight, 1161  
GetMargins, 1703, 1786  
GetMarginTopLeft, 1161  
GetMarginWidth, 1103  
GetMask, 130, 1002, 1004  
GetMaskBlue, 917  
GetMaskGreen, 917  
GetMaskRed, 917  
GetMatch, 1263  
GetMatchCount, 1263  
GetMax, 1467, 1499, 1503  
GetMaxCommands, 257  
GetMaxDocsOpen, 531  
GetMaxFiles, 607  
GetMaximumSizeX, 1400  
GetMaximumSizeY, 1400  
GetMaxMBNulLen, 1042  
GetMaxPage, 1209, 1222  
GetMaxPointSize, 681  
GetMaxSize, 1815  
GetMBNulLen, 1042  
GetMenu, 1093, 1099, 1103, 1435  
GetMenuBar, 687  
GetMenuCount, 1093  
GetMenuId, 1099  
GetMenuItemCount, 1083  
GetMenuItemItems, 1083  
GetMenuItemLabel, 1093  
GetMenuItemLabelText, 1094  
GetMenus, 607  
GetMessage, 515, 603  
GetMethod, 1893  
GetMetric, 96, 1596  
GetMillisecond, 365  
GetMilliseconds, 1686  
GetMimeType, 641, 694, 931  
GetMimeTypeFromExt, 638  
GetMin, 1467, 1499, 1503  
GetMinimumPaneSize, 1511  
GetMinimumSizeX, 1400  
GetMinimumSizeY, 1400  
GetMinMarginBottomRight, 1162  
GetMinMarginTopLeft, 1162  
GetMinPage, 1209, 1222  
GetMinSize, 1450, 1459, 1815  
GetMinute, 364  
GetMinutes, 1686  
GetMJD, 373  
GetMode, 330, 905, 1602, 1667, 1756, 1893  
GetModel, 320, 322  
GetModelColumn, 315  
GetModes, 522  
GetModificationTime, 617, 694  
GetModifiedJulianDayNumber, 373  
GetModifiers, 959  
GetModule, 1520  
GetMonth, 364  
GetMonthName, 357  
GetMonths, 345  
GetMouseCursor, 823  
GetMovementThreshold, 948  
GetMultiLineTextExtent, 467  
GetName, 147  
GetName, 31, 63  
GetName, 250  
GetName, 512  
GetName, 512  
GetName, 522  
GetName, 522  
GetName, 567  
GetName, 567  
GetName, 618  
GetName, 618  
GetName, 867  
GetName, 867  
GetName, 930  
GetName, 930  
GetName, 1015  
GetName, 1015  
GetName, 1103  
GetName, 1103  
GetName, 1273  
GetName, 1273  
GetName, 1352  
GetName, 1352  
GetName, 1377  
GetName, 1377  
GetName, 1389  
GetName, 1389  
GetName, 1520  
GetName, 1520  
GetName, 1662  
GetName, 1662  
GetName, 1774  
GetName, 1774  
GetName, 1815  
GetName, 1815  
GetName, 1874  
GetName, 1874  
GetName, 1878  
GetName, 1878  
GetName, 1886  
GetNativeContext, 725  
GetNativeFontInfoDesc, 661  
GetNativeFontInfoUserDesc, 662  
GetNativeMatrix, 728  
GetNativePath, 732  
GetNewCursor, 431  
GetNewStyleSheet, 1349  
GetNext, 60, 394, 432, 512, 644, 823, 1135, 1865, 1874, 1878  
GetNextChild, 1739  
GetNextEntry, 65, 272, 1604, 1896  
GetNextError, 395  
GetNextGroup, 272  
GetNextHandler, 580  
GetNextItem, 989  
GetNextKey, 1274

---

GetNextLine, 1661  
 GetNextSelected, 1010, 1786  
 GetNextSibling, 1739, 1816  
 GetNextStyle, 1368  
 GetNextToken, 1585  
 GetNextValue, 1274  
 GetNextVisible, 1739  
 GetNextWeekDay, 371  
 GetNoCopies, 1202, 1209  
 GetNodeContent, 1392, 1874, 1886  
 GetNonFlexibleGrowMode, 653  
 GetNormalColour, 893  
 GetNormalTextFontName, 1588  
 GetNotebook, 1146  
 GetNumberAxes, 949  
 GetNumberButtons, 949  
 GetNumberCols, 751, 793  
 GetNumberJoysticks, 949  
 GetNumberOfCols, 327  
 GetNumberOfColumns, 320, 432  
 GetNumberOfDays, 358  
 GetNumberOfEntries, 272  
 GetNumberOfFiles, 558  
 GetNumberOfGroups, 272  
 GetNumberOfLines, 1329, 1642  
 GetNumberOfRows, 327  
 GetNumberRows, 751, 793  
 GetObject, 121  
 GetOffset, 63, 1520  
 GetOldItem, 1750  
 GetOldLog, 1026  
 GetOldSelection, 1145, 1728  
 GetOldStyleSheet, 1349  
 GetOpenCommand, 642  
 GetOpenedAnchor, 874  
 GetOpenedPage, 874  
 GetOpenedPageTitle, 874  
 GetOperatingSystemFamilyName, 1190  
 GetOperatingSystemId, 1190  
 GetOperatingSystemIdName, 1190  
 GetOption, 919, 1593  
 GetOptionInt, 919, 1593  
 GetOrCreateCellAttr, 752  
 GetOrderByClause, 432  
 GetOrFindMaskColour, 917  
 GetOrientation, 150, 1202, 1240, 1396, 1426, 1427  
 GetOrigin, 816  
 GetOSMajorVersion, 1189  
 GetOSMinorVersion, 1190  
 GetOutlineLevel, 1287, 1627  
 GetOutputEncoding, 886  
 GetOutputStream, 700, 1226  
 GetOutputStreamBuffer, 1074  
 GetOwner, 315, 323, 330  
 GetPage, 107, 1140, 1863  
 GetPageAreaSizer, 1860  
 GetPageBitmap, 107  
 GetPageCount, 107, 1140  
 GetPageId, 1358  
 GetPageIdCount, 1358  
 GetPageImage, 1140, 1358, 1725  
 GetPageIndex, 107  
 GetPageInfo, 1216, 1372  
 GetPageParent, 1725  
 GetPageSetupData, 834, 1159, 1369  
 GetPageSize, 1412, 1467, 1861  
 GetPageSizeMM, 1216  
 GetPageSizePixels, 1216  
 GetPageText, 107, 1140, 1725  
 GetPalette, 130, 917  
 GetPane, 102, 213  
 GetPaperId, 1162, 1202  
 GetPaperRectPixels, 1216  
 GetPaperSize, 1162  
 GetParagraphSpacingAfter, 1287, 1627  
 GetParagraphSpacingBefore, 1287, 1627  
 GetParagraphStyle, 1389  
 GetParagraphStyleCount, 1389  
 GetParagraphStyleName, 1287, 1627  
 GetParam, 210, 867, 1520  
 GetParamAsColour, 867  
 GetParamAsInt, 868  
 GetParamCount, 210, 1521  
 GetParamNode, 1392, 1886  
 GetParamValue, 1392, 1886  
 GetParent, 31, 823, 1816, 1874  
 GetParentWindow, 813, 834, 1369  
 GetPartialTextExtents, 467, 725  
 GetPassword, 395, 412, 1760  
 GetPath, 273, 515, 518, 568, 603, 618, 632, 633, 635, 712, 1760  
 GetPaths, 603  
 GetPathSeparator, 618  
 GetPathSeparators, 618  
 GetPathTerminators, 618  
 GetPathWithSep, 619  
 GetPeer, 1477  
 GetPen, 468  
 GetPhysicalPoint, 1329  
 GetPickerCtrlProportion, 1184  
 GetPid, 1229, 1230  
 GetPixel, 216, 468, 1168  
 GetPlatformEquivalents, 570  
 GetPlaybackrate, 1063  
 GetPluginsDir, 1525  
 GetPoint, 883, 1002, 1751  
 GetPointSize, 662  
 GetPollingMax, 949  
 GetPollingMin, 949  
 GetPopupControl, 240  
 GetPopupWindow, 240  
 GetPort, 1760  
 GetPortId, 1191  
 GetPortIdName, 1191  
 GetPortIdShortName, 1191  
 GetPos, 707  
 GetPosition, 178, 282, 558, 785, 789, 817, 949, 955, 960, 1124, 1125, 1129, 1254, 1349, 1426, 1427, 1450, 1459, 1504, 1585, 1816, 1887  
 GetPosX, 823  
 GetPosY, 823  
 GetPOVCTSPosition, 950  
 GetPOVPosition, 950

---

GetPPI, 468  
GetPPIPrinter, 1217  
GetPPIScreen, 1217  
GetPrecision, 790  
GetPreferredFormat, 314  
GetPrev, 432, 1865  
GetPreviewRect, 1369  
GetPrevious, 1135  
GetPreviousHandler, 580  
GetPrevLine, 1661  
GetPrevSibling, 1739, 1817  
GetPrevVisible, 1740  
GetPrevWeekDay, 371  
GetPrintableName, 548  
GetPrintCommand, 642  
GetPrintData, 834, 1162, 1209, 1369  
GetPrintDC, 1207  
GetPrintDialogData, 1207, 1213  
GetPrinterName, 1202  
GetPrintout, 1223  
GetPrintoutForPrinting, 1223  
GetPrintPreview, 1198  
GetPrintToFile, 1209  
GetPriority, 1675  
GetProduct, 860  
GetProductId, 949  
GetProductName, 949  
GetProgramHandle, 524, 566  
GetProperties, 1875  
GetProperty, 122  
GetProportion, 1460  
GetPropVal, 1874  
GetProtocol, 60, 638, 645, 1764  
GetProtocols, 60, 645  
GetQuality, 1202  
GetQuery, 1760  
GetQueryTableName, 433  
GetQuickBestSize, 1740  
GetRange, 342, 703, 1329, 1349, 1412, 1643  
GetRataDie, 373  
GetRatio, 1460  
GetRawKeyCode, 960  
GetRawKeyFlags, 960  
GetReceivedFormat, 335  
GetRect, 168, 1270, 1460, 1817  
GetRed, 917  
GetRedoAccelerator, 257  
GetRedoMenuLabel, 257  
GetRefCount, 1152  
GetRefData, 1149  
GetRelatedFrame, 874  
GetRenderer, 316, 729, 771, 1308  
GetRepetitionCounting, 1024  
GetReplaceString, 649, 650  
GetReportName, 492  
GetRequestedLength, 1240  
GetResolution, 1195  
GetResourcesDir, 1525  
GetResponse, 889  
GetRestartNumbering, 1386  
GetReturnCode, 501  
GetRGB, 1169  
GetRichTextBuffer, 1372  
GetRichTextCtrl, 1378, 1380, 1382, 1386  
GetRight, 1255  
GetRightCol, 787  
GetRightIndent, 1288, 1620  
GetRightLocation, 638  
GetRole, 31  
GetRoot, 1869  
GetRootId, 713  
GetRootItem, 1740  
GetRow, 322, 705, 782, 785  
GetRowCount, 1140, 1245  
GetRowGridLinePen, 750  
GetRowLabelAlignment, 752  
GetRowLabelSize, 752  
GetRowLabelValue, 752, 795  
GetRowMinimalAcceptableHeight, 752  
GetRowMinimalHeight, 752  
GetRowNum, 433  
GetRowOrCol, 789  
GetRows, 797  
GetRowSize, 752  
GetRowspan, 708  
GetRudderMax, 950  
GetRudderMin, 950  
GetRudderPosition, 950  
GetSashGravity, 1511  
GetSashPosition, 1507, 1511  
GetSashVisible, 1399  
GetScheme, 1761  
GetScreenPosition, 1817  
GetScreenRect, 1818  
GetScreenType, 1598  
GetScrollLineX, 752  
GetScrollLineY, 753  
GetScrollPixelsPerUnit, 1418  
GetScrollPos, 1818  
GetScrollRange, 1818  
GetScrollThumb, 1818  
GetSecond, 365  
GetSeconds, 1686  
GetSelectedCells, 753  
GetSelectedCols, 753  
GetSelectedCount, 1786  
GetSelectedFont, 680  
GetSelectedItemCount, 990  
GetSelectedRows, 753  
GetSelectedStyle, 1386  
GetSelectedStyleDefinition, 1386  
GetSelectedTextBgColour, 854  
GetSelectedTextColour, 854  
GetSelection, 108, 230, 254, 289, 320, 1141, 1145, 1210, 1245, 1329, 1437, 1643, 1725, 1728, 1740, 1786  
GetSelectionBackground, 753, 1786  
GetSelectionBlockBottomRight, 754  
GetSelectionBlockTopLeft, 753  
GetSelectionClientData, 1437  
GetSelectionForeground, 754  
GetSelectionMode, 753  
GetSelectionRange, 1329  
GetSelections, 31, 321, 977, 1130, 1740



- GetSelEnd, 1468
- GetSelStart, 1468
- GetServer, 1761
- GetSetChecked, 1755
- GetSetEnabled, 1755
- GetSetShown, 1755
- GetSetText, 1755
- GetShadowWidth, 703
- GetSheetStyle, 1234
- GetShortPath, 619
- GetShowHelp, 669
- GetShown, 1755
- GetShowOnFirstPage, 1361
- GetSize, 41, 63, 178, 191, 294, 304, 334, 468, 619, 937, 1240, 1255, 1444, 1450, 1460, 1545, 1602, 1818, 1887
- GetSizeHint, 92
- GetSizeMM, 469
- GetSizer, 1460, 1819
- GetSkipped, 574
- GetSocket, 1491
- GetSocketEvent, 1491
- GetSortable, 316
- GetSource, 860
- GetSpacer, 1460
- GetSpan, 707
- GetSplashStyle, 1505
- GetSplashWindow, 1505
- GetSplitMode, 1512
- GetSplitterParams, 1280
- GetStandardPaths, 57
- GetStart, 1374
- GetState, 31, 1004, 1063
- GetStateImageList, 1741
- GetStaticBox, 1532
- GetStatusBar, 687
- GetStatusBarPane, 688
- GetStatusText, 1539
- GetStipple, 153, 1179
- GetStream, 484, 694
- GetStreamBuf, 485
- GetString, 254, 290, 780, 1015, 1246, 1583, 1585, 1774
- GetStrings, 290
- GetStringSelection, 290, 1246, 1329, 1437, 1643
- GetStringValue, 247
- GetStyle, 153, 662, 1180, 1308, 1309, 1329, 1330, 1356, 1377, 1380, 1392, 1644, 1669, 1887
- GetStyleChoice, 1382
- GetStyleDefinition, 1356
- GetStyleForRange, 1309, 1330
- GetStyleListBox, 1383
- GetStyleMergedWithBase, 1377
- GetStyleSheet, 1309, 1330, 1356, 1378, 1380, 1383, 1386
- GetStyleStackSize, 1309
- GetStyleType, 1380, 1383
- GetSubBitmap, 131
- GetSubImage, 918
- GetSubItemRect, 988
- GetSubMenu, 1104
- GetSupportedEncodingsCount, 677
- GetSupportedTags, 869
- GetSymbol, 524, 565, 1589
- GetSymbolAorW, 566
- GetSymbolChar, 1589
- GetSysName, 1016
- GetSystemEncoding, 1016
- GetSystemEncodingName, 1016
- GetSystemLanguage, 1017
- GetSystemMadeBy, 1894
- GetTabCtrlHeight, 108
- GetTable, 754
- GetTableCount, 395
- GetTableName, 433
- GetTablePath, 433
- GetTabs, 1288, 1621
- GetTabSize, 98
- GetTarget, 852
- GetTempDir, 619, 1364, 1526
- GetTemplates, 532
- GetTemporaryImageLocations, 1364
- GetText, 1002, 1005, 1103, 1361, 1392, 1654, 1756, 1887
- GetTextBackground, 469
- GetTextColour, 176, 771, 990, 1005, 1008, 1104, 1288, 1361, 1621
- GetTextCtrl, 240, 1185
- GetTextCtrlProportion, 1184
- GetTextEffectFlags, 1288, 1628
- GetTextEffects, 1288, 1628
- GetTextExtent, 469, 725, 1819
- GetTextForeground, 470
- GetTextIndent, 240
- GetTextLength, 1654
- GetTextRect, 240
- GetThemeBackgroundColour, 1141
- GetThread, 1680
- GetThumbLength, 1468
- GetThumbPosition, 1413
- GetThumbSize, 1413
- GetTickFreq, 1468
- GetTicks, 364
- GetTimeNow, 358
- GetTimeout, 1505
- GetTimes, 620
- GetTimestamp, 574, 1024
- Getting results, 205
- Getting started: a simple example, 2143
- GetTip, 1690, 1713
- GetTitle, 549, 1083, 1217, 1369, 1714
- GetTm, 363
- GetTmNow, 358
- GetToolBar, 501, 688, 1056
- GetToolBitmapSize, 1702
- GetToolClientData, 1703
- GetToolEnabled, 1703
- GetToolkitMajorVersion, 1191
- GetToolkitMinorVersion, 1192
- GetToolkitVersion, 57
- GetToolLongHelp, 1703
- GetToolPacking, 1704
- GetToolPos, 1704
- GetToolsCount, 1702

GetToolSeparation, 1704  
GetToolShortHelp, 1704  
GetToolSize, 1702  
GetToolState, 1705  
GetToolTip, 1820  
GetTop, 1255  
GetToPage, 1210  
GetTopItem, 990  
GetTopLeft, 1255  
GetTopLeftCoords, 787  
GetTopRight, 1255  
GetTopRow, 787  
GetTopWindow, 48  
GetTotalDays, 345  
GetTotalEntries, 1896  
GetTotalHeight, 833  
GetTotalSize, 512  
GetTraceMasks, 1021  
GetTraits, 48  
GetTransform, 726  
GetTreeCtrl, 713  
GetType, 147, 306, 931, 1352, 1774, 1779, 1875  
GetTypeFlag, 1602  
GetTypeNames, 793  
GetUid, 413  
GetUMax, 950  
GetUMin, 950  
GetUncombinedStyle, 1309, 1330  
GetUnderlined, 662  
GetUndoAccelerator, 257  
GetUndoMenuLabel, 257  
GetUnicodeKey, 960  
GetUpdateInterval, 1756  
GetUpdateRegion, 1820  
GetUPosition, 950  
GetURL, 893, 894, 1288, 1628, 1767  
GetUseBestVisual, 48  
GetUser, 1761  
GetUserConfigDir, 1526  
GetUserData, 1460  
GetUserDataDir, 1526  
GetUserId, 1601  
GetUserID, 413  
GetUserInfo, 1761  
GetUserLocalDataDir, 1527  
GetUsername, 395  
GetUserName, 1601  
GetUserScale, 470  
GetValidator, 1820  
GetValue, 32, 182, 230, 240, 322, 327, 330, 342, 703, 793, 1034, 1251, 1330, 1469, 1499, 1502, 1644, 1657, 1686, 1694, 1878  
GetValueAsBool, 794  
GetValueAsCustom, 794  
GetValueAsDouble, 794  
GetValueAsLong, 794  
GetValueClassInfo, 1779  
GetVariantType, 331  
GetVendorName, 49, 273  
GetVerbose, 1024  
GetVersion, 568, 1280, 1869, 1881  
GetVGap, 798  
GetView, 526, 536, 795  
GetViewName, 544, 1782  
GetViewRect, 990  
GetViews, 549  
GetViewStart, 1419  
GetViewWidth, 754  
GetVirtualSize, 1419, 1821  
GetVisibleBegin, 1793  
GetVisibleEnd, 1793  
GetVisibleLineForCaretPosition, 1330  
GetVisited, 893  
GetVisitedColour, 893  
GetVMax, 951  
GetVMin, 951  
GetVoidPtr, 1774  
GetVolume, 620, 1063  
GetVolumeSeparator, 620  
GetVPosition, 951  
GetW, 1270  
GetWeekDay, 177, 364, 372  
GetWeekDayInSameWeek, 371  
GetWeekDayName, 358  
GetWeekOfMonth, 365  
GetWeekOfYear, 365  
GetWeeks, 345, 1687  
GetWeight, 662  
GetWheelDelta, 1125  
GetWheelRotation, 1125  
GetWhereClause, 434  
GetWidestItem, 1155  
GetWidestItemWidth, 1155  
GetWidth, 131, 316, 790, 823, 899, 918, 1005, 1180, 1256, 1270, 1441  
GetWildcard, 603  
GetWindow, 32, 179, 886, 1460, 1713, 1768  
GetWindow1, 1512  
GetWindow2, 1512  
GetWindowBeingRemoved, 1507  
GetWindowBorderSize, 1821  
GetWindowMenu, 1056  
GetWindowStyleFlag, 1821  
GetWindowVariant, 1821  
GetWritableChar, 1567  
GetWriteBuf, 1068, 1567  
GetWxObjectPtr, 1774  
GetX, 960, 1125, 1256, 1270, 1430, 1507  
GetXMax, 951  
GetXMin, 951  
GetXRCID, 1881  
GetY, 961, 1126, 1256, 1270, 1430, 1507  
GetYear, 364  
GetYearDay, 373  
GetYears, 346  
GetYMax, 951  
GetYMin, 951  
GetZMax, 951  
GetZMin, 952  
GetZoomControl, 1198  
GetZPosition, 952, 956  
GiveFeedback, 560  
Global functions, 1019  
GoToLine, 1660

GradientFillConcentric, 470  
GradientFillLinear, 470  
Grant, 395  
Green, 216  
Grid sample, 2035  
GridLinesEnabled, 751  
Gripper, 113  
GripperTop, 113  
GuessType, 1661  
GUI system, 1985

## —H—

HandleEvent, 56  
HandleReturn, 778  
HandleSettingChange, 1714  
HandleTag, 869  
Handling socket events, 1475  
HangUp, 508  
Hardware architectures (CPU), 1987  
Hardware buttons in wxWinCE, 2237  
Hardware type, 1987  
HasAlignment, 771, 1288, 1621  
HasAlpha, 918  
HasBackgroundColour, 175, 770, 1008, 1289, 1621  
HasBorder, 114, 176, 1400  
HasBorderColour, 175  
HasBulletName, 1289, 1628  
HasBulletNumber, 1289, 1628  
HasBulletStyle, 1289, 1628  
HasBulletText, 1289, 1628  
HasCaption, 114  
HasCapture, 1821  
HasCharacterAttributes, 1331  
HasCharacterStyleName, 1289, 1628  
HasCloseButton, 114  
HasCursor, 1430  
HasEditor, 771  
HasEnding, 868  
HasEntry, 273  
HasExt, 620  
HasFiles, 512  
HasFlag, 114, 1289, 1822  
HasFont, 175, 771, 1008, 1289, 1621  
HasFontFaceName, 1289  
HasFontItalic, 1290  
HasFontSize, 1291  
HasFontUnderlined, 1291  
HasFontWeight, 1292  
HasFragment, 1761  
HasGripper, 114  
HasGripperTop, 114  
HasGroup, 273  
HasHandlerForPath, 634  
HasLeftIndent, 1290, 1621  
HasLineSpacing, 1290, 1629  
HasListStyleName, 1290, 1629  
HasMask, 918  
HasMaximizeButton, 114  
HasMinimizeButton, 114  
HasModifiers, 961  
HasMoreTokens, 1585  
HasMultiplePages, 1822  
HasMultipleSelection, 1787  
HasName, 620  
HasNextPage, 1861  
HasOption, 919, 1593  
HasOutlineLevel, 1290, 1629  
HasPage, 1217, 1372  
HasPageBreak, 1290, 1629  
HasParagraphAttributes, 1331  
HasParagraphSpacingAfter, 1290, 1629  
HasParagraphSpacingBefore, 1290, 1629  
HasParagraphStyleName, 1291, 1629  
HasParam, 868, 1392, 1887  
HasPath, 1762  
HasPinButton, 115  
HasPort, 1762  
HasPOV, 952  
HasPOV4Dir, 952  
HasPOVCTS, 952  
HasPrevPage, 1861  
HasProp, 1875  
HasQuery, 1762  
HasRange, 780  
HasRenderer, 771  
HasRightIndent, 1291, 1621  
HasRudder, 952  
HasScheme, 1762  
HasScrollbar, 1822  
HasSelection, 1331, 1589  
HasServer, 1762  
HasSourceLocation, 1521  
HasStderr, 58  
HasStream, 485  
HasSubDirs, 512  
HasSubKey, 1274  
HasSubKeys, 1275  
HasSymbol, 566  
HasTabs, 1291, 1621  
HasTextColour, 175, 770, 1008, 1291, 1622  
HasTextCtrl, 1185  
HasTextEffects, 1291, 1629  
HasTransparentBackground, 1822  
HasU, 952  
HasURL, 1291, 1630  
HasUser, 1762  
HasV, 952  
HasValue, 1274  
HasValues, 1274  
HasVolume, 621  
HasZ, 952  
HaveRects, 1270  
height, 965, 1253  
Help Files Format, 2181  
Helper functions, 1111  
Hibernation in wxWinCE, 2237  
Hide, 115, 179, 556, 1451, 1822  
HideCellEditControl, 754  
HideHint, 102  
HidePopup, 240  
Hiding controls using sizers, 2100  
HistoryBack, 874

HistoryCanBack, 874  
 HistoryCanForward, 874  
 HistoryClear, 875  
 HistoryForward, 875  
 HitTest, 32, 173, 978, 990, 1141, 1310, 1331, 1644, 1741, 1793  
 Hostname, 944, 945, 946  
 Hour, 1687  
 Hours, 1687  
 hover, 140  
 How events are processed, 2078  
 How the wxGrid classes relate to each other, 2144  
 How wxRichTextCtrl is implemented, 2195  
 HSVtoRGB, 918  
 HSVValue, 918  
 HSVValue::HSVValue, 918  
 HTML Printing, 2180  
 HTML samples, 2035



Iconize, 501, 1715  
 Iconized, 903  
 identifiers, 2082  
 Identifying art resources, 88  
 Image sample, 2035  
 IMPLEMENT\_ABSTRACT\_CLASS, 1966  
 IMPLEMENT\_ABSTRACT\_CLASS2, 1966  
 IMPLEMENT\_APP, 1967  
 IMPLEMENT\_CLASS, 1967  
 IMPLEMENT\_CLASS2, 1967  
 IMPLEMENT\_DYNAMIC\_CLASS, 1967, 1968  
 IMPLEMENT\_DYNAMIC\_CLASS2, 1968  
 ImportXML, 1392  
 IncBy, 1441  
 IncRef, 769, 1779  
 IncTo, 1442  
 Index, 80, 86, 1567  
 IndexOf, 969, 1135  
 Inflate, 1256  
 InheritAttributes, 1822  
 Init, 159, 247, 569, 711, 1017, 1310, 1331, 1361, 1630  
 InitAllHandlers, 1881  
 InitAlpha, 919  
 InitColWidths, 754  
 InitCommandEvent, 1332  
 InitDialog, 1172, 1823  
 InitDocument, 544  
 Initialization functions, 1111  
 Initialize, 257, 532, 811, 1200, 1512  
 InitializeClasses, 191  
 InitParser, 861  
 InitRowHeights, 754  
 InitStandardHandlers, 131, 920, 1310  
 Input Filters, 2182  
 insert, 802, 806, 972  
 Insert, 80, 81, 86, 92, 138, 139, 290, 434, 969, 970, 1083, 1094, 1451, 1774, 2174  
 InsertCell, 827  
 InsertCheckItem, 1084  
 InsertChild, 1875

InsertChildAfter, 1875  
 InsertCols, 754, 795  
 InsertColumn, 991  
 InsertControl, 1705  
 InsertDir, 621  
 InsertHandler, 131, 920, 1310  
 InsertImageWithUndo, 1310  
 InsertItem, 992, 1742  
 InsertItems, 978  
 InsertLine, 1662  
 InsertNewlineWithUndo, 1311  
 InsertPage, 108, 1142, 1725  
 InsertPane, 102  
 InsertRadioItem, 1084  
 InsertRows, 755, 795  
 InsertSeparator, 1084, 1705  
 InsertSpacer, 1451  
 InsertStretchSpacer, 1452  
 InsertSubPage, 1725  
 InsertTextWithUndo, 1311  
 InsertTool, 1705  
 Installing your PocketPC and Smartphone applications, 2240  
 Internat(ionalization) sample, 2036  
 InterruptWait, 1477  
 Intersect, 1257, 1266, 1267  
 Intersects, 707, 1257  
 Introduction, 2047, 2052, 2077, 2088, 2127, 2143  
 InvalidateBestSize, 1823  
 Invert, 728  
 Invoke, 122  
 IPAddress, 945, 947  
 Is3rdStateAllowedForUser, 182  
 Is3State, 183  
 IsAbsolute, 621  
 IsActive, 49, 1715  
 IsAlive, 1675  
 IsAllowed, 1147  
 IsAllowedTraceMask, 1025  
 IsAlwaysMaximized, 1715  
 IsAlwaysOnline, 508  
 IsAnotherRunning, 1439  
 IsAscii, 1567  
 IsAvailable, 1018  
 IsBetween, 366  
 IsBold, 1742  
 IsBottomDockable, 115  
 IsButton, 956, 1126  
 IsCancelButtonVisible, 1435  
 IsCaseSensitive, 621  
 IsCellEditControlEnabled, 755  
 IsCharacterStyle, 1292, 1630  
 IsCheckable, 1104  
 IsChecked, 183, 186, 255, 1084, 1095, 1104  
 IsCollapsed, 212  
 IsColNull, 435  
 IsCommandEvent, 574  
 IsCompatible, 1281  
 IsConnected, 1477  
 IsConnectedEvent, 505  
 IsCreated, 247, 777  
 IsCurrent, 1787

- IsCurrentCellReadOnly, 755
- IsCursorClosedOnCommit, 435
- IsData, 1478
- IsDefault, 1292, 1622, 1630
- IsDefaultStyleShowing, 1332
- IsDetached, 1675
- IsDialing, 507
- IsDir, 64, 622
- IsDirReadable, 621
- IsDirty, 258
- IsDirWritable, 621
- IsDisconnected, 1478
- IsDocked, 115
- IsDoubleBuffered, 1823
- IsDST, 375
- IsDSTApplicable, 359
- IsDynamic, 191
- IsEarlierThan, 366
- IsEditable, 755, 1332, 1644
- IsEditCancelled, 322, 1003, 1751
- IsEmpty, 81, 86, 291, 970, 1257, 1267, 1275, 1568, 1644, 1742
- IsEmptyCell, 793
- IsEnabled, 1085, 1095, 1104, 1823
- IsEncodingAvailable, 677
- IsEqual, 728, 1267
- IsEqualTo, 366, 1687
- IsEqualUpTo, 367
- IsErrorAvailable, 1226
- IsExpanded, 212, 1742
- IsExpandingEnvVars, 273
- IsExposed, 1824
- IsFalse, 1593
- IsFileExecutable, 621
- IsFileReadable, 622
- IsFileWritable, 622
- IsFixed, 115
- IsFixedWidth, 661
- IsFloatable, 115
- IsFloating, 115
- IsFrozen, 1333, 1824
- IsFullScreen, 1715
- IsFullySpecified, 1441
- IsFwdOnlyCursors, 389, 396
- IsGregorianDate, 365
- IsHatch, 154
- IsHoliday, 176
- IsHotSensitive, 1519
- IsIconInstalled, 1608
- IsIconized, 502, 1715
- IsIdentity, 728
- IsInputAvailable, 1227
- IsInputOpened, 1227
- IsInSelection, 755
- IsInside, 1260
- IsItemEnabled, 1246
- IsItemShown, 1246
- IsKindOf, 191, 1149
- IsLaterThan, 366
- IsLeapYear, 358
- IsLeftDockable, 115
- IsLoaded, 566, 1018
- IsLocalHost, 946
- IsLongerThan, 1687
- IsMadeByUnix, 1894
- IsMain, 1675
- IsMainLoopRunning, 49
- IsMaximized, 1716
- IsModal, 502
- IsModified, 549, 1311, 1333, 1645
- IsMovable, 115
- IsMove, 956
- IsMultiLine, 1333, 1645
- IsNegative, 1687
- IsNodeExpanded, 1725
- IsNull, 729, 1568, 1687, 1774
- IsNumber, 1568
- IsNumbered, 1367
- IsOfClass, 1887
- IsOfType, 1112
- IsOk, 24, 41, 116, 132, 154, 179, 217, 261, 297, 302, 492, 507, 590, 591, 609, 622, 629, 663, 900, 903, 922, 953, 1018, 1134, 1162, 1170, 1180, 1192, 1203, 1210, 1478, 1496, 1546, 1609, 1748, 1765, 1869
- IsOneShot, 1681
- IsOnline, 508
- IsOpen, 397
- IsOpened, 197, 513, 586, 595, 1275, 1616, 1659
- IsOutside, 1374
- IsOwnEvent, 506
- IsPageScroll, 1126
- IsParagraphStyle, 1292, 1630
- IsPassingMessages, 1026
- IsPathSeparator, 622
- IsPaused, 1675
- IsPickerCtrlGrowable, 1185
- IsPlaying, 44, 1496
- IsPopup, 1099
- IsPopupShown, 240
- IsPopupWindowState, 239
- IsPositionVisible, 1333
- IsPositive, 1687
- IsPreview, 1218
- IsPrimary, 522
- IsQueryOnly, 435
- IsReadOnly, 64, 756, 771
- IsRecordingDefaults, 273
- IsReference, 1762
- IsRelative, 622
- IsResizable, 116
- IsRetained, 1420, 1824
- IsRightDockable, 116
- IsRunning, 1676, 1681
- IsSameAs, 1150, 1568
- IsSameDate, 367
- IsSameTime, 367
- IsSearchButtonVisible, 1435
- IsSeekable, 1546
- IsSelected, 321, 979, 1010, 1742, 1787
- IsSelection, 255, 756
- IsSelectionAligned, 1333
- IsSelectionBold, 1333
- IsSelectionItalics, 1333

IsSelectionUnderlined, 1333  
 IsSeparator, 1104  
 IsShorterThan, 1688  
 IsShown, 116, 1452, 1461, 1824  
 IsShownOnScreen, 1825  
 IsSingleLine, 1334, 1645  
 IsSizer, 1461  
 IsSortOrderAscending, 316  
 IsSpacer, 1461  
 IsSplit, 1512  
 IsStrictlyBetween, 366  
 IsSubMenu, 1104  
 IsSupported, 198  
 IsText, 1894  
 IsTextCtrlGrowable, 1185  
 IsToolBar, 116  
 IsTopDockable, 116  
 IsTopLevel, 1825  
 IsTransparent, 920  
 IsTrueValue, 775  
 IsType, 1774  
 IsUsingNativeDecorations, 1716  
 IsUsingUniversalWidgets, 58, 1192  
 IsValid, 363, 1262  
 IsValidFacename, 673  
 IsValueKindOf, 1775  
 IsVertical, 704, 1534  
 IsVisible, 179, 544, 756, 1352, 1742, 1793  
 IsWestEuropeanCountry, 359  
 IsWhitespaceOnly, 1875  
 IsWindow, 1461  
 IsWithin, 1375  
 IsWord, 1568  
 IsWorkDay, 365  
 IsZMove, 956  
 Item, 81, 86, 970  
 ItemHasChildren, 1743

## —K—

Key access, 267  
 Keyboard and mouse actions, 2144  
 KeyboardNavigate, 1334  
 KeywordSearch, 813, 842, 850  
 Kill, 1227, 1676

## —L—

Last, 81, 87, 1569  
 LastCount, 1478  
 LastError, 1478  
 LastRead, 942  
 LastWrite, 1157  
 Layer, 116  
 Layout, 823, 1452, 1825  
 Layout sample, 2036  
 LayoutContent, 1334  
 LayoutDialog, 1234  
 LayoutFrame, 963  
 LayoutMDIFrame, 964  
 LayoutWindow, 964  
 LazyCreate, 248

Leave, 295  
 Leaving, 1126  
 left, 966  
 Left, 117, 1457, 1569  
 LeftClick, 334  
 LeftDClick, 1126  
 LeftDockable, 117  
 LeftDown, 1126  
 LeftIsDown, 1126  
 LeftOf, 940  
 LeftUp, 1127  
 Len, 1569  
 Length, 586, 595, 1063, 1569, 1617  
 Library binary compatibility, 2232  
 Limits And Compatibility, 2218  
 LimitTo, 1375  
 LineBreak, 1334  
 List of reference-counted wxWidgets classes, 2046  
 Listctrl sample, 2036  
 ListLoaded, 566  
 Load, 41, 566, 607, 1063, 1064, 1280, 1869, 1870, 1881  
 LoadBitmap, 1881  
 LoadDialog, 1881  
 LoadFile, 41, 44, 131, 147, 813, 875, 899, 920, 921, 931, 1311, 1334, 1352, 1645  
 LoadFrame, 1882  
 LoadIcon, 1882  
 LoadLibrary, 524  
 LoadMenu, 1882  
 LoadMenuBar, 1882  
 LoadObject, 549  
 LoadPage, 875  
 LoadPanelInfo, 103  
 LoadPanel, 1882  
 LoadPerspective, 103  
 LoadToolBar, 1882  
 LoadURI, 1064  
 LoadURIWithProxy, 1064  
 LocalHost, 946, 947  
 Lock, 1133  
 LogError, 397  
 Logging functions, 1019  
 LogicalToDeviceX, 471  
 LogicalToDeviceXRel, 471  
 LogicalToDeviceY, 471  
 LogicalToDeviceYRel, 471  
 Looking up an archive entry by name, 2225  
 Lower, 1569, 1825  
 LowerCase, 1569

## —M—

m\_altDown, 957, 1121  
 m\_childDocument, 525, 536  
 m\_childView, 525, 536  
 m\_combo, 246  
 m\_commandProcessor, 546  
 m\_controlDown, 958, 1121  
 m\_currentView, 527  
 m\_defaultDocumentNameCounter, 527

---

m\_defaultExt, 540  
m\_description, 540  
m\_directory, 541  
m\_docClassInfo, 541  
m\_docs, 528  
m\_docTypeName, 541  
m\_documentFile, 546  
m\_documentManager, 541  
m\_documentModified, 546  
m\_documentTemplate, 546  
m\_documentTitle, 546  
m\_documentTypeName, 546  
m\_documentViews, 547  
m\_fileFilter, 541  
m\_fileHistory, 527, 528, 606  
m\_fileHistoryN, 606  
m\_fileMaxFiles, 606  
m\_fileMenu, 606  
m\_files, 557  
m\_flags, 528, 541  
m\_keyCode, 958  
m\_leftDown, 1121  
m\_linesPerAction, 1122  
m\_maxDocsOpen, 527  
m\_metaDown, 958, 1121  
m\_middleDown, 1121  
m\_noFiles, 557  
m\_pos, 558  
m\_propagationLevel, 573  
m\_refData, 1148  
m\_rightDown, 1121  
m\_shiftDown, 958, 1121  
m\_templates, 528  
m\_thread, 1679  
m\_viewClassInfo, 541  
m\_viewDocument, 1780  
m\_viewFrame, 1780  
m\_viewTypeName, 541, 1780  
m\_wheelDelta, 1122  
m\_wheelRotation, 1122  
m\_x, 958, 1122  
m\_y, 958, 1122  
MacFindDefaultTypeAndCreator, 622  
MacRegisterDefaultTypeAndCreator, 623  
Macros for template array definition, 73  
MacSetDefaultTypeAndCreator, 623  
MainLoop, 49  
Make your own reference-counted class, 2047  
MakeAbsolute, 623  
MakeCellVisible, 756  
MakeConnection, 192, 478, 1610  
MakeDefaultName, 532  
MakeFromTimezone, 374  
MakeKey, 809  
MakeLower, 1569  
MakeModal, 1826  
MakeNewDocumentName, 532  
MakeNull, 1775  
MakeRelativeTo, 623  
MakeString, 1775  
MakeTimezone, 374  
MakeUpper, 1570  
MakeUTC, 374  
MapScreenSizeToDevice, 1219  
MapScreenSizeToPage, 1219  
MapScreenSizeToPageMargins, 1219  
MapScreenSizeToPaper, 1218  
MarkAsSaved, 258  
MarkDirty, 1334, 1646  
Matches, 1263, 1570  
Matching, 2217  
max\_size, 972  
Maximize, 1049, 1716  
MaximizeButton, 117  
MaxSize, 117  
MaxX, 471  
MaxY, 471  
mb\_str, 1570  
MB2WC, 297, 1039, 1043, 1044, 1045, 1046  
Mediaplayer sample, 2036  
Member, 970, 1775  
Memory management, 74, 1557  
Menubars and toolbars in wxWinCE, 2238  
Merge, 1622  
MessageParameters class, 640  
MetaDown, 785, 787, 789, 961, 1127  
Metasyntax, 2216  
Mid, 1570  
MiddleDClick, 1127  
MiddleDown, 1127  
MiddleIsDown, 1127  
MiddleUp, 1127  
Millisecond, 1689  
Milliseconds, 1688  
Minimal sample, 2031  
MinimizeButton, 117  
MinSize, 117  
Minute, 1688  
Minutes, 1688  
MinX, 471  
MinY, 471  
Mirror, 923  
Miscellaneous, 1557, 1989  
Miscellaneous functions, 267  
Mkdir, 624  
MkDir, 697  
Modify, 549, 1311  
ModifyColumn, 398  
Modifying an archive, 2224  
Month, 346  
Months, 346  
More DDE details, 2178  
MoreRequested, 905  
Movable, 117  
Move, 179, 556, 1826  
MoveAfterInTabOrder, 1827  
MoveBeforeInTabOrder, 1827  
MoveCaret, 1334  
MoveCaretBack, 1335  
MoveCaretForward, 1335  
MoveCursorDown, 756  
MoveCursorDownBlock, 757  
MoveCursorLeft, 756  
MoveCursorLeftBlock, 757

MoveCursorRight, 757  
MoveCursorRightBlock, 757  
MoveCursorUp, 757  
MoveCursorUpBlock, 758  
MoveDown, 1335  
MoveEnd, 1335  
MoveHome, 1335  
MoveLeft, 1335  
MovePageDown, 758  
MovePageUp, 758  
MoveRight, 1335  
MoveToLineEnd, 1335  
MoveToLineStart, 1336  
MoveToParagraphEnd, 1336  
MoveToParagraphStart, 1336  
MoveToPoint, 730  
MoveUp, 1336  
Moving, 1127  
Multiply, 346, 1688

## —N—

Name, 117  
Native font information, 2121  
Navigate, 32, 1827  
Neg, 347, 1688  
Negate, 346, 1688  
new, 1151  
New, 663  
NewEntry, 61  
Newline, 1336  
NewStream, 61, 645  
Next, 809  
normal, 139  
Normalize, 624  
Notebook sample, 2036  
Notify, 1478, 1682  
NotifyEvent, 32  
Now, 359  
Nth, 970  
NullList, 1775  
Number, 291, 970, 1311, 1336  
Number of elements and simple item access, 75  
NumberList, 1311, 1336

## —O—

Object comparison, 2046  
Object destruction, 2046  
Obtaining individual components, 1757  
Offset, 1257, 1267, 1268  
OffsetLogicalOrigin, 1220  
Ok, 472, 1108, 1223  
OnAcceptConnection, 483, 1432, 1615  
OnActivate, 526, 536  
OnActivateView, 1782  
OnAdvise, 278, 480, 1612  
OnAssertFailure, 49  
OnBeginDocument, 1220  
OnBeginPrinting, 1220  
OnButtonClick, 241  
OnCalculateLayout, 1397

OnCellClicked, 875  
OnCellMouseHover, 876  
OnChangedViewList, 549  
OnChangeFilename, 1782  
OnChar, 1669  
OnClear, 1337  
OnClose, 1782  
OnCloseDocument, 550  
OnCloseWindow, 526, 537, 538, 540, 1200, 1505  
OnClosingDocument, 1782  
OnCmdLineError, 50  
OnCmdLineHelp, 50  
OnCmdLineParsed, 50  
OnComboDoubleClick, 248  
OnComboKeyEvent, 248  
OnCompareItems, 1743  
OnContextMenu, 1337  
OnCopy, 1337  
OnCreate, 550, 1782  
OnCreateClient, 1056  
OnCreateCommandProcessor, 550  
OnCreateFileHistory, 532  
OnCreatePrintout, 1783  
OnCreateStatusBar, 688  
OnCreateToolBar, 689  
OnCut, 1337  
OnData, 562  
OnDir, 519  
OnDisconnect, 279, 480, 1613  
OnDismiss, 248  
OnDoubleClickSash, 1513  
OnDragOver, 563  
OnDraw, 1421, 1783  
OnDrawBackground, 1155, 1787  
OnDrawItem, 1155, 1787  
OnDrawSeparator, 1788  
OnDrop, 562, 605, 1655  
OnDropFiles, 605, 1337, 1646  
OnDropText, 1655  
OnEndDocument, 1220  
OnEndPrinting, 1221  
OnEnter, 563  
OnEntryUpdated, 68, 1897  
OnExceptionInMainLoop, 51  
OnExecute, 279, 480, 1613  
OnExit, 51, 1117, 1676  
OnFacename, 673  
OnFatalException, 51  
OnFile, 519  
OnFileClose, 533  
OnFileCloseAll, 533  
OnFileNew, 533  
OnFileOpen, 533  
OnFileRevert, 533  
OnFileSave, 533  
OnFileSaveAs, 533  
OnFontEncoding, 673  
OnFrameClose, 1032  
OnFrameCreate, 1031  
OnFrameDelete, 1032  
OnGetItem, 855, 1380  
OnGetItemAttr, 993



- OnGetItemColumnImage, 993
- OnGetItemImage, 993
- OnGetItemMarkup, 855
- OnGetItemText, 994
- OnGetLineHeight, 1793
- OnGetLinesHint, 1794
- OnInit, 52, 1118
- OnInitCmdLine, 52
- OnInternalIdle, 1827
- OnLeave, 564
- OnLeftClick, 1706
- OnLeftDown, 1381
- Online help in wxWinCE, 2240
- OnLinkClicked, 855, 876
- OnLog, 1022
- OnMakeConnection, 193, 478, 1610
- OnMeasureItem, 1156, 1788
- OnMeasureItemWidth, 1156
- OnMouseEnter, 1706
- OnNewDocument, 550
- OnOpenDocument, 550
- OnOpenError, 519
- OnOpeningURL, 877
- OnPaint, 1197
- OnPaste, 1337
- OnPoke, 279, 481, 1613
- OnPopup, 248
- OnPreparePrinting, 1221, 1373
- OnPrintPage, 1221, 1373
- OnQueryLayoutInfo, 1397
- OnQuit, 814
- OnRedo, 1337
- OnRequest, 279, 481, 1613
- OnRightClick, 1707
- OnRun, 52
- OnSashPositionChange, 1513
- OnSaveDocument, 550
- OnSaveModified, 551
- OnSelChange, 1142
- OnSelect, 1381
- OnSelectAll, 1338
- OnServerReply, 495
- OnSetTitle, 877
- OnStackFrame, 1522
- OnStartAdvise, 279, 481, 1613
- OnStopAdvise, 279, 481, 1613
- OnSysColourChanged, 502, 1172
- OnSysRead, 1546
- OnSysSeek, 1546
- OnSysTell, 1546
- OnSysWrite, 1546
- OnTerminate, 1228
- OnUndo, 1338
- OnUnhandledException, 52
- OnUnsplit, 1513
- OnUpdate, 1783
- OnUpdateClear, 1338
- OnUpdateCopy, 1338
- OnUpdateCut, 1338
- OnUpdatePaste, 1338
- OnUpdateRedo, 1338
- OnUpdateSelectAll, 1338
- OnUpdateUndo, 1339
- Open, 198, 399, 436, 513, 587, 595, 1228, 1275, 1616, 1662
- OpenContainer, 886
- OpenEntry, 65, 1604, 1896
- OpenFile, 636, 639
- OpenURL, 861
- Operating systems, 1986
- Operation, 1059
- Operations, 612, 1684
- operator, 363, 706, 709, 1576
  - =, 348, 628, 1193
- operator-, 347
- operator
  - =, 84, 306
- operator-, 1035
- operator-, 1035
- operator-, 1035
- operator-, 1035
- operator-, 1035
- operator--, 1036
- operator--, 1036
- operator-, 1376
- operator-, 1376
- operator
  - =, 1443
- operator-, 1688
- operator, 1748
- operator
  - =, 1748
- operator
  - =, 1776
- operator
  - =, 1776
- operator
  - =, 1776
- operator
  - =, 1776
- operator
  - =, 1776
- operator
  - =, 1776
- operator
  - =, 1776
- operator
  - =, 1776
- operator
  - =, 1776
- operator
  - =, 1776
- operator
  - =, 1776
- operator
  - =, 1776
- operator -, 1194, 1443
- operator --, 451
- operator !, 1748
- operator !=, 84, 156, 218, 306, 668, 1182, 1194, 1258, 1443, 1578, 1748, 1776, 1777
- operator (), 1578
- operator \*, 1406, 1443
- operator \*=, 1443

operator /, 1443  
 operator /=, 1443  
 operator [], 1403, 1577, 1777  
 operator +, 1194, 1443, 1577  
 operator ++, 451, 1271  
 operator +=, 1194, 1443, 1577  
 operator <, 1579  
 operator <<, 1578, 1653  
 operator <=, 1579  
 operator =, 24, 42, 84, 135, 156, 217, 219, 302, 668, 670, 901, 929, 1164, 1170, 1182, 1194, 1206, 1211, 1258, 1269, 1443, 1577, 1775, 1776  
 operator -=, 1194  
 operator -=, 1194  
 operator -=, 1443  
 operator ==, 84, 156, 217, 306, 668, 1182, 1194, 1258, 1443, 1578, 1748, 1762, 1776  
 operator >, 1578  
 operator ->, 1406  
 operator >=, 1578, 1579  
 operator >>, 1578  
 operator bool, 1271  
 operator const wxChar\*, 1578  
 operator double, 1777  
 operator long, 1777  
 operator void\*, 1778  
 operator wxChar, 1777  
 operator wxChar \*, 1580, 1581  
 operator wxDateTime, 1778  
 operator wxString, 1777  
 operator!, 706, 709, 1576  
 operator!=, 348, 628, 1193  
 operator\*, 68  
 operator\*=, 346, 1688  
 operator[], 39, 84, 802, 1660  
 operator+, 1035, 1375  
 operator++, 68, 1035  
 operator+=, 344, 347, 367, 1685  
 operator=, 78, 119, 628, 902, 1033, 1034, 1298, 1362, 1376, 1603, 1633, 1871, 1877, 1894  
 operator-=, 368  
 operator-=, 368  
 operator-=, 1689  
 operator==, 348, 628, 706, 709, 1193, 1376  
 Operators, 1194, 1443, 1748  
 Options, 268  
 OrderBy, 437  
 Other Python GUIs, 2199  
 Other string related functions and classes, 2050

## —P—

ParamName, 39  
 ParamType, 39  
 Parse, 209, 861  
 ParseDate, 369  
 ParseDateTime, 369  
 ParseFormat, 368  
 ParseInner, 870  
 ParseRfc822Date, 368  
 ParseTime, 369  
 Parsing and formatting dates, 354  
 Parsing command line, 205  
 PassMessages, 1026  
 Paste, 230, 241, 1339, 1646  
 PasteFromClipboard, 1312  
 Path management, 264  
 Pause, 1064, 1544, 1676  
 Peek, 943, 1482  
 Pending, 53  
 PercentOf, 940  
 PinButton, 118  
 Play, 44, 1064, 1108, 1496  
 Player controls, 1060  
 Pluggable event handlers, 2081  
 Poke, 280, 481, 1614  
 Pop, 92  
 pop\_back, 972  
 pop\_front, 972  
 PopEventHandler, 1828  
 PopExtension, 646  
 PopStatusText, 1539  
 PopTagHandler, 863  
 PopupMenu, 1609, 1828  
 Position, 118  
 PositionCaret, 1339  
 PositionToXY, 1339, 1646  
 Positive thinking, 13  
 Post, 1428  
 Precompiled headers, 8  
 PrepareDC, 1421  
 Prepend, 1085, 1452, 1570  
 PrependCheckItem, 1085  
 PrependDir, 625  
 PrependItem, 1743  
 PrependRadioItem, 1086  
 PrependSeparator, 1086  
 PrependSpacer, 1452  
 PrependStretchSpacer, 1453  
 PreProcessTip, 1690  
 PreviewBuffer, 1370  
 PreviewFile, 835, 1370  
 PreviewText, 835  
 Print, 1213, 1223  
 PrintBuffer, 1370  
 PrintClasses, 485  
 PrintDialog, 1213  
 Printf, 1571  
 PrintFile, 835, 1370  
 PrintfV, 1571  
 PrintStatistics, 485  
 PrintText, 835  
 Process, 492  
 ProcessCommand, 689

Pad, 1570  
 PageDown, 1339  
 PageSetup, 836, 1369  
 PageUp, 1339  
 PaintBackground, 777, 1339  
 PaintComboControl, 248  
 PaintPage, 1223  
 PaneBorder, 118  
 ParamCount, 39

ProcessDockResult, 103  
 ProcessEvent, 580  
 ProcessMessage, 53  
 ProcessMouseClicked, 824  
 Programming with wxBoxSizer, 2102  
 Programming with wxFlexGridSizer, 2105  
 Programming with wxGridSizer, 2105  
 Programming with wxRichTextCtrl, 2193  
 Programming with wxStaticBoxSizer, 2105  
 PromoteList, 1312, 1339  
 Proportion, 1457  
 Pulse, 705, 1232  
 Push, 93  
 push\_back, 972  
 push\_front, 972  
 PushEventHandler, 1829  
 PushFront, 61, 646  
 PushStatusText, 1539  
 PushTagHandler, 862  
 Put, 809  
 PutC, 1157  
 PutChar, 1549, 1667  
 PutNextDirEntry, 70, 1606, 1899  
 PutNextEntry, 70, 1607, 1899  
 PutProperty, 123  
 Pwd, 697

—Q—

Quantize, 1238  
 Query, 437  
 Query database, 1111  
 QueryBySqlStmt, 439  
 QueryMatching, 440  
 QueryOnKeyFields, 441  
 QueryValue, 1275  
 Quit, 814

—R—

Raise, 1830  
 rbegin, 973  
 Read, 273, 274, 587, 596, 943, 1483, 1548, 1779  
 Read16, 310, 1664  
 Read16S, 1664  
 Read32, 310, 1664  
 Read32S, 1665  
 Read64, 310, 311  
 Read8, 310, 1664  
 Read8S, 1664  
 ReadAll, 587  
 ReadCustomization, 842, 845, 847, 850, 877  
 ReadDouble, 311, 1665  
 ReadFile, 838  
 ReadLine, 1665  
 ReadMailcap, 1113  
 ReadMimeType, 1113  
 ReadMsg, 1483  
 ReadString, 311, 1665  
 ReadWord, 1665  
 Realize, 1543, 1707  
 RecalcSizes, 150, 774, 1453

ReceiveFrom, 307  
 Reconnect, 1236  
 ReCreateTree, 713  
 Red, 217  
 Redirect, 1229  
 Redo, 230, 256, 1340, 1647  
 Ref, 1150  
 Reference counting and why you shouldn't care  
   about it, 2050  
 Refresh, 442, 1095, 1830  
 RefreshAll, 1794  
 RefreshItem, 994  
 RefreshItems, 994  
 RefreshLine, 1794  
 RefreshLines, 1794  
 RefreshLists, 850  
 RefreshRect, 1830  
 RegisterDataType, 758  
 RegisterHotKey, 1830  
 Regular Expression Character Names, 2219  
 Regular Expression Syntax, 2210  
 release, 1405  
 ReleaseCapture, 953  
 ReleaseMouse, 1831  
 Remaining issues, 2242  
 remove, 973  
 Remove, 61, 81, 87, 93, 230, 241, 646, 937, 1086,  
   1095, 1340, 1453, 1571, 1647  
 RemoveAll, 937  
 RemoveAt, 82, 87  
 RemoveCharacterStyle, 1389  
 RemoveChild, 1831, 1876  
 RemoveDir, 625  
 RemoveDocument, 533  
 RemoveEventHandler, 1312, 1832  
 RemoveFile, 492, 1072  
 RemoveFileFromHistory, 608  
 RemoveGrowableCol, 654  
 RemoveGrowableRow, 654  
 RemoveHandler, 133, 923, 1313  
 RemoveHelp, 818  
 RemoveIcon, 1609  
 RemoveLast, 1571  
 RemoveLastDir, 625  
 RemoveLine, 1662  
 RemoveListStyle, 1390  
 RemoveMenu, 608  
 RemoveNotifier, 327  
 RemovePage, 108, 1143  
 RemoveParagraphStyle, 1390  
 RemoveStyle, 1390  
 RemoveTool, 1707  
 RemoveTraceMask, 1025  
 RemoveView, 551  
 RemoveViewingColumn, 327  
 Removing items, 75  
 Rename, 697, 1275  
 Rename entries/groups, 267  
 RenameEntry, 275  
 RenameGroup, 275  
 RenameValue, 1275  
 rend, 973

Render, 334, 833  
Render sample, 2037  
Rendering media, 1058  
RenderPage, 1223  
Reparent, 1832  
Replace, 230, 241, 923, 937, 1095, 1264, 1340, 1453, 1571, 1647  
ReplaceAll, 1264  
ReplaceFirst, 1264  
ReplaceWindow, 1514  
ReportError, 1213  
Request, 280, 481, 1614  
RequestMore, 905  
RequestUserAttention, 1716  
Rescale, 923  
ReserveSpaceEvenIfHidden, 1457  
reset, 1403, 1405  
Reset, 492, 778, 1271  
ResetAndClearCommands, 1313  
ResetAttr, 173  
ResetBoundingBox, 472  
ResetBuffer, 1550  
ResetClip, 723  
ResetTime, 362  
ResetUpdateTime, 1756  
Resizable, 118  
resize, 973  
Resize, 923  
Resolve, 1763  
Resort, 329  
Resource file, 6  
Restore, 1050  
RestoreState, 1479  
Resume, 1022, 1232, 1544, 1677  
ResumePropagation, 575  
reverse, 973  
RGBtoHSV, 922  
RGBValue, 922  
RGBValue::RGBValue, 922  
right, 966  
Right, 118, 1457, 1572  
RightClick, 334  
RightDClick, 1128  
RightDockable, 118  
RightDown, 1128  
RightIsDown, 1128  
RightOf, 940  
RightUp, 1128  
Rmdir, 625  
RmDir, 697  
RmFile, 697  
RollbackTrans, 401  
Rotate, 725, 728, 924  
Rotate sample, 2037  
Rotate90, 924  
RotateHue, 924  
Row, 118  
RowAppended, 324, 327  
RowChanged, 324, 327  
RowDeleted, 324, 328  
RowInserted, 324, 328  
RowPrepended, 324, 328

RowsReordered, 324, 328  
RTTI, 8  
Run, 1677  
RunWizard, 1862

---

## —S—

SafeSet, 118  
SameAs, 625, 940  
Save, 551, 598, 608, 1870  
SaveAs, 551  
SaveEditControlValue, 758  
SaveFile, 133, 148, 924, 925, 932, 1313, 1340, 1352, 1648  
SaveObject, 551  
SavePanelInfo, 103  
SavePerspective, 103  
SaveState, 1479  
Scale, 725, 728, 926, 1442  
ScanParam, 868  
ScreenToClient, 1832  
Scroll, 1421  
Scroll subwindow sample, 2037  
ScrollIntoView, 1340  
ScrollLines, 1794, 1833  
ScrollList, 994  
ScrollPages, 1795, 1833  
ScrollTo, 1743  
ScrollToLine, 1795  
ScrollWindow, 1834  
SearchEventTable, 581  
Searching and replacing, 1556  
Searching and sorting, 75  
Second, 1689  
Seconds, 1689  
Seek, 588, 596, 1064, 1549, 1617  
SeekEnd, 588, 596  
SeekI, 943  
SeekO, 162, 1157  
Select, 32, 291, 1010, 1788, 2175  
SelectAll, 759, 878, 1341, 1789  
SelectBlock, 759  
SelectCol, 759  
SelectDocumentPath, 534  
SelectDocumentType, 534  
selected, 139  
Selecting, 785, 787  
Selection functions, 738  
SelectionToDeviceRect, 759  
SelectionToText, 878  
SelectItem, 1743  
SelectLine, 878  
SelectNone, 1341  
SelectObject, 1070  
SelectObjectAsSource, 1070  
SelectRange, 1789  
SelectRow, 759  
SelectViewType, 534  
SelectWord, 878  
SendCommand, 696  
SendIdleEvents, 53  
SendSizeEvent, 690

- SendTo, 308
- Service, 945, 947
- Set, 22, 217, 275, 361, 362, 366, 678, 728, 819, 941, 979, 1280, 1442, 1883
- Set3StateValue, 183
- SetAcceleratorTable, 1834
- SetAccessible, 1834
- SetAccessTime, 1600
- SetActiveTarget, 1022
- SetActualColor, 887
- SetAffirmativeButton, 1543
- SetAffirmativeId, 503
- SetAlign, 827, 887, 1005
- SetAlignHor, 827
- SetAlignment, 316, 770, 1240, 1292, 1397, 1622
- SetAlignVer, 828
- SetAllowSymbols, 669
- SetAlpha, 927
- SetAndShowDefaultStyle, 1341
- SetAnimation, 44
- SetApplyOnSelection, 1381
- SetAppName, 53
- SetArchitecture, 1192
- SetArtists, 19
- SetArtProvider, 103, 108
- SetAscending, 329
- SetAscii, 697
- SetAttr, 173, 796
- SetAttributes, 1356
- SetAttrProvider, 796
- SetAuthStr, 413
- SetAutoLayout, 1834
- SetAxisOrientation, 472
- SetBackground, 472
- SetBackgroundColour, 174, 770, 828, 995, 1005, 1008, 1105, 1292, 1622, 1835
- SetBackgroundMode, 472
- SetBackgroundStyle, 1836
- SetBaseld, 608
- SetBaseStyle, 1377
- SetBasicStyle, 1313, 1341
- SetBellOnError, 1768
- SetBezelFace, 704
- SetBin, 1203
- SetBinary, 697
- SetBitmap, 145, 316, 1105, 1529, 1862
- SetBitmapDisabled, 143
- SetBitmapFocus, 143
- SetBitmapHover, 144
- SetBitmapLabel, 144
- SetBitmapResource, 1708
- SetBitmaps, 1105
- SetBitmapSelected, 144
- SetBlinkTime, 179
- SetBlockingFactor, 1606
- SetBookCtrl, 1235
- SetBorder, 175, 829, 1461, 1862
- SetBorderColour, 175
- SetBorders, 879
- SetBoundingRect, 1692
- SetBrush, 472, 726
- SetBufferIO, 1550
- SetBufSize, 1067
- SetBulletFont, 1293, 1630
- SetBulletName, 1293, 1630
- SetBulletNumber, 1293, 1630
- SetBulletStyle, 1293, 1631
- SetBulletText, 1293, 1631
- SetButtonBitmaps, 242
- SetButtonPosition, 242
- SetButtonsImageList, 1743
- SetCancelButton, 1543
- SetCanvas, 1223
- SetCanVeto, 201
- SetCap, 1180
- SetCapture, 953
- SetCaret, 1836
- SetCaretPosition, 1341
- SetCellAlignment, 759
- SetCellBackgroundColour, 760
- SetCellEditor, 760
- SetCellFont, 760
- SetCellRenderer, 760
- SetCellTextColour, 760
- SetCellValue, 760
- SetChar, 1572
- SetCharacter, 1349
- SetCharacterStyleName, 1294, 1631
- SetCheckpoint, 486
- SetCheckPrevious, 486
- SetChildren, 1876
- SetChooseFull, 219
- SetChosenFont, 670
- SetClassName, 54
- SetClearExt, 615
- SetClientData, 195, 255, 291, 582, 1479
- SetClientObject, 195, 255, 292, 582
- SetClientSize, 1836
- SetClipboard, 1108
- SetClippingRegion, 473
- SetCmdLine, 206, 207
- SetCol, 706, 782
- SetColAttr, 761, 797
- SetColDefs, 443
- SetColFormatBool, 761
- SetColFormatCustom, 761
- SetColFormatFloat, 761
- SetColFormatNumber, 761
- SetColLabelAlignment, 762
- SetColLabelSize, 762
- SetColLabelValue, 762, 796
- SetCollapsed, 214
- SetCollate, 1203, 1210
- SetColMinimalAcceptableWidth, 762
- SetColMinimalWidth, 762
- SetColNull, 446
- SetColor, 96
- SetColour, 96, 154, 219, 224, 225, 670, 719, 1180, 1203
- SetColPos, 762
- SetCols, 798
- SetColSize, 763
- SetColspan, 708
- SetColumn, 322, 995, 1005

- SetColumnImage, 1010
- SetColumns, 189
- SetColumnWidth, 995
- SetCommandProcessor, 551
- SetComment, 1891, 1899
- SetConcurrency, 1677
- SetConfig, 678
- SetConfigPath, 678
- SetConnectCommand, 509
- SetConstraints, 1837
- SetContainer, 887
- SetContainingSizer, 1837
- SetContent, 1876
- SetControl, 782
- SetController, 845, 847
- SetConv, 59
- SetCopyright, 19
- SetCount, 82
- SetCountry, 359
- SetCreateTime, 1600
- SetCurrent, 719, 721
- SetCurrentPage, 1224
- SetCursor, 445, 561, 1431, 1837
- SetCustomColour, 219
- SetCustomPaintWidth, 243
- SetCwd, 626
- SetDashes, 1181
- SetData, 198, 304, 314, 337, 560, 927, 1006, 1135, 1582, 1775
- SetDataLen, 1068
- SetDataObject, 564
- SetDataViewColumn, 322
- SetDate, 171, 340
- SetDateTime, 63
- SetDay, 362
- SetDays, 347
- SetDC, 832, 887
- SetDebugErrorMessages, 401
- SetDebugMode, 486
- SetDefAttr, 772
- SetDefault, 166
- SetDefaultCellAlignment, 763
- SetDefaultCellBackgroundColour, 763
- SetDefaultCellFont, 763
- SetDefaultCellTextColour, 763
- SetDefaultColSize, 764
- SetDefaultDir, 413
- SetDefaultEditor, 763
- SetDefaultEncoding, 663
- SetDefaultExtension, 545
- SetDefaultInfo, 1162
- SetDefaultItem, 1717
- SetDefaultMinMargins, 1163
- SetDefaultPath, 713
- SetDefaultProxy, 1765
- SetDefaultRenderer, 764
- SetDefaultRowSize, 764
- SetDefaults, 1442
- SetDefaultSize, 1397
- SetDefaultStyle, 1313, 1341, 1648
- SetDefaultStyleToCursorStyle, 1341
- SetDelay, 1712
- SetDelayedLayoutThreshold, 1342
- SetDepth, 134, 900
- SetDesc, 208
- SetDescription, 20, 413, 545, 1377, 1390
- SetDevelopers, 20
- SetDeviceOrigin, 473
- SetDevMajor, 1601
- SetDevMinor, 1601
- SetDialogParent, 677
- SetDialogTitle, 678
- SetDimension, 1454, 1461
- SetDirectory, 545, 603
- SetDispatchPtr, 123
- SetDockSizeConstraint, 104
- SetDocument, 526, 537, 1783
- SetDocumentManager, 545
- SetDocumentName, 552
- SetDocumentTemplate, 552
- SetDocWriters, 20
- SetDomain, 1883
- SetDropTarget, 1838
- SetDsn, 413
- SetDuplex, 1204
- SetEditable, 1342, 1649
- SetEditCanceled, 322
- SetEditMenu, 258
- SetEditor, 770
- SetEmptyCellSize, 774
- SetEmptyExt, 626
- SetEncoding, 1352, 1870
- SetEnd, 1375
- SetEndianness, 1192
- SetEscapeld, 503
- SetEventHandler, 1479, 1838
- SetEventObject, 575
- SetEventType, 575
- SetEvtHandlerEnabled, 583
- SetExcludes, 1670
- SetExitOnFrameDelete, 54
- SetExpandEnvVars, 275
- SetExt, 626
- SetExtension, 149, 933, 1352
- SetExternalAttributes, 1892
- SetExtra, 1892
- SetExtraLong, 255
- SetExtraStyle, 1839
- SetFaceName, 663
- SetFamily, 664
- SetFieldsCount, 1539
- SetFile, 486
- SetFileCounter, 1364
- SetFileEncoding, 1870
- SetFileFilter, 545
- SetFilename, 552, 603, 1342
- SetFileType, 413
- SetFilter, 713
- SetFilterIndex, 603, 713
- SetFindString, 651
- SetFirstItem, 980
- SetFlag, 119, 1461
- SetFlags, 98, 104, 168, 545, 650, 1240, 1294, 1349, 1352, 1386, 1480, 1622, 1883

- SetFlexibleDirection, 654
- SetFocus, 1172, 1840
- SetFocusFromKbd, 1840
- SetFocusIgnoringChildren, 1172
- SetFont, 96, 108, 175, 473, 682, 726, 770, 1006, 1008, 1105, 1342, 1361, 1623, 1840
- SetFontBold, 887
- SetFontFace, 887
- SetFontFaceName, 1294
- SetFontFixed, 887
- SetFontItalic, 888
- SetFontName, 1589
- SetFonts, 832, 836, 864, 879, 888
- SetFontSize, 888, 1295
- SetFontSizeMapping, 1364
- SetFontStyle, 1295
- SetFontUnderlined, 888, 1295
- SetFontWeight, 1295
- SetFooter, 836, 864
- SetFooterText, 1361, 1370
- SetForce, 201
- SetForegroundColour, 1840
- SetFormat, 336, 343
- SetFormattingDialogFactory, 1356
- SetFrame, 1224, 1783
- SetFrameParameters, 814
- SetFromClause, 446
- SetFromPage, 1210
- SetFromUnicode, 1589
- SetFS, 863
- SetFullName, 626
- SetGridCursor, 764
- SetGridLineColour, 764
- SetGroupId, 1601
- SetGroupName, 1601
- SetHandlerFlags, 1342
- SetHeader, 836, 864, 890
- SetHeaderColours, 171
- SetHeaderFooterData, 1370, 1373
- SetHeaderFooterFont, 1370
- SetHeaderFooterTextColour, 1370
- SetHeaderText, 1361, 1371
- SetHeight, 134, 900, 1257, 1443
- SetHelp, 1105
- SetHelpController, 815
- SetHelpString, 1086, 1096
- SetHelpText, 1841
- SetHenv, 414
- SetHGap, 798
- SetHighlightColours, 172
- SetHoliday, 173, 175
- SetHolidayColours, 172
- SetHour, 362
- SetHoverColour, 893
- SetHtmlFile, 865
- SetHtmlText, 832, 865
- SetIcon, 20, 503, 1529, 1609, 1717
- SetIcons, 503, 1717
- SetId, 306, 575, 824, 1006, 1752, 1841
- SetImage, 1006
- SetImageList, 995, 1143, 1356, 1726, 1744
- SetInactiveBitmap, 45
- SetIncludes, 1670
- SetIndent, 829, 1744
- SetInitialBestSize, 1838
- SetInitialFont, 670
- SetInitialSize, 1836
- SetInitSize, 1461
- SetInnerSizer, 1235
- SetInputEncoding, 888
- SetInsertionPoint, 231, 243, 1342, 1649
- SetInsertionPointEnd, 231, 243, 1342, 1649
- SetInstallPrefix, 1527
- SetInstance, 54
- SetInt, 255
- SetInternalMargin, 1184
- SetIntPosition, 1551
- SetIsDir, 64
- SetIsReadOnly, 64
- SetIsText, 1894
- SetItem, 995, 996
- SetItemBackgroundColour, 997, 1744
- SetItemBitmap, 139
- SetItemBold, 1744
- SetItemCount, 997, 1789
- SetItemData, 997, 1744
- SetItemDropHighlight, 1745
- SetItemFont, 997, 1745
- SetItemHasChildren, 1745
- SetItemHelpText, 1247
- SetItemImage, 997, 998, 1745
- SetItemLabel, 1105
- SetItemMinSize, 1454
- SetItemPosition, 774, 998
- SetItemPtrData, 998
- SetItemSelectedImage, 1745
- SetItemSpan, 775
- SetItemState, 998
- SetItemText, 998, 1745
- SetItemTextColour, 998, 1746
- SetItemToolTip, 1248
- SetJoin, 1181
- SetLabel, 167, 286, 1087, 1096, 1247, 1536, 1841
- SetLabelBackgroundColour, 764
- SetLabelFont, 764
- SetLabelTextColour, 765
- SetLabelTop, 1097
- SetLastDirectory, 535
- SetLayoutDirection, 474
- SetLeftIndent, 1295, 1623
- SetLeftMenu, 1717
- SetLength, 1581
- SetLevel, 487, 1899
- SetLevelAttributes, 1367
- SetLicence, 20
- SetLicense, 20
- SetLineCount, 1795
- SetLineSize, 1469
- SetLineSpacing, 1295, 1631
- SetLink, 824, 888
- SetLinkClicked, 883
- SetLinkColor, 888
- SetLinkName, 1602
- SetListStyle, 1313, 1314, 1343

- SetListStyleName, 1296, 1632
- SetLocal, 1481
- SetLocalExtra, 1893
- SetLog, 1027
- SetLoggingOff, 201
- SetLogicalFunction, 474
- SetLogicalOrigin, 1220
- SetLogLevel, 1024
- SetLogo, 208
- SetManagedWindow, 104
- SetMapMode, 474
- SetMarginBottomRight, 1163
- SetMargins, 765, 865, 1362, 1373, 1708, 1789
- SetMarginTopLeft, 1163
- SetMarginWidth, 1105
- SetMask, 134, 928, 1006
- SetMaskColour, 928
- SetMaskFromImage, 928
- SetMaxDocsOpen, 535
- SetMaximumSizeX, 1401
- SetMaximumSizeY, 1401
- SetMaxLength, 1649
- SetMaxPage, 1210
- SetMaxSize, 1718, 1842
- SetMeasuringFont, 98, 109
- SetMenu, 1105, 1434
- SetMenuBar, 690
- SetMenuLabel, 1097
- SetMenuStrings, 258
- SetMessage, 515, 604
- SetMethod, 1893
- SetMetric, 96
- SetMillisecond, 363
- SetMimeType, 933
- SetMinHeight, 830, 1540
- SetMinimumPaneSize, 1515
- SetMinimumSizeX, 1401
- SetMinimumSizeY, 1401
- SetMinMarginBottomRight, 1163
- SetMinMarginTopLeft, 1163
- SetMinPage, 1210
- SetMinSize, 1454, 1718, 1842
- SetMinute, 363
- SetModal, 504
- SetMode, 905, 1602, 1667, 1756, 1893
- SetModel, 323
- SetModified, 1650
- SetMonth, 362
- SetMonths, 347
- SetMovementThreshold, 953
- SetName, 21, 63, 149, 626, 932, 1353, 1377, 1390, 1842, 1876, 1878
- SetNativeFontInfo, 664
- SetNativeFontInfoUserDesc, 665
- SetNegativeButton, 1543
- SetNewStyleSheet, 1350
- SetNext, 825, 1866, 1876, 1878
- SetNextHandler, 583
- SetNextStyle, 1368
- SetNoCopies, 1204, 1210
- SetNonFlexibleGrowMode, 654
- SetNormalColour, 893
- SetNormalFont, 98, 108
- SetNormalTextFontName, 1589
- SetNotifier, 64, 1894
- SetNotify, 1481
- SetOldSelection, 1145
- SetOldStyleSheet, 1350
- SetOnlineStatus, 508
- SetOperatingSystemId, 1192
- SetOption, 929, 1594
- SetOrCalcColumnSizes, 765
- SetOrCalcRowSizes, 765
- SetOrderByClause, 447
- SetOrientation, 1204, 1240, 1397
- SetOrigin, 817
- SetOSVersion, 1192
- SetOutlineLevel, 1296, 1632
- SetOwnBackgroundColour, 1842
- SetOwner, 324, 331, 1682
- SetOwnFont, 1843
- SetOwnForegroundColour, 1843
- SetPadding, 1143
- SetPage, 880
- SetPageBitmap, 109
- SetPageBreak, 1296, 1632
- SetPageImage, 1143, 1726
- SetPageSetupData, 1371
- SetPageSize, 1143, 1469, 1862
- SetPageText, 109, 1143, 1726
- SetPalette, 134, 475, 929, 1843
- SetPaperId, 1163, 1204
- SetPaperSize, 1163
- SetParagraphSpacingAfter, 1296, 1632
- SetParagraphSpacingBefore, 1296, 1632
- SetParagraphStyleName, 1296, 1632
- SetParameters, 776, 779, 780, 781, 790
- SetParent, 825, 1876
- SetParentResource, 1887
- SetParentWindow, 814, 837, 1371
- SetParser, 870
- SetPassive, 698
- SetPassword, 414, 698, 1237
- SetPath, 276, 515, 518, 604, 632, 633, 713
- SetPen, 475, 726
- SetPickerCtrlGrowable, 1185
- SetPickerCtrlProportion, 1184
- SetPlaybackRate, 1064
- SetPointSize, 666
- SetPopupAnchor, 243
- SetPopupControl, 243
- SetPopupExtents, 243
- SetPopupMaxHeight, 244
- SetPopupMinWidth, 244
- SetPortId, 1193
- SetPos, 707, 825
- SetPosition, 282, 817, 1350, 1504
- SetPrecision, 790
- SetPrev, 1866
- SetPreviewRect, 1371
- SetPreviousHandler, 583
- SetPrintData, 1163, 1211, 1371
- SetPrinterName, 1205
- SetPrintout, 1224



- SetPrintToFile, 1211
- SetPriority, 1677
- SetProperties, 1876
- SetProportion, 1462
- SetProxy, 1766
- SetQuality, 1206
- SetQueryTimeout, 448
- SetQuickBestSize, 1746
- SetRange, 343, 670, 704, 1350, 1375, 1469, 1499, 1502
- SetRatio, 1462
- SetReadOnly, 765, 770
- SetRecordDefaults, 276
- SetRect, 168
- SetRedoAccelerator, 258
- SetRefData, 1150
- SetRelatedFrame, 880
- SetRelatedStatusBar, 880
- SetRenderer, 770, 1314
- SetRepetitionCounting, 1024
- SetReplaceString, 651
- SetRequestedLength, 1240
- SetResolution, 1195
- SetRestartNumbering, 1386
- SetReturnCode, 504
- SetRGB, 929
- SetRichTextBuffer, 1373
- SetRichTextCtrl, 1378, 1381, 1383, 1386
- SetRightIndent, 1296, 1623
- SetRightMenu, 1719
- SetRoot, 1870
- SetRow, 323, 706, 782
- SetRowAttr, 765, 796
- SetRowLabelAlignment, 765
- SetRowLabelSize, 766
- SetRowLabelValue, 766, 795
- SetRowMinimalAcceptableHeight, 766
- SetRowMinimalHeight, 766
- SetRows, 798
- SetRowSize, 766
- SetRowspan, 708
- SetSashBorder, 1401
- SetSashGravity, 1514
- SetSashPosition, 1508, 1515
- SetSashSize, 1515
- SetSashVisible, 1401
- SetScrollbar, 1413, 1843
- SetScrollbars, 1422
- SetScrollLineX, 767
- SetScrollLineY, 767
- SetScrollPos, 1844
- SetScrollRate, 1423
- SetSecond, 363
- SetSelectedFont, 98, 108, 680
- SetSelection, 109, 231, 244, 292, 321, 1143, 1145, 1211, 1247, 1343, 1437, 1470, 1502, 1650, 1726, 1789
- SetSelectionBackground, 767, 1790
- SetSelectionForeground, 767
- SetSelectionMode, 767
- SetSelectionRange, 321, 1343
- SetSelections, 321, 1131
- SetSetupDialog, 1211
- SetShadowWidth, 704
- SetShape, 1719
- SetSheetStyle, 1235, 1358
- SetShowHelp, 670
- SetShowOnFirstPage, 1362, 1371
- SetShowToolTips, 1386
- SetSingleStyle, 998
- SetSize, 63, 180, 777, 832, 1241, 1257, 1602, 1845
- SetSizeHints, 1454, 1718
- SetSizer, 1462, 1846
- SetSizerAndFit, 1847
- SetSizingInfo, 98
- SetSortable, 317
- SetSortOrder, 316
- SetSpacer, 1462
- SetSpan, 707
- SetSplitMode, 1516
- SetSqlLogging, 402
- SetStandardError, 487
- SetStart, 1375
- SetState, 1006
- SetStatImageList, 1746
- SetStateMask, 1006
- SetStatusBar, 690
- SetStatusBarPane, 691
- SetStatusStyles, 1541
- SetStatusText, 691, 1540
- SetStatusWidths, 691, 1540
- SetStipple, 154, 1181
- SetStream, 487
- SetString, 255, 292, 1586
- SetStringSelection, 293, 1248
- SetStringSeparators, 1665
- SetStringValue, 248
- SetStyle, 155, 666, 1181, 1314, 1343, 1357, 1377, 1650, 1670
- SetStyleDefinition, 1357
- SetStyleEx, 1344
- SetStyleSheet, 1315, 1344, 1379, 1381, 1383, 1387
- SetStyleType, 1381, 1383
- SetSubMenu, 1106
- SetSwitchChars, 207
- SetSymbol, 1589
- SetSystemMadeBy, 1894
- SetTabCtrlHeight, 109
- SetTable, 768
- SetTabs, 1297, 1623
- SetTargetWindow, 1423
- SetTempDir, 842, 844, 1365
- SetTemporaryImageLocations, 1365
- SetText, 245, 1006, 1106, 1362, 1654, 1757
- SetTextBackground, 475
- SetTextColour, 174, 770, 999, 1007, 1008, 1106, 1297, 1362, 1623
- SetTextCtrlGrowable, 1185
- SetTextCtrlProportion, 1184
- SetTextEffectFlags, 1297, 1632
- SetTextEffects, 1297, 1633
- SetTextForeground, 475

- SetTextIndent, 245
- SetThemeEnabled, 1847
- SetThumbLength, 1470
- SetThumbPosition, 1413
- SetTick, 1470
- SetTickFreq, 1471
- SetTimeout, 1482
- SetTimes, 627
- SetTimestamp, 575, 1024
- SetTip, 1713
- SetTipWindowPtr, 1692
- SetTitle, 317, 552, 1087, 1371, 1719
- SetTitleFormat, 842, 845, 847, 850
- SetToCurrent, 361
- SetToLastMonthDay, 372
- SetToLastWeekDay, 372
- SetToNextWeekDay, 371
- SetToolBar, 692, 1057
- SetToolBitmapSize, 1708
- SetToolClientData, 1709
- SetToolDisabledBitmap, 1709
- SetToolkitVersion, 1193
- SetToolLongHelp, 1709
- SetToolNormalBitmap, 1710
- SetToolPacking, 1710
- SetToolSeparation, 1711
- SetToolShortHelp, 1710
- SetToolTip, 1751, 1847, 1848
- SetToPage, 1211
- SetToPrevWeekDay, 371
- SetTopWindow, 54
- SetToWeekDay, 371
- SetToWeekDayInSameWeek, 370
- SetToWeekOfYear, 372
- SetToYearDay, 373
- SetTraceMask, 1025
- SetTransferMode, 698
- SetTransform, 726
- SetTranslators, 21
- SetTransparent, 1720
- SetType, 149, 306, 933, 1353, 1877
- SetTypeFlag, 1602
- SetUid, 414
- SetUmask, 598
- SetUnderlined, 667
- SetUndoAccelerator, 258
- SetUnicodeMode, 1589
- SetUniformBitmapSize, 109
- Setup, 1213
- SetUpdateInterval, 1757
- SetupScrollbars, 1345
- SetupWindow, 1888
- SetURL, 894, 1298, 1633, 1766, 1767
- SetUseBestVisual, 55
- SetUser, 698, 1237
- SetUserId, 1601
- SetUserId, 414
- SetUserName, 1601
- SetUserScale, 476
- SetValidator, 1848
- SetValue, 183, 231, 245, 323, 328, 331, 343, 704, 793, 1251, 1276, 1345, 1471, 1500, 1502, 1651, 1657, 1694, 1878
- SetValueAsBool, 794
- SetValueAsCustom, 794
- SetValueAsDouble, 794
- SetValueAsLong, 794
- SetValueWithEvent, 245
- SetVendorName, 55
- SetVerbose, 1023
- SetVersion, 21, 1870
- SetVGap, 798
- SetView, 526, 537, 794
- SetViewer, 814
- SetViewName, 1783
- SetVirtualSize, 1848
- SetVirtualSizeHints, 1455, 1848
- SetVisible, 1353
- SetVisited, 893
- SetVisitedColour, 893
- SetVolume, 627, 1065
- SetWebSite, 21
- SetWeekDay, 177
- SetWeeks, 347
- SetWeight, 667
- SetWellKnownHost, 509
- SetWhereClause, 448
- SetWidth, 135, 791, 900, 1007, 1181, 1258, 1443
- SetWidthFloat, 831
- SetWildcard, 604
- SetWindow, 32, 1462, 1768
- SetWindowMenu, 1057
- SetWindowStyle, 1746, 1849
- SetWindowStyleFlag, 999, 1849
- SetWindowVariant, 1849
- SetX, 1258
- SetY, 1258
- SetYear, 362
- SetYears, 347
- SetZoom, 1224
- SetZoomControl, 1199
- Shaped, 1458
- ShiftDown, 785, 787, 789, 961, 1128
- ShouldInheritColours, 1849
- ShouldPreventAppExit, 1720
- ShouldPropagate, 575
- Show, 119, 180, 494, 504, 556, 777, 1031, 1248, 1455, 1462, 1755, 1850
- ShowAssertDialog, 58
- ShowCancelButton, 1435
- ShowCellEditControl, 768
- ShowFullScreen, 1720
- ShowHelp, 819, 1358
- ShowHelpAtPoint, 819
- ShowHidden, 713
- ShowHint, 104
- ShowModal, 222, 505, 515, 604, 671, 1107, 1131, 1159, 1207, 1437, 1657
- ShowPlayerControls, 1065
- ShowPopup, 245
- ShowPosition, 1345, 1652
- ShowSearchButton, 1435
- ShowWindowList, 99
- ShowWindowMenu, 110

Shrink, 82, 87, 1572  
 Signal, 261  
 Simplify the problem, 13  
 size, 802, 807, 973  
 Size, 927  
 Skip, 575  
 Sleep, 1677  
 Socket state, 1474  
 Sockets sample, 2037  
 Some advice about using wxString, 2049  
 Sort, 82, 87, 970  
 SortChildren, 1746  
 SortItems, 999  
 Sound sample, 2038  
 Source level compatibility, 2231  
 splice, 973  
 Split, 109  
 SplitHorizontally, 1516  
 SplitPath, 627  
 SplitVertically, 1517  
 SplitVolume, 627  
 SQLColumnName, 402  
 SQLTableName, 403  
 Start, 1544, 1682  
 StartAdvise, 280, 482, 1614  
 StartDoc, 476  
 StartDrag, 334  
 StartDrawingOnTop, 1407  
 Starting to use wxRichTextCtrl, 2193  
 StartingClick, 778  
 StartingKey, 778  
 StartPage, 476  
 StartsWith, 1572  
 Statbar sample, 2038  
 Static functions, 264, 352, 1684  
 std::string compatibility functions, 1558  
 Stop, 45, 1065, 1496, 1682  
 StopAdvise, 280, 482, 1614  
 StopParsing, 863  
 StopPropagation, 576  
 Strategies for exceptions handling, 2088  
 Stream, 1552  
 String length, 1554  
 Strip, 1572  
 StrokeLine, 726  
 StrokeLines, 726, 727  
 StrokePath, 725  
 Submit, 258  
 SubmitAction, 1315  
 SubString, 1573  
 Substring extraction, 1555  
 Subtract, 347, 368, 1267, 1689  
 Supported bitmap file formats, 2118  
 Supported languages, 1012  
 SuppressingUndo, 1315, 1345  
 Suspend, 1022  
 swap, 1403, 1406  
 Swap, 1375  
 SwapBuffers, 719  
 Sync, 162

## —T—

TableExists, 403  
 TablePrivileges, 404  
 Tag Handlers, 2183  
 Tags supported by wxHTML, 2186  
 TakeData, 304  
 Technicalities, 2089  
 Tell, 588, 597, 1066, 1549, 1617  
 TellI, 944  
 TellO, 1157  
 Templates, 8  
 TestDestroy, 1677  
 Testing for WinCE SDKs, 2236  
 Tests, 1684  
 Tests of existence, 266  
 Text sample, 2038  
 Thaw, 1345, 1850  
 The data provider (source) duties, 2152  
 The data receiver (target) duties, 2152  
 The idea behind sizers, 2098  
 The version numbering scheme, 2231  
 Themed borders on Windows, 2235  
 This, 1678  
 Thread sample, 2039  
 Tile, 1058  
 Time, 1544  
 Time zone and DST support, 355  
 Time zone considerations, 2055  
 To8BitData, 1573  
 ToAscii, 1573  
 Today, 360  
 ToDouble, 1034, 1573  
 Toggle, 1747, 1790  
 ToggleItemSelection, 1747  
 ToggleTool, 1711  
 ToggleWindowStyle, 1850  
 ToInternal, 1375  
 ToLong, 1035, 1574  
 ToLongLong, 1574  
 Toolbar sample, 2039  
 ToolbarPane, 119  
 top, 966  
 Top, 119  
 TopDockable, 119  
 ToString, 1035  
 ToText, 881  
 ToTimezone, 374  
 Touch, 628  
 ToUF8, 1575  
 ToULong, 1574  
 ToULongLong, 1575  
 ToUTC, 374  
 ToWChar, 1043  
 TransferDataFromWindow, 1851  
 TransferDataToWindow, 1851  
 TransferFromWindow, 715, 1670  
 TransferToWindow, 715, 1670, 1769  
 Transform, 732  
 TransformDistance, 729  
 TransformPoint, 729  
 Translate, 725, 728

TranslateSqlState, 405  
Traverse, 513  
Treectrl sample, 2039  
Trim, 1575  
TripleBorder, 1458  
Truncate, 1575  
TryLock, 1133  
TryWait, 1428  
Tuning wxString for your application, 2051  
Type of NULL, 8  
Types of wxThreads, 1671

## —U—

Unconstrained, 940  
Undo, 232, 245, 250, 259, 1345, 1652  
Unescape, 1763  
UngetAppendBuf, 1068  
Ungetch, 944  
UnGetNativePath, 732  
UngetWriteBuf, 1068, 1575  
Unicode and ANSI modes, 2057  
Unicode and the outside world, 2059  
Unicode support in wxWidgets, 2058  
Unicode-related compilation settings, 2059  
UnInit, 104  
Union, 1258, 1268  
Unload, 567, 1883  
UnloadLibrary, 525  
Unlock, 1133  
UNow, 360  
Unread, 1484  
UnRef, 1150  
UnregisterHotKey, 1851  
Unselect, 321, 1747  
UnselectAll, 1747  
UnselectItem, 1747  
UnsetNotifier, 64, 1894  
UnShare, 1151  
Unsplit, 1518  
Update, 105, 449, 1232, 1852, 2175  
UpdateAllViews, 552  
UpdateAttrCols, 796  
UpdateAttrRows, 796  
UpdateBackingFromWindow, 556  
UpdateDisplay, 1357  
UpdateRow, 455  
UpdateSize, 1518  
UpdateStyles, 1379, 1381, 1383  
UpdateUI, 1087  
UpdateWhere, 450  
UpdateWindowUI, 1852  
Upper, 1575  
UpperCase, 1576  
URLToFileName, 636  
Usage, 209  
Use a debugger, 13  
Use ASSERT, 12  
Use logging functions, 13  
Use relative positioning or constraints, 12  
Use the wxWidgets debugging facilities, 13  
Use wxString in preference to character arrays, 12

Use wxWidgets resource files, 12  
UseAltPopupWindow, 245  
UseConfig, 843, 850  
UseMenu, 608  
UseNativeDecorations, 1721  
UseNativeDecorationsByDefault, 1721  
UseNormalFont, 1589  
UsePrimarySelection, 198  
UseStringValue, 775  
Using binary resource files, 2107  
Using embedded resources, 2108  
Using the toolbar library, 2139  
Using wxPython, 2200  
utf8\_str, 1576

## —V—

Validate, 331, 1670, 1769, 1853  
ValidateRow, 455  
ValidHost, 193, 478, 1610  
ValueChanged, 324, 328  
version, 1281  
Veto, 201, 1147, 1196  
Video size, 1059  
virtual Cleared, 326

## —W—

Wait, 262, 1429, 1484, 1485, 1678  
WaitForAccept, 1494  
WaitForRead, 1486  
WaitForWrite, 1486  
WaitOnConnect, 1489  
WaitTimeout, 262, 1429  
Walk, 1522  
WalkFromException, 1522  
WarpPointer, 1853  
wc\_str, 1576  
WC2MB, 297, 1040, 1043, 1044, 1045, 1046, 1047  
wchar\_str, 1576  
Week, 347, 1689  
Weeks, 348, 1689  
What is Unicode?, 2056  
What is wxPython?, 2198  
Where, 450  
Where to go for help, 2210  
Why use wxPython?, 2199  
Why you shouldn't care about it, 2046  
Widgets sample, 2040  
width, 966, 1253  
widthSash, 1519  
Window, 119  
Window identifiers, 2082  
Window layout examples, 2096  
Window sizing in wxWinCE, 2237  
WindowToClientSize, 1853  
Wizard sample, 2040  
WordLeft, 1345  
WordRight, 1345  
Wrap, 1536  
Write, 276, 588, 589, 597, 1157, 1158, 1487, 1548,

- 
- 1617, 1662, 1779
  - Write16, 338, 1667
  - Write32, 338, 1667
  - Write64, 339
  - Write8, 338, 1667
  - WriteCustomization, 843, 846, 847, 850, 881
  - WriteDouble, 339, 1667
  - WriteImage, 1345, 1346
  - WriteMsg, 1487
  - WriteSqlLog, 406
  - WriteString, 339, 1667
  - WriteText, 1346, 1652
  - Writing values into the string, 1556
  - WX\_APPEND\_ARRAY, 77
  - WX\_CLEAR\_ARRAY, 78
  - wx\_const\_cast, 1970
  - WX\_DECLARE\_EXPORTED\_OBJARRAY, 76
  - WX\_DECLARE\_OBJARRAY, 76
  - WX\_DECLARE\_USER\_EXPORTED\_OBJARRA  
Y, 76
  - WX\_DEFINE\_ARRAY, 75
  - WX\_DEFINE\_EXPORTED\_ARRAY, 75
  - WX\_DEFINE\_EXPORTED\_OBJARRAY, 77
  - WX\_DEFINE\_OBJARRAY, 77
  - WX\_DEFINE\_SORTED\_ARRAY, 76
  - WX\_DEFINE\_SORTED\_EXPORTED\_ARRAY,  
76
  - WX\_DEFINE\_SORTED\_USER\_EXPORTED\_AR  
RAY, 76
  - WX\_DEFINE\_USER\_EXPORTED\_ARRAY, 75
  - WX\_DEFINE\_USER\_EXPORTED\_OBJARRAY,  
77
  - WX\_GL\_AUX\_BUFFERS, 716
  - WX\_GL\_BUFFER\_SIZE, 716
  - WX\_GL\_DEPTH\_SIZE, 717
  - WX\_GL\_DOUBLEBUFFER, 716
  - WX\_GL\_LEVEL, 716
  - WX\_GL\_MIN\_ACCUM\_ALPHA, 717
  - WX\_GL\_MIN\_ACCUM\_BLUE, 717
  - WX\_GL\_MIN\_ACCUM\_GREEN, 717
  - WX\_GL\_MIN\_ACCUM\_RED, 717
  - WX\_GL\_MIN\_ALPHA, 717
  - WX\_GL\_MIN\_BLUE, 717
  - WX\_GL\_MIN\_GREEN, 717
  - WX\_GL\_MIN\_RED, 716
  - WX\_GL\_RGBA, 716
  - WX\_GL\_STENCIL\_SIZE, 717
  - WX\_GL\_STEREO, 716
  - WX\_PREPEND\_ARRAY, 78
  - wx\_reinterpret\_cast, 1970
  - wx\_static\_cast, 1970
  - wx\_truncate\_cast, 1971
  - wxAboutBox, 1934
  - wxAboutDialogInfo, 18
  - wxAboutDialogInfo::AddArtist, 18
  - wxAboutDialogInfo::AddDeveloper, 19
  - wxAboutDialogInfo::AddDocWriter, 19
  - wxAboutDialogInfo::AddTranslator, 19
  - wxAboutDialogInfo::SetArtists, 19
  - wxAboutDialogInfo::SetCopyright, 19
  - wxAboutDialogInfo::SetDescription, 20
  - wxAboutDialogInfo::SetDevelopers, 20
  - wxAboutDialogInfo::SetDocWriters, 20
  - wxAboutDialogInfo::SetIcon, 20
  - wxAboutDialogInfo::SetLicence, 20
  - wxAboutDialogInfo::SetLicense, 20
  - wxAboutDialogInfo::SetName, 21
  - wxAboutDialogInfo::SetTranslators, 21
  - wxAboutDialogInfo::SetVersion, 21
  - wxAboutDialogInfo::SetWebSite, 21
  - wxAboutDialogInfo::wxAboutDialogInfo, 18
  - wxAC\_DEFAULT\_STYLE, 42
  - wxAC\_NO\_AUTORESIZE, 42
  - wxAcceleratorEntry, 22
  - wxAcceleratorEntry::GetCommand, 22
  - wxAcceleratorEntry::GetFlags, 22
  - wxAcceleratorEntry::GetKeyCode, 22
  - wxAcceleratorEntry::Set, 22
  - wxAcceleratorEntry::wxAcceleratorEntry, 21
  - wxAcceleratorTable, 24
  - wxAcceleratorTable::~wxAcceleratorTable, 24
  - wxAcceleratorTable::IsOk, 24
  - wxAcceleratorTable::operator =, 24
  - wxAcceleratorTable::wxAcceleratorTable, 23
  - wxAccessible, 29
  - wxAccessible::~wxAccessible, 29
  - wxAccessible::DoDefaultAction, 29
  - wxAccessible::GetChild, 30
  - wxAccessible::GetChildCount, 30
  - wxAccessible::GetDefaultAction, 30
  - wxAccessible::GetDescription, 30
  - wxAccessible::GetFocus, 30
  - wxAccessible::GetHelpText, 30
  - wxAccessible::GetKeyboardShortcut, 30
  - wxAccessible::GetLocation, 31
  - wxAccessible::GetName, 31
  - wxAccessible::GetParent, 31
  - wxAccessible::GetRole, 31
  - wxAccessible::GetSelections, 31
  - wxAccessible::GetState, 31
  - wxAccessible::GetValue, 32
  - wxAccessible::GetWindow, 32
  - wxAccessible::HitTest, 32
  - wxAccessible::Navigate, 32
  - wxAccessible::NotifyEvent, 32
  - wxAccessible::Select, 32
  - wxAccessible::SetWindow, 32
  - wxAccessible::wxAccessible, 29
  - wxActivateEvent, 33
  - wxActivateEvent::GetActive, 34
  - wxActivateEvent::GetCapturedWindow, 1118
  - wxActivateEvent::wxActivateEvent, 33
  - wxActiveXContainer, 38
  - wxActiveXContainer::wxActiveXContainer, 38
  - wxActiveXEvent::GetDispatchId, 39
  - wxActiveXEvent::operator[], 39
  - wxActiveXEvent::ParamCount, 39
  - wxActiveXEvent::ParamName, 39
  - wxActiveXEvent::ParamType, 39
  - wxALIGN\_BOTTOM, 1447
  - wxALIGN\_CENTER wxALIGN\_CENTRE, 1447
  - wxALIGN\_CENTER\_HORIZONTAL  
wxALIGN\_CENTRE\_HORIZONTAL, 1447
  - wxALIGN\_CENTER\_VERTICAL
-

- wxALIGN\_CENTRE\_VERTICAL, 1447
- wxALIGN\_CENTRE, 1534
- wxALIGN\_LEFT, 1447, 1534
- wxALIGN\_RIGHT, 180, 1447, 1534
- wxALIGN\_TOP, 1447
- wxALL, 1447
- wxALWAYS\_SHOW\_SB, 1797
- wxANIHandler, 907
- wxAnimation, 40
- wxAnimation::~wxAnimation, 40
- wxAnimation::GetDelay, 40
- wxAnimation::GetFrame, 40
- wxAnimation::GetFrameCount, 40
- wxAnimation::GetSize, 41
- wxAnimation::IsOk, 41
- wxAnimation::Load, 41
- wxAnimation::LoadFile, 41
- wxAnimation::operator =, 42
- wxAnimation::wxAnimation, 40
- wxANIMATION\_TYPE\_ANI, 41, 42
- wxANIMATION\_TYPE\_ANY, 41, 42
- wxANIMATION\_TYPE\_GIF, 41, 42
- wxAnimationCtrl, 43
- wxAnimationCtrl::Create, 43
- wxAnimationCtrl::GetAnimation, 44
- wxAnimationCtrl::GetInactiveBitmap, 44
- wxAnimationCtrl::IsPlaying, 44
- wxAnimationCtrl::LoadFile, 44
- wxAnimationCtrl::Play, 44
- wxAnimationCtrl::SetAnimation, 44
- wxAnimationCtrl::SetInactiveBitmap, 44
- wxAnimationCtrl::Stop, 45
- wxAnimationCtrl::wxAnimationCtrl, 43
- wxApp, 46
- wxApp::~wxApp, 46
- wxApp::argc, 46
- wxApp::argv, 46
- wxApp::CreateLogTarget, 46
- wxApp::CreateTraits, 46
- wxApp::Dispatch, 46
- wxApp::ExitMainLoop, 47
- wxApp::FilterEvent, 47
- wxApp::GetAppName, 47
- wxApp::GetClassName, 47
- wxApp::GetExitOnFrameDelete, 47
- wxApp::GetInstance, 48
- wxApp::GetTopWindow, 48
- wxApp::GetTraits, 48
- wxApp::GetUseBestVisual, 48
- wxApp::GetVendorName, 49
- wxApp::HandleEvent, 55
- wxApp::IsActive, 49
- wxApp::IsMainLoopRunning, 49
- wxApp::MainLoop, 49
- wxApp::OnAssertFailure, 49
- wxApp::OnCmdLineError, 50
- wxApp::OnCmdLineHelp, 50
- wxApp::OnCmdLineParsed, 50
- wxApp::OnExceptionInMainLoop, 51
- wxApp::OnExit, 51
- wxApp::OnFatalException, 51
- wxApp::OnInit, 52
- wxApp::OnInitCmdLine, 52
- wxApp::OnRun, 52
- wxApp::OnUnhandledException, 52
- wxApp::Pending, 53
- wxApp::ProcessMessage, 52
- wxApp::SendIdleEvents, 53
- wxApp::SetAppName, 53
- wxApp::SetClassName, 54
- wxApp::SetExitOnFrameDelete, 54
- wxApp::SetInstance, 54
- wxApp::SetTopWindow, 54
- wxApp::SetUseBestVisual, 55
- wxApp::SetVendorName, 55
- wxApp::wxApp, 46
- wxApp::Yield, 56
- wxAppTraits::CreateFontMapper, 57
- wxAppTraits::CreateLogTarget, 57
- wxAppTraits::CreateMessageOutput, 57
- wxAppTraits::CreateRenderer, 57
- wxAppTraits::GetDesktopEnvironment, 57
- wxAppTraits::GetStandardPaths, 57
- wxAppTraits::GetToolkitVersion, 57
- wxAppTraits::HasStderr, 58
- wxAppTraits::IsUsingUniversalWidgets, 58
- wxAppTraits::ShowAssertDialog, 58
- wxArchiveClassFactory::CanHandle, 59
- wxArchiveClassFactory::Find, 59
- wxArchiveClassFactory::Get/SetConv, 59
- wxArchiveClassFactory::GetFirst/GetNext, 60
- wxArchiveClassFactory::GetInternalName, 60
- wxArchiveClassFactory::GetProtocol, 60
- wxArchiveClassFactory::GetProtocols, 60
- wxArchiveClassFactory::NewEntry, 61
- wxArchiveClassFactory::NewStream, 61
- wxArchiveClassFactory::PushFront, 61
- wxArchiveClassFactory::Remove, 61
- wxArchiveEntry::Clone, 62
- wxArchiveEntry::Get/SetDateTime, 62
- wxArchiveEntry::Get/SetName, 63
- wxArchiveEntry::Get/SetSize, 63
- wxArchiveEntry::GetInternalFormat, 63
- wxArchiveEntry::GetInternalName, 63
- wxArchiveEntry::GetOffset, 63
- wxArchiveEntry::IsDir/SetIsDir, 64
- wxArchiveEntry::IsReadOnly/SetIsReadOnly, 64
- wxArchiveEntry::Set/UnsetNotifier, 64
- wxArchiveInputStream::CloseEntry, 65
- wxArchiveInputStream::GetNextEntry, 65
- wxArchiveInputStream::OpenEntry, 65
- wxArchivIterator, 67, 68
- wxArchivIterator::operator\*, 68
- wxArchivIterator::operator++, 68
- wxArchivIterator::wxArchivIterator, 67
- wxArchiveNotifier::OnEntryUpdated, 68
- wxArchiveOutputStream::~~wxArchiveOutputStrea  
m, 69
- wxArchiveOutputStream::Close, 69
- wxArchiveOutputStream::CloseEntry, 69
- wxArchiveOutputStream::CopyArchiveMetaData,  
69
- wxArchiveOutputStream::CopyEntry, 70
- wxArchiveOutputStream::PutNextDirEntry, 70

- wxArchiveOutputStream::PutNextEntry, 70
- wxArray, 78
- wxArray copy constructor and assignment operator, 78
- wxArray::~~wxArray, 78
- wxArray::Add, 79
- wxArray::Alloc, 79
- wxArray::Clear, 79
- wxArray::Count, 79
- wxArray::Empty, 80
- wxArray::GetCount, 80
- wxArray::Index, 80
- wxArray::Insert, 80
- wxArray::IsEmpty, 81
- wxArray::Item, 81
- wxArray::Last, 81
- wxArray::Remove, 81
- wxArray::RemoveAt, 82
- wxArray::SetCount, 82
- wxArray::Shrink, 82
- wxArray::Sort, 82
- wxArrayString, 84
- wxArrayString::~~wxArrayString, 84
- wxArrayString::Add, 85
- wxArrayString::Alloc, 85
- wxArrayString::Clear, 85
- wxArrayString::Count, 85
- wxArrayString::Empty, 85
- wxArrayString::GetCount, 86
- wxArrayString::Index, 86
- wxArrayString::Insert, 86
- wxArrayString::IsEmpty, 86
- wxArrayString::Item, 86
- wxArrayString::Last, 87
- wxArrayString::operator!=, 84
- wxArrayString::operator[], 84
- wxArrayString::operator=, 84
- wxArrayString::operator==, 84
- wxArrayString::Remove, 87
- wxArrayString::RemoveAt, 87
- wxArrayString::Shrink, 87
- wxArrayString::Sort, 87
- wxArrayString::wxArrayString, 84
- wxArtProvider::~~wxArtProvider, 91
- wxArtProvider::CreateBitmap, 91
- wxArtProvider::Delete, 91
- wxArtProvider::GetBitmap, 91
- wxArtProvider::GetIcon, 92
- wxArtProvider::Insert, 92
- wxArtProvider::Pop, 92
- wxArtProvider::Push, 93
- wxArtProvider::Remove, 93
- wxASSERT, 1980
- wxASSERT\_MIN\_BITSIZE, 1980
- wxASSERT\_MSG, 1980
- wxAUI\_NB\_BOTTOM, 106
- wxAUI\_NB\_CLOSE\_BUTTON, 106
- wxAUI\_NB\_CLOSE\_ON\_ACTIVE\_TAB, 106
- wxAUI\_NB\_CLOSE\_ON\_ALL\_TABS, 106
- wxAUI\_NB\_DEFAULT\_STYLE, 105
- wxAUI\_NB\_SCROLL\_BUTTONS, 105
- wxAUI\_NB\_TAB\_EXTERNAL\_MOVE, 105
- wxAUI\_NB\_TAB\_FIXED\_WIDTH, 105
- wxAUI\_NB\_TAB\_MOVE, 105
- wxAUI\_NB\_TAB\_SPLIT, 105
- wxAUI\_NB\_TOP, 106
- wxAUI\_NB\_WINDOWLIST\_BUTTON, 105
- wxAuiDockArt, 94
- wxAuiDockArt::~~wxAuiDockArt, 95
- wxAuiDockArt::DrawBackground, 95
- wxAuiDockArt::DrawBorder, 95
- wxAuiDockArt::DrawCaption, 95
- wxAuiDockArt::DrawGripper, 95
- wxAuiDockArt::DrawPaneButton, 95
- wxAuiDockArt::DrawSash, 95
- wxAuiDockArt::GetColor, 96
- wxAuiDockArt::GetColour, 96
- wxAuiDockArt::GetFont, 96
- wxAuiDockArt::GetMetric, 96
- wxAuiDockArt::SetColor, 96
- wxAuiDockArt::SetColour, 96
- wxAuiDockArt::SetFont, 96
- wxAuiDockArt::SetMetric, 96
- wxAuiDockArt::wxAuiDockArt, 94
- wxAuiManager, 101
- wxAuiManager::~~wxAuiManager, 101
- wxAuiManager::AddPane, 101
- wxAuiManager::DetachPane, 101
- wxAuiManager::GetAllPanels, 101
- wxAuiManager::GetArtProvider, 101
- wxAuiManager::GetDockSizeConstraint, 101
- wxAuiManager::GetFlags, 102
- wxAuiManager::GetManagedWindow, 102
- wxAuiManager::GetManager, 102
- wxAuiManager::GetPane, 102
- wxAuiManager::HideHint, 102
- wxAuiManager::InsertPane, 102
- wxAuiManager::LoadPanelInfo, 103
- wxAuiManager::LoadPerspective, 103
- wxAuiManager::ProcessDockResult, 103
- wxAuiManager::SavePanelInfo, 103
- wxAuiManager::SavePerspective, 103
- wxAuiManager::SetArtProvider, 103
- wxAuiManager::SetDockSizeConstraint, 104
- wxAuiManager::SetFlags, 104
- wxAuiManager::SetManagedWindow, 104
- wxAuiManager::ShowHint, 104
- wxAuiManager::UnInit, 104
- wxAuiManager::Update, 104
- wxAuiManager::wxAuiManager, 101
- wxAuiNotebook, 106
- wxAuiNotebook::AddPage, 106
- wxAuiNotebook::AdvanceSelection, 106
- wxAuiNotebook::Create, 106
- wxAuiNotebook::DeletePage, 106
- wxAuiNotebook::GetArtProvider, 107
- wxAuiNotebook::GetHeightForPageHeight, 107
- wxAuiNotebook::GetPage, 107
- wxAuiNotebook::GetPageBitmap, 107
- wxAuiNotebook::GetPageCount, 107
- wxAuiNotebook::GetPageIndex, 107
- wxAuiNotebook::GetPageText, 107
- wxAuiNotebook::GetSelection, 107
- wxAuiNotebook::GetTabCtrlHeight, 108

wxAuiNotebook::InsertPage, 108  
wxAuiNotebook::RemovePage, 108  
wxAuiNotebook::SetArtProvider, 108  
wxAuiNotebook::SetFont, 108  
wxAuiNotebook::SetMeasuringFont, 108  
wxAuiNotebook::SetNormalFont, 108  
wxAuiNotebook::SetPageBitmap, 109  
wxAuiNotebook::SetPageText, 109  
wxAuiNotebook::SetSelectedFont, 108  
wxAuiNotebook::SetSelection, 109  
wxAuiNotebook::SetTabCtrlHeight, 109  
wxAuiNotebook::SetUniformBitmapSize, 109  
wxAuiNotebook::ShowWindowMenu, 109  
wxAuiNotebook::wxAuiNotebook, 106  
wxAuiPanelInfo, 110  
wxAuiPanelInfo::~wxAuiPanelInfo, 110  
wxAuiPanelInfo::BestSize, 110  
wxAuiPanelInfo::Bottom, 111  
wxAuiPanelInfo::BottomDockable, 111  
wxAuiPanelInfo::Caption, 111  
wxAuiPanelInfo::CaptionVisible, 111  
wxAuiPanelInfo::Centre, 111  
wxAuiPanelInfo::CentrePane, 111  
wxAuiPanelInfo::CloseButton, 112  
wxAuiPanelInfo::DefaultPane, 112  
wxAuiPanelInfo::DestroyOnClose, 112  
wxAuiPanelInfo::Direction, 112  
wxAuiPanelInfo::Dock, 112  
wxAuiPanelInfo::Dockable, 112  
wxAuiPanelInfo::DockFixed, 112  
wxAuiPanelInfo::Fixed, 113  
wxAuiPanelInfo::Float, 113  
wxAuiPanelInfo::Floatable, 113  
wxAuiPanelInfo::FloatingPosition, 113  
wxAuiPanelInfo::FloatingSize, 113  
wxAuiPanelInfo::Gripper, 113  
wxAuiPanelInfo::GripperTop, 113  
wxAuiPanelInfo::HasBorder, 114  
wxAuiPanelInfo::HasCaption, 114  
wxAuiPanelInfo::HasCloseButton, 114  
wxAuiPanelInfo::HasFlag, 114  
wxAuiPanelInfo::HasGripper, 114  
wxAuiPanelInfo::HasGripperTop, 114  
wxAuiPanelInfo::HasMaximizeButton, 114  
wxAuiPanelInfo::HasMinimizeButton, 114  
wxAuiPanelInfo::HasPinButton, 114  
wxAuiPanelInfo::Hide, 115  
wxAuiPanelInfo::IsBottomDockable, 115  
wxAuiPanelInfo::IsDocked, 115  
wxAuiPanelInfo::IsFixed, 115  
wxAuiPanelInfo::IsFloatable, 115  
wxAuiPanelInfo::IsFloating, 115  
wxAuiPanelInfo::IsLeftDockable, 115  
wxAuiPanelInfo::IsMovable, 115  
wxAuiPanelInfo::IsOk, 116  
wxAuiPanelInfo::IsResizable, 116  
wxAuiPanelInfo::IsRightDockable, 116  
wxAuiPanelInfo::IsShown, 116  
wxAuiPanelInfo::IsToolBar, 116  
wxAuiPanelInfo::IsTopDockable, 116  
wxAuiPanelInfo::Layer, 116  
wxAuiPanelInfo::Left, 116  
wxAuiPanelInfo::LeftDockable, 117  
wxAuiPanelInfo::MaximizeButton, 117  
wxAuiPanelInfo::MaxSize, 117  
wxAuiPanelInfo::MinimizeButton, 117  
wxAuiPanelInfo::MinSize, 117  
wxAuiPanelInfo::Movable, 117  
wxAuiPanelInfo::Name, 117  
wxAuiPanelInfo::operator=, 119  
wxAuiPanelInfo::PaneBorder, 118  
wxAuiPanelInfo::PinButton, 118  
wxAuiPanelInfo::Position, 118  
wxAuiPanelInfo::Resizable, 118  
wxAuiPanelInfo::Right, 118  
wxAuiPanelInfo::RightDockable, 118  
wxAuiPanelInfo::Row, 118  
wxAuiPanelInfo::SafeSet, 118  
wxAuiPanelInfo::SetFlag, 119  
wxAuiPanelInfo::Show, 119  
wxAuiPanelInfo::ToolBarPane, 119  
wxAuiPanelInfo::Top, 119  
wxAuiPanelInfo::TopDockable, 119  
wxAuiPanelInfo::Window, 119  
wxAuiPanelInfo::wxAuiPanelInfo, 110  
wxAuiTabArt, 97  
wxAuiTabArt::Clone, 97  
wxAuiTabArt::DrawBackground, 97  
wxAuiTabArt::DrawButton, 97  
wxAuiTabArt::DrawTab, 97  
wxAuiTabArt::GetBestTabCtrlSize, 98  
wxAuiTabArt::GetIndentSize, 98  
wxAuiTabArt::GetTabSize, 98  
wxAuiTabArt::SetFlags, 98  
wxAuiTabArt::SetMeasuringFont, 98  
wxAuiTabArt::SetNormalFont, 98  
wxAuiTabArt::SetSelectedFont, 98  
wxAuiTabArt::SetSizingInfo, 98  
wxAuiTabArt::ShowWindowList, 99  
wxAuiTabArt::wxAuiTabArt, 97  
wxAutoBufferedPaintDC, 161  
wxAutoBufferedPaintDC::wxAutoBufferedPaintDC, 160  
wxAutomationObject, 120  
wxAutomationObject::~wxAutomationObject, 120  
wxAutomationObject::CallMethod, 120  
wxAutomationObject::CreateInstance, 121  
wxAutomationObject::GetDispatchPtr, 121  
wxAutomationObject::GetInstance, 121  
wxAutomationObject::GetObject, 121  
wxAutomationObject::GetProperty, 122  
wxAutomationObject::Invoke, 122  
wxAutomationObject::PutProperty, 123  
wxAutomationObject::SetDispatchPtr, 123  
wxAutomationObject::wxAutomationObject, 120  
wxBeginBusyCursor, 1934  
wxBell, 1935  
wxBitmap, 124, 125  
wxBITMAP, 1944  
wxBitmap::~wxBitmap, 127  
wxBitmap::AddHandler, 128  
wxBitmap::CleanUpHandlers, 128  
wxBitmap::ConvertToImage, 128  
wxBitmap::CopyFromIcon, 128



wxBitmap::Create, 128  
wxBitmap::FindHandler, 129  
wxBitmap::GetDepth, 130  
wxBitmap::GetHandlers, 130  
wxBitmap::GetHeight, 130  
wxBitmap::GetMask, 130  
wxBitmap::GetPalette, 130  
wxBitmap::GetSubBitmap, 131  
wxBitmap::GetWidth, 131  
wxBitmap::InitStandardHandlers, 131  
wxBitmap::InsertHandler, 131  
wxBitmap::IsOk, 132  
wxBitmap::LoadFile, 131  
wxBitmap::operator =, 135  
wxBitmap::RemoveHandler, 132  
wxBitmap::SaveFile, 133  
wxBitmap::SetDepth, 134  
wxBitmap::SetHeight, 134  
wxBitmap::SetMask, 134  
wxBitmap::SetPalette, 134  
wxBitmap::SetWidth, 135  
wxBitmap::wxBitmap, 124  
wxBITMAP\_TYPE\_ANI, 909, 916, 921  
wxBITMAP\_TYPE\_ANY, 909, 916  
wxBITMAP\_TYPE\_BMP, 126, 908, 916  
wxBITMAP\_TYPE\_BMP\_RESOURCE, 126  
wxBITMAP\_TYPE\_CUR, 299, 909, 916  
wxBITMAP\_TYPE\_CUR\_RESOURCE, 299  
wxBITMAP\_TYPE\_GIF, 126, 897, 908, 916  
wxBITMAP\_TYPE\_ICO, 300, 897, 909, 916  
wxBITMAP\_TYPE\_ICO\_RESOURCE, 897  
wxBITMAP\_TYPE\_JPEG, 908, 916  
wxBITMAP\_TYPE\_PCX, 908, 916  
wxBITMAP\_TYPE\_PICT\_RESOURCE, 126  
wxBITMAP\_TYPE\_PNG, 908, 916  
wxBITMAP\_TYPE\_PNM, 908, 916  
wxBITMAP\_TYPE\_TGA, 909  
wxBITMAP\_TYPE\_TIF, 908, 916  
wxBITMAP\_TYPE\_XBM, 126, 299, 897  
wxBITMAP\_TYPE\_XPM, 126, 897, 909, 916  
wxBitmapButton, 140, 141  
wxBitmapButton::~~wxBitmapButton, 141  
wxBitmapButton::Create, 142  
wxBitmapButton::GetBitmapDisabled, 142  
wxBitmapButton::GetBitmapFocus, 142  
wxBitmapButton::GetBitmapHover, 142  
wxBitmapButton::GetBitmapLabel, 142  
wxBitmapButton::GetBitmapSelected, 143  
wxBitmapButton::SetBitmapDisabled, 143  
wxBitmapButton::SetBitmapFocus, 143  
wxBitmapButton::SetBitmapHover, 144  
wxBitmapButton::SetBitmapLabel, 144  
wxBitmapButton::SetBitmapSelected, 144  
wxBitmapButton::wxBitmapButton, 140  
wxBitmapComboBox, 136, 137  
wxBitmapComboBox::~~wxBitmapComboBox, 138  
wxBitmapComboBox::Append, 138  
wxBitmapComboBox::Create, 138  
wxBitmapComboBox::GetBitmapSize, 138  
wxBitmapComboBox::GetItemBitmap, 138  
wxBitmapComboBox::Insert, 138  
wxBitmapComboBox::SetItemBitmap, 139  
wxBitmapComboBox::wxBitmapComboBox, 136  
wxBitmapDataObject, 145  
wxBitmapDataObject::GetBitmap, 145  
wxBitmapDataObject::SetBitmap, 145  
wxBitmapHandler, 146  
wxBitmapHandler::~~wxBitmapHandler, 146  
wxBitmapHandler::Create, 146  
wxBitmapHandler::GetExtension, 147  
wxBitmapHandler::GetName, 147  
wxBitmapHandler::GetType, 147  
wxBitmapHandler::LoadFile, 147  
wxBitmapHandler::SaveFile, 148  
wxBitmapHandler::SetExtension, 149  
wxBitmapHandler::SetName, 149  
wxBitmapHandler::SetType, 149  
wxBitmapHandler::wxBitmapHandler, 146  
wxBK\_BOTTOM, 2128  
wxBK\_DEFAULT, 1712, 2128  
wxBK\_LEFT, 2128  
wxBK\_RIGHT, 2128  
wxBK\_TOP, 2128  
wxBMPHandler, 906  
wxBORDER\_DOUBLE, 1796  
wxBORDER\_NONE, 1796  
wxBORDER\_RAISED, 1796  
wxBORDER\_SIMPLE, 1796  
wxBORDER\_STATIC, 1796  
wxBORDER\_SUNKEN, 1796  
wxBORDER\_THEME, 1796  
wxBOTTOM, 1447  
wxBoxSizer, 150  
wxBoxSizer::CalcMin, 150  
wxBoxSizer::GetOrientation, 150  
wxBoxSizer::RecalcSizes, 150  
wxBoxSizer::wxBoxSizer, 150  
wxBrush, 151, 152  
wxBrush::~~wxBrush, 152  
wxBrush::GetColour, 153  
wxBrush::GetStipple, 153  
wxBrush::GetStyle, 153  
wxBrush::IsHatch, 154  
wxBrush::IsOk, 154  
wxBrush::operator !=, 156  
wxBrush::operator =, 156  
wxBrush::operator ==, 156  
wxBrush::SetColour, 154  
wxBrush::SetStipple, 154  
wxBrush::SetStyle, 155  
wxBrush::wxBrush, 151  
wxBrushList, 157  
wxBrushList::FindOrCreateBrush, 157  
wxBrushList::wxBrushList, 157  
wxBU\_AUTODRAW, 140  
wxBU\_BOTTOM, 140, 164  
wxBU\_EXACTFIT, 140, 164  
wxBU\_LEFT, 140, 164  
wxBU\_RIGHT, 140, 164  
wxBU\_TOP, 140, 164  
wxBufferedDC, 158  
wxBufferedDC::~~wxBufferedDC, 159  
wxBufferedDC::Init, 159  
wxBufferedDC::wxBufferedDC, 158

- wxBufferedOutputStream, 161
- wxBufferedOutputStream::~wxBufferedOutputStream, 162
- wxBufferedOutputStream::SeekO, 162
- wxBufferedOutputStream::Sync, 162
- wxBufferedOutputStream::wxBufferedOutputStream, 161
- wxBufferedPaintDC, 160
- wxBufferedPaintDC::~wxBufferedPaintDC, 160
- wxBufferedPaintDC::wxBufferedPaintDC, 160
- wxBusyCursor, 162
- wxBusyCursor::~wxBusyCursor, 163
- wxBusyCursor::wxBusyCursor, 162
- wxBusyInfo, 164
- wxBusyInfo::~wxBusyInfo, 164
- wxBusyInfo::wxBusyInfo, 164
- wxButton, 165
- wxButton::~wxButton, 166
- wxButton::Create, 166
- wxButton::GetDefaultSize, 166
- wxButton::GetLabel, 166
- wxButton::SetDefault, 166
- wxButton::SetLabel, 167
- wxButton::wxButton, 165
- wxCAL\_MONDAY\_FIRST, 169
- wxCAL\_NO\_MONTH\_CHANGE, 169
- wxCAL\_NO\_YEAR\_CHANGE, 169
- wxCAL\_SEQUENTIAL\_MONTH\_SELECTION, 169
- wxCAL\_SHOW\_HOLIDAYS, 169
- wxCAL\_SHOW\_SURROUNDING\_WEEKS, 169
- wxCAL\_SUNDAY\_FIRST, 169
- wxCalculateLayoutEvent, 168
- wxCalculateLayoutEvent::GetFlags, 168
- wxCalculateLayoutEvent::GetRect, 168
- wxCalculateLayoutEvent::SetFlags, 168
- wxCalculateLayoutEvent::SetRect, 168
- wxCalculateLayoutEvent::wxCalculateLayoutEvent, 168
- wxCalendarCtrl, 170
- wxCalendarCtrl::~wxCalendarCtrl, 170
- wxCalendarCtrl::Create, 170
- wxCalendarCtrl::EnableHolidayDisplay, 171
- wxCalendarCtrl::EnableMonthChange, 171
- wxCalendarCtrl::EnableYearChange, 171
- wxCalendarCtrl::GetAttr, 173
- wxCalendarCtrl::GetDate, 171
- wxCalendarCtrl::GetHeaderColourBg, 172
- wxCalendarCtrl::GetHeaderColourFg, 171
- wxCalendarCtrl::GetHighlightColourBg, 172
- wxCalendarCtrl::GetHighlightColourFg, 172
- wxCalendarCtrl::GetHolidayColourBg, 173
- wxCalendarCtrl::GetHolidayColourFg, 172
- wxCalendarCtrl::HitTest, 173
- wxCalendarCtrl::ResetAttr, 173
- wxCalendarCtrl::SetAttr, 173
- wxCalendarCtrl::SetDate, 171
- wxCalendarCtrl::SetHeaderColours, 171
- wxCalendarCtrl::SetHighlightColours, 172
- wxCalendarCtrl::SetHoliday, 173
- wxCalendarCtrl::SetHolidayColours, 172
- wxCalendarCtrl::wxCalendarCtrl, 170
- wxCalendarDateAttr, 174
- wxCalendarDateAttr::GetBackgroundColour, 176
- wxCalendarDateAttr::GetBorder, 176
- wxCalendarDateAttr::GetBorderColour, 176
- wxCalendarDateAttr::GetFont, 176
- wxCalendarDateAttr::GetTextColour, 176
- wxCalendarDateAttr::HasBackgroundColour, 175
- wxCalendarDateAttr::HasBorder, 176
- wxCalendarDateAttr::HasBorderColour, 175
- wxCalendarDateAttr::HasFont, 175
- wxCalendarDateAttr::HasTextColour, 175
- wxCalendarDateAttr::IsHoliday, 176
- wxCalendarDateAttr::SetBackgroundColour, 174
- wxCalendarDateAttr::SetBorder, 175
- wxCalendarDateAttr::SetBorderColour, 175
- wxCalendarDateAttr::SetFont, 175
- wxCalendarDateAttr::SetHoliday, 175
- wxCalendarDateAttr::SetTextColour, 174
- wxCalendarDateAttr::wxCalendarDateAttr, 174
- wxCalendarEvent::GetWeekDay, 177
- wxCalendarEvent::SetWeekDay, 177
- wxCAPTION, 497, 682, 1047, 1052, 1113
- wxCaret, 178
- wxCaret::Create, 178
- wxCaret::GetBlinkTime, 178
- wxCaret::GetPosition, 178
- wxCaret::GetSize, 178
- wxCaret::GetWindow, 179
- wxCaret::Hide, 179
- wxCaret::IsOk, 179
- wxCaret::IsVisible, 179
- wxCaret::Move, 179
- wxCaret::SetBlinkTime, 179
- wxCaret::SetSize, 180
- wxCaret::Show, 180
- wxCaret::wxCaret, 177
- wxCB\_DROPDOWN, 226
- wxCB\_READONLY, 136, 226, 234
- wxCB\_SIMPLE, 226
- wxCB\_SORT, 136, 226, 234
- wxCC\_SPECIAL\_DCLICK, 234
- wxCC\_STD\_BUTTON, 234
- wxCHANGE\_UMASK, 1922
- wxCHB\_BOTTOM, 190
- wxCHB\_DEFAULT, 189
- wxCHB\_LEFT, 190
- wxCHB\_RIGHT, 190
- wxCHB\_TOP, 189
- wxCHECK, 1982
- wxCHECK\_GCC\_VERSION, 1908
- wxCHECK\_MSG, 1982
- wxCHECK\_RET, 1982
- wxCHECK\_VERSION, 1909
- wxCHECK\_VERSION\_FULL, 1909
- wxCHECK\_W32API\_VERSION, 1909
- wxCHECK2, 1982
- wxCHECK2\_MSG, 1982
- wxCheckBox, 181
- wxCheckBox::~wxCheckBox, 182
- wxCheckBox::Create, 182
- wxCheckBox::Get3StateValue, 182
- wxCheckBox::GetValue, 182

---

wxCheckBox::Is3rdStateAllowedForUser, 182  
 wxCheckBox::Is3State, 183  
 wxCheckBox::IsChecked, 183  
 wxCheckBox::Set3StateValue, 183  
 wxCheckBox::SetValue, 183  
 wxCheckBox::wxCheckBox, 181  
 wxCheckListBox, 184  
 wxCheckListBox::~~wxCheckListBox, 185  
 wxCheckListBox::Check, 185  
 wxCheckListBox::IsChecked, 186  
 wxCheckListBox::wxCheckListBox, 184  
 wxChildFocusEvent, 281  
 wxChildFocusEvent::GetWindow, 281  
 wxChildFocusEvent::wxChildFocusEvent, 281  
 wxCHK\_2STATE, 180  
 wxCHK\_3STATE, 180  
 wxCHK\_ALLOW\_3RD\_STATE\_FOR\_USER, 180  
 wxChoice, 187  
 wxChoice::~~wxChoice, 188  
 wxChoice::Create, 188  
 wxChoice::GetColumns, 188  
 wxChoice::GetCurrentSelection, 188  
 wxChoice::SetColumns, 189  
 wxChoice::wxChoice, 187  
 wxClassInfo, 190, 2045  
 wxClassInfo::CreateObject, 190  
 wxClassInfo::FindClass, 190  
 wxClassInfo::GetBaseClassName1, 191  
 wxClassInfo::GetBaseClassName2, 191  
 wxClassInfo::GetClassName, 191  
 wxClassInfo::GetSize, 191  
 wxClassInfo::InitializeClasses, 191  
 wxClassInfo::IsDynamic, 191  
 wxClassInfo::IsKindOf, 191  
 wxClassInfo::wxClassInfo, 190  
 wxClient, 192  
 wxClient::MakeConnection, 192  
 wxClient::OnMakeConnection, 192  
 wxClient::ValidHost, 193  
 wxClient::wxClient, 192  
 wxClientData, 194  
 wxClientData::~~wxClientData, 194  
 wxClientData::wxClientData, 194  
 wxClientDataContainer, 195  
 wxClientDataContainer::~~wxClientDataContainer, 195  
 wxClientDataContainer::GetClientData, 195  
 wxClientDataContainer::GetClientObject, 195  
 wxClientDataContainer::SetClientData, 195  
 wxClientDataContainer::SetClientObject, 195  
 wxClientDataContainer::wxClientDataContainer, 195  
 wxClientDC, 193  
 wxClientDC::wxClientDC, 193  
 wxClientDisplayRect, 1945  
 wxCLIP\_CHILDREN, 1797  
 wxClipboard, 196  
 wxClipboard::~~wxClipboard, 197  
 wxClipboard::AddData, 197  
 wxClipboard::Clear, 197  
 wxClipboard::Close, 197  
 wxClipboard::Flush, 197  
 wxClipboard::GetData, 197  
 wxClipboard::IsOpened, 197  
 wxClipboard::IsSupported, 198  
 wxClipboard::Open, 198  
 wxClipboard::SetData, 198  
 wxClipboard::UsePrimarySelection, 198  
 wxClipboard::wxClipboard, 196  
 wxClipboardOpen, 1949  
 wxClipboardTextEvent, 199  
 wxClipboardTextEvent::wxClipboardTextEvent, 199  
 wxCLOSE\_BOX, 497, 682, 1114  
 wxCloseClipboard, 1949  
 wxCloseEvent, 200  
 wxCloseEvent::CanVeto, 201  
 wxCloseEvent::GetLoggingOff, 201  
 wxCloseEvent::SetCanVeto, 201  
 wxCloseEvent::SetForce, 201  
 wxCloseEvent::SetLoggingOff, 201  
 wxCloseEvent::Veto, 201  
 wxCloseEvent::wxCloseEvent, 200  
 wxCLRP\_DEFAULT\_STYLE, 222  
 wxCLRP\_SHOW\_LABEL, 223  
 wxCLRP\_USE\_TEXTCTRL, 223  
 wxCmdLineParser, 205, 206  
 wxCmdLineParser::~~wxCmdLineParser, 207  
 wxCmdLineParser::AddOption, 208  
 wxCmdLineParser::AddParam, 209  
 wxCmdLineParser::AddSwitch, 208  
 wxCmdLineParser::AreLongOptionsEnabled, 207  
 wxCmdLineParser::ConvertStringToArgs, 206  
 wxCmdLineParser::DisableLongOptions, 207  
 wxCmdLineParser::EnableLongOptions, 207  
 wxCmdLineParser::Found, 209, 210  
 wxCmdLineParser::GetParam, 210  
 wxCmdLineParser::GetParamCount, 210  
 wxCmdLineParser::Parse, 209  
 wxCmdLineParser::SetCmdLine, 206  
 wxCmdLineParser::SetDesc, 208  
 wxCmdLineParser::SetLogo, 207  
 wxCmdLineParser::SetSwitchChars, 207  
 wxCmdLineParser::Usage, 209  
 wxCmdLineParser::wxCmdLineParser, 205, 206  
 wxCollapsiblePane, 211  
 wxCollapsiblePane::Collapse, 213  
 wxCollapsiblePane::Create, 211  
 wxCollapsiblePane::Expand, 213  
 wxCollapsiblePane::GetPane, 213  
 wxCollapsiblePane::IsCollapsed, 212  
 wxCollapsiblePane::IsExpanded, 212  
 wxCollapsiblePane::wxCollapsiblePane, 211  
 wxCollapsiblePaneEvent, 213  
 wxCollapsiblePaneEvent::GetCollapsed, 214  
 wxCollapsiblePaneEvent::SetCollapsed, 214  
 wxCollapsiblePaneEvent::wxCollapsiblePaneEvent, 213  
 wxColour, 215  
 wxColour::Alpha, 216  
 wxColour::Blue, 216  
 wxColour::GetAsString, 216  
 wxColour::GetPixel, 216  
 wxColour::Green, 216

---

- wxColour::IsOk, 217
- wxColour::operator !=, 218
- wxColour::operator =, 217
- wxColour::operator ==, 217
- wxColour::Red, 217
- wxColour::Set, 217
- wxColour::wxColour, 215
- wxColourData, 218
- wxColourData::~wxColourData, 218
- wxColourData::GetChooseFull, 218
- wxColourData::GetColour, 219
- wxColourData::GetCustomColour, 219
- wxColourData::operator =, 219
- wxColourData::SetChooseFull, 219
- wxColourData::SetColour, 219
- wxColourData::SetCustomColour, 219
- wxColourData::wxColourData, 218
- wxColourDatabase, 220
- wxColourDatabase::AddColour, 220
- wxColourDatabase::Find, 221
- wxColourDatabase::FindName, 221
- wxColourDatabase::wxColourDatabase, 220
- wxColourDialog, 221
- wxColourDialog overview, 2128
- wxColourDialog::~wxColourDialog, 222
- wxColourDialog::Create, 222
- wxColourDialog::GetColourData, 222
- wxColourDialog::ShowModal, 222
- wxColourDialog::wxColourDialog, 221
- wxColourDisplay, 1945
- wxColourPickerCtrl, 223
- wxColourPickerCtrl::Create, 223
- wxColourPickerCtrl::GetColour, 224
- wxColourPickerCtrl::SetColour, 224
- wxColourPickerCtrl::wxColourPickerCtrl, 223
- wxColourPickerEvent, 225
- wxColourPickerEvent::GetColour, 225
- wxColourPickerEvent::SetColour, 225
- wxColourPickerEvent::wxColourPickerEvent, 225
- wxComboBox, 227
- wxComboBox::~wxComboBox, 228
- wxComboBox::CanCopy, 228
- wxComboBox::CanCut, 228
- wxComboBox::CanPaste, 228
- wxComboBox::CanRedo, 229
- wxComboBox::CanUndo, 229
- wxComboBox::Copy, 229
- wxComboBox::Create, 228
- wxComboBox::Cut, 229
- wxComboBox::GetCurrentSelection, 229
- wxComboBox::GetInsertionPoint, 229
- wxComboBox::GetLastPosition, 229
- wxComboBox::GetSelection, 230
- wxComboBox::GetValue, 230
- wxComboBox::Paste, 230
- wxComboBox::Redo, 230
- wxComboBox::Remove, 230
- wxComboBox::Replace, 230
- wxComboBox::SetInsertionPoint, 231
- wxComboBox::SetInsertionPointEnd, 231
- wxComboBox::SetSelection, 231
- wxComboBox::SetValue, 231
- wxComboBox::Undo, 232
- wxComboBox::wxComboBox, 226
- wxComboCtrl, 235
- wxComboCtrl::~wxComboCtrl, 236
- wxComboCtrl::AnimateShow, 236
- wxComboCtrl::Copy, 236
- wxComboCtrl::Create, 236
- wxComboCtrl::Cut, 236
- wxComboCtrl::DoSetPopupControl, 237
- wxComboCtrl::DoShowPopup, 237
- wxComboCtrl::EnablePopupAnimation, 237
- wxComboCtrl::GetBitmapDisabled, 237
- wxComboCtrl::GetBitmapHover, 238
- wxComboCtrl::GetBitmapNormal, 238
- wxComboCtrl::GetBitmapPressed, 238
- wxComboCtrl::GetButtonSize, 238
- wxComboCtrl::GetCustomPaintWidth, 238
- wxComboCtrl::GetFeatures, 238
- wxComboCtrl::GetInsertionPoint, 239
- wxComboCtrl::GetLastPosition, 239
- wxComboCtrl::GetPopupControl, 240
- wxComboCtrl::GetPopupWindow, 240
- wxComboCtrl::GetTextCtrl, 240
- wxComboCtrl::GetTextIndent, 240
- wxComboCtrl::GetTextRect, 240
- wxComboCtrl::GetValue, 240
- wxComboCtrl::HidePopup, 240
- wxComboCtrl::IsPopupShown, 240
- wxComboCtrl::IsPopupWindowState, 239
- wxComboCtrl::OnButtonClick, 241
- wxComboCtrl::Paste, 241
- wxComboCtrl::Remove, 241
- wxComboCtrl::Replace, 241
- wxComboCtrl::SetButtonBitmaps, 242
- wxComboCtrl::SetButtonPosition, 242
- wxComboCtrl::SetCustomPaintWidth, 243
- wxComboCtrl::SetInsertionPoint, 243
- wxComboCtrl::SetInsertionPointEnd, 243
- wxComboCtrl::SetPopupAnchor, 243
- wxComboCtrl::SetPopupControl, 243
- wxComboCtrl::SetPopupExtents, 243
- wxComboCtrl::SetPopupMaxHeight, 244
- wxComboCtrl::SetPopupMinWidth, 244
- wxComboCtrl::SetSelection, 244
- wxComboCtrl::SetText, 245
- wxComboCtrl::SetTextIndent, 245
- wxComboCtrl::SetValue, 245
- wxComboCtrl::SetValueWithEvent, 245
- wxComboCtrl::ShowPopup, 245
- wxComboCtrl::Undo, 245
- wxComboCtrl::UseAltPopupWindow, 245
- wxComboCtrl::wxComboCtrl, 235
- wxComboPopup, 246
- wxComboPopup::Create, 246
- wxComboPopup::Dismiss, 246
- wxComboPopup::GetAdjustedSize, 246
- wxComboPopup::GetControl, 247
- wxComboPopup::GetStringValue, 247
- wxComboPopup::Init, 247
- wxComboPopup::IsCreated, 247
- wxComboPopup::LazyCreate, 248
- wxComboPopup::m\_combo, 246

- wxComboPopup::OnComboDoubleClick, 248
- wxComboPopup::OnComboKeyEvent, 248
- wxComboPopup::OnDismiss, 248
- wxComboPopup::OnPopup, 248
- wxComboPopup::PaintComboControl, 248
- wxComboPopup::SetStringValue, 248
- wxComboPopup::wxComboPopup, 246
- wxCommand, 249
- wxCommand overview, 2135
- wxCommand::~~wxCommand, 249
- wxCommand::CanUndo, 249
- wxCommand::Do, 249
- wxCommand::GetName, 250
- wxCommand::Undo, 250
- wxCommand::wxCommand, 249
- wxCommandEvent, 253
- wxCommandEvent::Checked, 254
- wxCommandEvent::GetClientData, 254
- wxCommandEvent::GetClientObject, 254
- wxCommandEvent::GetExtraLong, 254
- wxCommandEvent::GetInt, 254
- wxCommandEvent::GetSelection, 254
- wxCommandEvent::GetString, 254
- wxCommandEvent::IsChecked, 255
- wxCommandEvent::IsSelection, 255
- wxCommandEvent::SetClientData, 255
- wxCommandEvent::SetClientObject, 255
- wxCommandEvent::SetExtraLong, 255
- wxCommandEvent::SetInt, 255
- wxCommandEvent::SetString, 255
- wxCommandEvent::wxCommandEvent, 253
- wxCommandProcessor, 256
- wxCommandProcessor overview, 2136
- wxCommandProcessor::~~wxCommandProcessor, 256
- wxCommandProcessor::CanUndo, 256
- wxCommandProcessor::ClearCommands, 256
- wxCommandProcessor::GetCommands, 257
- wxCommandProcessor::GetEditMenu, 257
- wxCommandProcessor::GetMaxCommands, 257
- wxCommandProcessor::GetRedoAccelerator, 257
- wxCommandProcessor::GetRedoMenuLabel, 257
- wxCommandProcessor::GetUndoAccelerator, 257
- wxCommandProcessor::GetUndoMenuLabel, 257
- wxCommandProcessor::Initialize, 257
- wxCommandProcessor::IsDirty, 257
- wxCommandProcessor::MarkAsSaved, 258
- wxCommandProcessor::Redo, 256
- wxCommandProcessor::SetEditMenu, 258
- wxCommandProcessor::SetMenuStrings, 258
- wxCommandProcessor::SetRedoAccelerator, 258
- wxCommandProcessor::SetUndoAccelerator, 258
- wxCommandProcessor::Submit, 258
- wxCommandProcessor::Undo, 259
- wxCommandProcessor::wxCommandProcessor, 256
- wxCOMPILE\_TIME\_ASSERT, 1981
- wxCOMPILE\_TIME\_ASSERT2, 1981
- wxCONCAT, 1951
- wxConcatFiles, 1923
- wxCondition, 261
- wxCondition::~~wxCondition, 261
- wxCondition::Broadcast, 261
- wxCondition::IsOk, 261
- wxCondition::Signal, 261
- wxCondition::Wait, 262
- wxCondition::WaitTimeout, 262
- wxCondition::wxCondition, 261
- wxConfigBase, 268
- wxConfigBase::~~wxConfigBase, 270
- wxConfigBase::Create, 270
- wxConfigBase::DeleteAll, 270
- wxConfigBase::DeleteEntry, 270
- wxConfigBase::DeleteGroup, 270
- wxConfigBase::DontCreateOnDemand, 270
- wxConfigBase::Exists, 271
- wxConfigBase::Flush, 271
- wxConfigBase::Get, 271
- wxConfigBase::GetAppName, 271
- wxConfigBase::GetEntryType, 271
- wxConfigBase::GetFirstEntry, 272
- wxConfigBase::GetFirstGroup, 271
- wxConfigBase::GetNextEntry, 272
- wxConfigBase::GetNextGroup, 272
- wxConfigBase::GetNumberOfEntries, 272
- wxConfigBase::GetNumberOfGroups, 272
- wxConfigBase::GetPath, 273
- wxConfigBase::GetVendorName, 273
- wxConfigBase::HasEntry, 273
- wxConfigBase::HasGroup, 273
- wxConfigBase::IsExpandingEnvVars, 273
- wxConfigBase::IsRecordingDefaults, 273
- wxConfigBase::Read, 273
- wxConfigBase::RenameEntry, 275
- wxConfigBase::RenameGroup, 275
- wxConfigBase::Set, 275
- wxConfigBase::SetExpandEnvVars, 275
- wxConfigBase::SetPath, 275
- wxConfigBase::SetRecordDefaults, 276
- wxConfigBase::Write, 276
- wxConfigBase::wxConfigBase, 268
- wxConnection, 277, 278
- wxConnection::Advise, 278
- wxConnection::Disconnect, 278
- wxConnection::Execute, 278
- wxConnection::OnAdvise, 278
- wxConnection::OnDisconnect, 279
- wxConnection::OnExecute, 279
- wxConnection::OnPoke, 279
- wxConnection::OnRequest, 279
- wxConnection::OnStartAdvise, 279
- wxConnection::OnStopAdvise, 279
- wxConnection::Poke, 280
- wxConnection::Request, 280
- wxConnection::StartAdvise, 280
- wxConnection::StopAdvise, 280
- wxConnection::wxConnection, 277
- wxConstCast, 1968

- wxContextHelp, 283
- wxContextHelp::~~wxContextHelp, 283
- wxContextHelp::BeginContextHelp, 283
- wxContextHelp::EndContextHelp, 284
- wxContextHelp::wxContextHelp, 283
- wxContextHelpButton, 284
- wxContextHelpButton::wxContextHelpButton, 284
- wxContextMenuEvent, 282
- wxContextMenuEvent::GetPosition, 282
- wxContextMenuEvent::SetPosition, 282
- wxContextMenuEvent::wxContextMenuEvent, 282
- wxControl::Command, 285
- wxControl::GetLabel, 286
- wxControl::GetLabelText, 286
- wxControl::SetLabel, 286
- wxControlWithItems::Append, 287
- wxControlWithItems::Clear, 287
- wxControlWithItems::Delete, 288
- wxControlWithItems::FindString, 288
- wxControlWithItems::GetClientData, 288
- wxControlWithItems::GetClientObject, 289
- wxControlWithItems::GetCount, 289
- wxControlWithItems::GetSelection, 289
- wxControlWithItems::GetString, 289
- wxControlWithItems::GetStrings, 290
- wxControlWithItems::GetStringSelection, 290
- wxControlWithItems::Insert, 290
- wxControlWithItems::IsEmpty, 291
- wxControlWithItems::Number, 291
- wxControlWithItems::Select, 291
- wxControlWithItems::SetClientData, 291
- wxControlWithItems::SetClientObject, 292
- wxControlWithItems::SetSelection, 292
- wxControlWithItems::SetString, 292
- wxControlWithItems::SetStringSelection, 293
- wxCopyFile, 1923
- wxCountingOutputStream, 293
- wxCountingOutputStream::~~wxCountingOutputStream, 293
- wxCountingOutputStream::GetSize, 294
- wxCountingOutputStream::wxCountingOutputStream, 293
- wxCPU\_DEFAULT\_STYLE, 211
- wxCreateDynamicObject, 1968
- wxCreateFileTipProvider, 1935
- wxCRT\_SECT\_DECLARE, 1917
- wxCRT\_SECT\_DECLARE\_MEMBER, 1917
- wxCRT\_SECT\_LOCKER, 1918
- wxCRT\_CRITICAL\_SECTION, 1918
- wxCriticalSection, 294
- wxCriticalSection::~~wxCriticalSection, 294
- wxCriticalSection::Enter, 294
- wxCriticalSection::Leave, 295
- wxCriticalSection::wxCriticalSection, 294
- wxCriticalSectionLocker, 296
- wxCriticalSectionLocker::~~wxCriticalSectionLocker, 296
- wxCriticalSectionLocker::wxCriticalSectionLocker, 296
- wxCSCConv, 296, 2060
- wxCSCConv::~~wxCSCConv, 296
- wxCSCConv::IsOk, 297
- wxCSCConv::MB2WC, 297
- wxCSCConv::WC2MB, 297
- wxCSCConv::wxCSCConv, 296
- wxCURHandler, 907
- wxCurrentTipProvider::GetCurrentTip, 1690
- wxCursor, 298, 299
- wxCursor::~~wxCursor, 302
- wxCursor::IsOk, 302
- wxCursor::operator =, 302
- wxCursor::wxCursor, 298
- wxCustomDataObject, 303
- wxCustomDataObject::~~wxCustomDataObject, 303
- wxCustomDataObject::Alloc, 303
- wxCustomDataObject::Free, 304
- wxCustomDataObject::GetData, 304
- wxCustomDataObject::GetSize, 304
- wxCustomDataObject::SetData, 304
- wxCustomDataObject::TakeData, 304
- wxCustomDataObject::wxCustomDataObject, 303
- wxDataFormat, 305, 306
- wxDataFormat::GetId, 306
- wxDataFormat::GetType, 306
- wxDataFormat::operator !=, 306
- wxDataFormat::operator ==, 306
- wxDataFormat::SetId, 306
- wxDataFormat::SetType, 306
- wxDataFormat::wxDataFormat, 305, 306
- wxDatagramSocket, 307
- wxDatagramSocket::~~wxDatagramSocket, 307
- wxDatagramSocket::ReceiveFrom, 307
- wxDatagramSocket::SendTo, 308
- wxDatagramSocket::wxDatagramSocket, 307
- wxDatInputStream, 309
- wxDatInputStream::~~wxDatInputStream, 310
- wxDatInputStream::BigEndianOrdered, 310
- wxDatInputStream::Read16, 310
- wxDatInputStream::Read32, 310
- wxDatInputStream::Read64, 310
- wxDatInputStream::Read8, 310
- wxDatInputStream::ReadDouble, 311
- wxDatInputStream::ReadString, 311
- wxDatInputStream::wxDatInputStream, 309
- wxDataObject, 313
- wxDataObject::~~wxDataObject, 313
- wxDataObject::GetAllFormats, 313
- wxDataObject::GetDataHere, 314
- wxDataObject::GetDataSize, 314
- wxDataObject::GetFormatCount, 314
- wxDataObject::GetPreferredFormat, 314
- wxDataObject::SetData, 314
- wxDataObject::wxDataObject, 313
- wxDataObjectComposite, 335
- wxDataObjectComposite::Add, 335
- wxDataObjectComposite::GetReceivedFormat, 335
- wxDataObjectComposite::wxDataObjectComposite, 335
- wxDataObjectSimple, 336
- wxDataObjectSimple::GetDataHere, 336

- wxDataObjectSimple::GetDataSize, 336
- wxDataObjectSimple::GetFormat, 336
- wxDataObjectSimple::SetData, 337
- wxDataObjectSimple::SetFormat, 336
- wxDataObjectSimple::wxDataObjectSimple, 336
- wxDataOutputStream, 337
- wxDataOutputStream::~wxDataOutputStream, 338
- wxDataOutputStream::BigEndianOrdered, 338
- wxDataOutputStream::Write16, 338
- wxDataOutputStream::Write32, 338
- wxDataOutputStream::Write64, 339
- wxDataOutputStream::Write8, 338
- wxDataOutputStream::WriteDouble, 339
- wxDataOutputStream::WriteString, 339
- wxDataOutputStream::wxDataOutputStream, 337
- wxDataViewBitmapRenderer, 332
- wxDataViewBitmapRenderer::wxDataViewBitmapRenderer, 332
- wxDataViewColumn, 315
- wxDataViewColumn::~wxDataViewColumn, 315
- wxDataViewColumn::GetBitmap, 315
- wxDataViewColumn::GetModelColumn, 315
- wxDataViewColumn::GetOwner, 315
- wxDataViewColumn::GetRenderer, 316
- wxDataViewColumn::GetSortable, 316
- wxDataViewColumn::GetWidth, 316
- wxDataViewColumn::IsSortOrderAscending, 316
- wxDataViewColumn::SetAlignment, 316
- wxDataViewColumn::SetBitmap, 316
- wxDataViewColumn::SetSortable, 317
- wxDataViewColumn::SetSortOrder, 316
- wxDataViewColumn::SetTitle, 317
- wxDataViewColumn::wxDataViewColumn, 315
- wxDataViewCtrl, 318
- wxDataViewCtrl::~wxDataViewCtrl, 318
- wxDataViewCtrl::AppendBitmapColumn, 318
- wxDataViewCtrl::AppendColumn, 318
- wxDataViewCtrl::AppendDateColumn, 319
- wxDataViewCtrl::AppendProgressColumn, 319
- wxDataViewCtrl::AppendTextColumn, 319
- wxDataViewCtrl::AppendToggleColumn, 319
- wxDataViewCtrl::AssociateModel, 319
- wxDataViewCtrl::ClearColumns, 320
- wxDataViewCtrl::ClearSelection, 320
- wxDataViewCtrl::Create, 319
- wxDataViewCtrl::DeleteColumn, 320
- wxDataViewCtrl::GetColumn, 320
- wxDataViewCtrl::GetModel, 320
- wxDataViewCtrl::GetNumberOfColumns, 320
- wxDataViewCtrl::GetSelection, 320
- wxDataViewCtrl::GetSelections, 320
- wxDataViewCtrl::IsSelected, 321
- wxDataViewCtrl::SetSelection, 321
- wxDataViewCtrl::SetSelectionRange, 321
- wxDataViewCtrl::SetSelections, 321
- wxDataViewCtrl::Unselect, 321
- wxDataViewCtrl::wxDataViewCtrl, 318
- wxDataViewCustomRenderer, 333
- wxDataViewCustomRenderer::~wxDataViewCustomRenderer, 333
- wxDataViewCustomRenderer::Activate, 333
- wxDataViewCustomRenderer::GetDC, 333
- wxDataViewCustomRenderer::GetSize, 333
- wxDataViewCustomRenderer::LeftClick, 334
- wxDataViewCustomRenderer::Render, 334
- wxDataViewCustomRenderer::RightClick, 334
- wxDataViewCustomRenderer::StartDrag, 334
- wxDataViewCustomRenderer::wxDataViewCustomRenderer, 333
- wxDataViewDateRenderer, 333
- wxDataViewDateRenderer::wxDataViewDateRenderer, 333
- wxDataViewEvent, 321, 322
- wxDataViewEvent::Clone, 322
- wxDataViewEvent::GetColumn, 322
- wxDataViewEvent::GetDataViewColumn, 322
- wxDataViewEvent::GetModel, 322
- wxDataViewEvent::GetRow, 322
- wxDataViewEvent::GetValue, 322
- wxDataViewEvent::IsEditCancelled, 322
- wxDataViewEvent::SetColumn, 322
- wxDataViewEvent::SetDataViewColumn, 322
- wxDataViewEvent::SetEditCanceled, 322
- wxDataViewEvent::SetModel, 322
- wxDataViewEvent::SetRow, 323
- wxDataViewEvent::SetValue, 323
- wxDataViewEvent::wxDataViewEvent, 321
- wxDataViewListModel, 326
- wxDataViewListModel::~wxDataViewListModel, 326
- wxDataViewListModel::AddNotifier, 326
- wxDataViewListModel::AddViewingColumn, 326
- wxDataViewListModel::Cleared, 326
- wxDataViewListModel::GetColType, 327
- wxDataViewListModel::GetNumberOfCols, 327
- wxDataViewListModel::GetNumberOfRows, 327
- wxDataViewListModel::GetValue, 327
- wxDataViewListModel::RemoveNotifier, 327
- wxDataViewListModel::RemoveViewingColumn, 327
- wxDataViewListModel::RowAppended, 327
- wxDataViewListModel::RowChanged, 327
- wxDataViewListModel::RowDeleted, 328
- wxDataViewListModel::RowInserted, 328
- wxDataViewListModel::RowPrepended, 328
- wxDataViewListModel::RowsReordered, 328
- wxDataViewListModel::SetValue, 328
- wxDataViewListModel::ValueChanged, 328
- wxDataViewListModel::wxDataViewListModel, 326
- wxDataViewListModelNotifier, 323
- wxDataViewListModelNotifier::~wxDataViewListModelNotifier, 323
- wxDataViewListModelNotifier::Cleared, 323
- wxDataViewListModelNotifier::GetOwner, 323
- wxDataViewListModelNotifier::RowAppended, 323
- wxDataViewListModelNotifier::RowChanged, 324
- wxDataViewListModelNotifier::RowDeleted, 324
- wxDataViewListModelNotifier::RowInserted, 324
- wxDataViewListModelNotifier::RowPrepended, 324
- wxDataViewListModelNotifier::RowsReordered,

- 324
- wxDataViewListModelNotifier::SetOwner, 324
- wxDataViewListModelNotifier::ValueChanged, 324
- wxDataViewListModelNotifier::wxDataViewListModelNotifier, 323
- wxDataViewModel, 325
- wxDataViewModel::~wxDataViewModel, 325
- wxDataViewModel::wxDataViewModel, 325
- wxDataViewProgressRenderer, 332
- wxDataViewProgressRenderer::wxDataViewProgressRenderer, 332
- wxDataViewRenderer, 330
- wxDataViewRenderer::GetMode, 330
- wxDataViewRenderer::GetOwner, 330
- wxDataViewRenderer::GetValue, 330
- wxDataViewRenderer::GetVariantType, 331
- wxDataViewRenderer::SetOwner, 331
- wxDataViewRenderer::SetValue, 331
- wxDataViewRenderer::Validate, 331
- wxDataViewRenderer::wxDataViewRenderer, 330
- wxDataViewSortedListModel, 329
- wxDataViewSortedListModel::~wxDataViewSortedListModel, 329
- wxDataViewSortedListModel::GetAscending, 329
- wxDataViewSortedListModel::Resort, 329
- wxDataViewSortedListModel::SetAscending, 329
- wxDataViewSortedListModel::wxDataViewSortedListModel, 329
- wxDataViewTextRenderer, 331
- wxDataViewTextRenderer::wxDataViewTextRenderer, 331
- wxDataViewToggleRenderer, 332
- wxDataViewToggleRenderer::wxDataViewToggleRenderer, 332
- wxDateEvent::GetDate, 340
- wxDateEvent::SetDate, 340
- wxDatePickerCtrl, 341
- wxDatePickerCtrl::Create, 341
- wxDatePickerCtrl::GetRange, 342
- wxDatePickerCtrl::GetValue, 342
- wxDatePickerCtrl::SetFormat, 343
- wxDatePickerCtrl::SetRange, 343
- wxDatePickerCtrl::SetValue, 343
- wxDatePickerCtrl::wxDatePickerCtrl, 341
- wxDateSpan, 344
- wxDateSpan::Add, 344
- wxDateSpan::Day, 344
- wxDateSpan::Days, 345
- wxDateSpan::GetDays, 345
- wxDateSpan::GetMonths, 345
- wxDateSpan::GetTotalDays, 345
- wxDateSpan::GetWeeks, 345
- wxDateSpan::GetYears, 346
- wxDateSpan::Month, 346
- wxDateSpan::Months, 346
- wxDateSpan::Multiply, 346
- wxDateSpan::Neg, 346
- wxDateSpan::Negate, 346
- wxDateSpan::operator!=, 348
- wxDateSpan::operator==, 348
- wxDateSpan::SetDays, 347
- wxDateSpan::SetMonths, 347
- wxDateSpan::SetWeeks, 347
- wxDateSpan::SetYears, 347
- wxDateSpan::Subtract, 347
- wxDateSpan::Week, 347
- wxDateSpan::Weeks, 348
- wxDateSpan::wxDateSpan, 344
- wxDateSpan::Year, 348
- wxDateSpan::Years, 348
- wxDateTime, 360, 361
- wxDateTime and Holidays, 2056
- wxDateTime characteristics, 2053
- wxDateTime::Add, 367
- wxDateTime::ConvertYearToBC, 356
- wxDateTime::Format, 370
- wxDateTime::FormatDate, 370
- wxDateTime::FormatISODate, 370
- wxDateTime::FormatISOTime, 370
- wxDateTime::FormatTime, 370
- wxDateTime::FromTimezone, 374
- wxDateTime::GetAmPmStrings, 356
- wxDateTime::GetAsDOS, 366
- wxDateTime::GetBeginDST, 356
- wxDateTime::GetCentury, 357, 364
- wxDateTime::GetCountry, 356
- wxDateTime::GetCurrentMonth, 357
- wxDateTime::GetCurrentYear, 357
- wxDateTime::GetDateOnly, 363
- wxDateTime::GetDay, 364
- wxDateTime::GetDayOfYear, 365
- wxDateTime::GetEndDST, 357
- wxDateTime::GetHour, 364
- wxDateTime::GetJDN, 373
- wxDateTime::GetJulianDayNumber, 373
- wxDateTime::GetLastMonthDay, 372
- wxDateTime::GetLastWeekDay, 372
- wxDateTime::GetMillisecond, 365
- wxDateTime::GetMinute, 364
- wxDateTime::GetMJD, 373
- wxDateTime::GetModifiedJulianDayNumber, 373
- wxDateTime::GetMonth, 364
- wxDateTime::GetMonthName, 357
- wxDateTime::GetNextWeekDay, 371
- wxDateTime::GetNumberOfDays, 357
- wxDateTime::GetPreWeekDay, 371
- wxDateTime::GetRataDie, 373
- wxDateTime::GetSecond, 364
- wxDateTime::GetTicks, 364
- wxDateTime::GetTimeNow, 358
- wxDateTime::GetTm, 363
- wxDateTime::GetTmNow, 358
- wxDateTime::GetWeekDay, 364, 372
- wxDateTime::GetWeekDayInSameWeek, 371
- wxDateTime::GetWeekDayName, 358
- wxDateTime::GetWeekOfMonth, 365
- wxDateTime::GetWeekOfYear, 365
- wxDateTime::GetYear, 364
- wxDateTime::GetYearDay, 373
- wxDateTime::IsBetween, 366
- wxDateTime::IsDST, 374
- wxDateTime::IsDSTApplicable, 359



- wxDateTime::IsEarlierThan, 366
- wxDateTime::IsEqualTo, 366
- wxDateTime::IsEqualUpTo, 367
- wxDateTime::IsGregorianDate, 365
- wxDateTime::IsLaterThan, 366
- wxDateTime::IsLeapYear, 358
- wxDateTime::IsSameDate, 367
- wxDateTime::IsSameTime, 367
- wxDateTime::IsStrictlyBetween, 366
- wxDateTime::IsValid, 363
- wxDateTime::IsWestEuropeanCountry, 359
- wxDateTime::IsWorkDay, 365
- wxDateTime::MakeFromTimezone, 374
- wxDateTime::MakeTimezone, 374
- wxDateTime::MakeUTC, 374
- wxDateTime::Now, 359
- wxDateTime::operator=, 363
- wxDateTime::ParseDate, 369
- wxDateTime::ParseDateTime, 369
- wxDateTime::ParseFormat, 368
- wxDateTime::ParseRfc822Date, 368
- wxDateTime::ParseTime, 369
- wxDateTime::ResetTime, 362
- wxDateTime::Set, 361, 362
- wxDateTime::SetCountry, 359
- wxDateTime::SetDay, 362
- wxDateTime::SetFromDOS, 366
- wxDateTime::SetHour, 362
- wxDateTime::SetMillisecond, 363
- wxDateTime::SetMinute, 362
- wxDateTime::SetMonth, 362
- wxDateTime::SetSecond, 363
- wxDateTime::SetToCurrent, 361
- wxDateTime::SetToLastMonthDay, 372
- wxDateTime::SetToLastWeekDay, 372
- wxDateTime::SetToNextWeekDay, 371
- wxDateTime::SetToPrevWeekDay, 371
- wxDateTime::SetToWeekDay, 371
- wxDateTime::SetToWeekDayInSameWeek, 370
- wxDateTime::SetToWeekOfYear, 372
- wxDateTime::SetToYearDay, 373
- wxDateTime::SetYear, 362
- wxDateTime::Subtract, 367, 368
- wxDateTime::Today, 359
- wxDateTime::ToTimezone, 374
- wxDateTime::ToUTC, 374
- wxDateTime::UNow, 360
- wxDateTime::wxDateTime, 360, 361
- wxDb, 382
- wxDb/wxDbTable wxODBC Overview, 2153
- wxDb::Catalog, 383
- wxDb::Close, 384
- wxDb::CommitTrans, 384
- wxDb::CreateView, 385
- wxDb::Dbms, 386
- wxDb::DispAllErrors, 386
- wxDb::DispNextError, 387
- wxDb::DropView, 388
- wxDb::EscapeSqlChars, 388
- wxDb::ExecSql, 388
- wxDb::FwdOnlyCursors, 389
- wxDb::GetCatalog, 389
- wxDb::GetColumnCount, 390
- wxDb::GetColumns, 390
- wxDb::GetData, 392
- wxDb::GetDatabaseName, 393
- wxDb::GetDatasourceName, 393
- wxDb::GetHDBC, 393
- wxDb::GetHENV, 393
- wxDb::GetHSTMT, 393
- wxDb::GetKeyFields, 394
- wxDb::GetNext, 394
- wxDb::GetNextError, 394
- wxDb::GetPassword, 395
- wxDb::GetTableCount, 395
- wxDb::GetUsername, 395
- wxDb::Grant, 395
- wxDb::IsFwdOnlyCursors, 396
- wxDb::IsOpen, 397
- wxDb::LogError, 397
- wxDb::ModifyColumn, 398
- wxDb::Open, 399
- wxDb::RollbackTrans, 401
- wxDb::SetDebugErrorMessage, 401
- wxDb::SetSqlLogging, 402
- wxDb::SQLColumnName, 402
- wxDb::SQLTableName, 403
- wxDb::TableExists, 403
- wxDb::TablePrivileges, 404
- wxDb::TranslateSqlState, 405
- wxDb::WriteSqlLog, 406
- wxDb::wxDb, 382
- wxDbCloseConnections, 380
- wxDbColDef::Initialize, 407
- wxDbColFor::Format, 408
- wxDbColFor::Initialize, 408
- wxDbColInf::Initialize, 409
- wxDbConnectInf, 409
- wxDbConnectInf::~wxDbConnectInf, 411
- wxDbConnectInf::AllocHenv, 411
- wxDbConnectInf::FreeHenv, 411
- wxDbConnectInf::GetAuthStr, 411
- wxDbConnectInf::GetDefaultDir, 412
- wxDbConnectInf::GetDescription, 412
- wxDbConnectInf::GetDsn, 412
- wxDbConnectInf::GetFileType, 412
- wxDbConnectInf::GetHenv, 412
- wxDbConnectInf::GetPassword, 412
- wxDbConnectInf::GetUid, 413
- wxDbConnectInf::GetUserID, 413
- wxDbConnectInf::Initialize, 411
- wxDbConnectInf::SetAuthStr, 413
- wxDbConnectInf::SetDefaultDir, 413
- wxDbConnectInf::SetDescription, 413
- wxDbConnectInf::SetDsn, 413
- wxDbConnectInf::SetFileType, 413
- wxDbConnectInf::SetHenv, 414
- wxDbConnectInf::SetPassword, 414
- wxDbConnectInf::SetUid, 414
- wxDbConnectInf::SetUserID, 414
- wxDbConnectInf::wxDbConnectInf, 409
- wxDbConnectionsInUse, 381
- wxDbFreeConnection, 381
- wxDbGetConnection, 381

---

wxDboGetDataSource, 382  
wxDboGridColInfo, 452  
wxDboGridColInfo::~wxDboGridColInfo, 452  
wxDboGridColInfo::AddColInfo, 452  
wxDboGridColInfo::wxDboGridColInfo, 452  
wxDboGridTableBase, 455  
wxDboGridTableBase::AssignDbTable, 456  
wxDboGridTableBase::UpdateRow, 455  
wxDboGridTableBase::ValidateRow, 455  
wxDboGridTableBase::wxDboGridTableBase, 455  
wxDboInf::Initialize, 415  
wxDboLogExtendedErrorMsg, 382  
wxDboSqlLog, 382  
wxDboTable, 416  
wxDboTable::BuildDeleteStmt, 417  
wxDboTable::BuildSelectStmt, 418  
wxDboTable::BuildUpdateStmt, 418  
wxDboTable::BuildWhereClause, 419  
wxDboTable::CanSelectForUpdate, 420  
wxDboTable::CanUpdateByROWID, 420  
wxDboTable::ClearMemberVar, 421  
wxDboTable::ClearMemberVars, 422  
wxDboTable::CloseCursor, 422  
wxDboTable::Count, 422  
wxDboTable::CreateIndex, 423  
wxDboTable::CreateTable, 425  
wxDboTable::DB\_STATUS, 426  
wxDboTable::Delete, 426  
wxDboTable::DeleteCursor, 426  
wxDboTable::DeleteMatching, 426  
wxDboTable::DeleteWhere, 427  
wxDboTable::DropIndex, 428  
wxDboTable::DropTable, 429  
wxDboTable::From, 429  
wxDboTable::GetColDefs, 429  
wxDboTable::GetCursor, 430  
wxDboTable::GetDb, 430  
wxDboTable::GetFirst, 430  
wxDboTable::GetFromClause, 431  
wxDboTable::GetLast, 431  
wxDboTable::GetNewCursor, 431  
wxDboTable::GetNext, 432  
wxDboTable::GetNumberOfColumns, 432  
wxDboTable::GetOrderByClause, 432  
wxDboTable::GetPrev, 432  
wxDboTable::GetQueryTableName, 433  
wxDboTable::GetRowNum, 433  
wxDboTable::GetTableName, 433  
wxDboTable::GetTablePath, 433  
wxDboTable::GetWhereClause, 434  
wxDboTable::Insert, 434  
wxDboTable::IsColNull, 435  
wxDboTable::IsCursorClosedOnCommit, 435  
wxDboTable::IsQueryOnly, 435  
wxDboTable::Open, 436  
wxDboTable::operator --, 451  
wxDboTable::operator ++, 451  
wxDboTable::OrderBy, 437  
wxDboTable::Query, 437  
wxDboTable::QueryBySqlStmt, 439  
wxDboTable::QueryMatching, 440  
wxDboTable::QueryOnKeyFields, 441  
wxDboTable::Refresh, 442  
wxDboTable::SetColDefs, 443  
wxDboTable::SetColNull, 446  
wxDboTable::SetCursor, 445  
wxDboTable::SetFromClause, 446  
wxDboTable::SetOrderByClause, 447  
wxDboTable::SetQueryTimeout, 448  
wxDboTable::SetWhereClause, 448  
wxDboTable::Update, 449  
wxDboTable::UpdateWhere, 450  
wxDboTable::Where, 450  
wxDboTable::wxDboTable, 416, 417  
wxDboTableInf::Initialize, 451  
wxDc::Blit, 457  
wxDc::CalcBoundingBox, 458  
wxDc::Clear, 459  
wxDc::ComputeScaleAndOrigin, 459  
wxDc::CrossHair, 459  
wxDc::DestroyClippingRegion, 459  
wxDc::DeviceToLogicalX, 459  
wxDc::DeviceToLogicalXRel, 459  
wxDc::DeviceToLogicalY, 460  
wxDc::DeviceToLogicalYRel, 460  
wxDc::DrawArc, 460  
wxDc::DrawBitmap, 460  
wxDc::DrawCheckMark, 460  
wxDc::DrawCircle, 460  
wxDc::DrawEllipse, 461  
wxDc::DrawEllipticArc, 461  
wxDc::DrawIcon, 461  
wxDc::DrawLabel, 461  
wxDc::DrawLine, 462  
wxDc::DrawLines, 462  
wxDc::DrawPoint, 463  
wxDc::DrawPolygon, 462  
wxDc::DrawPolyPolygon, 463  
wxDc::DrawRectangle, 463  
wxDc::DrawRotatedText, 463  
wxDc::DrawRoundedRectangle, 464  
wxDc::DrawSpline, 464  
wxDc::DrawText, 464  
wxDc::EndDoc, 465  
wxDc::EndPage, 465  
wxDc::FloodFill, 465  
wxDc::GetBackground, 465  
wxDc::GetBackgroundMode, 465  
wxDc::GetBrush, 466  
wxDc::GetCharHeight, 466  
wxDc::GetCharWidth, 466  
wxDc::GetClippingBox, 466  
wxDc::GetFont, 466  
wxDc::GetLayoutDirection, 466  
wxDc::GetLogicalFunction, 467  
wxDc::GetMapMode, 467  
wxDc::GetMultiLineTextExtent, 467  
wxDc::GetPartialTextExtents, 467  
wxDc::GetPen, 468  
wxDc::GetPixel, 468  
wxDc::GetPPI, 468  
wxDc::GetSize, 468  
wxDc::GetSizeMM, 469  
wxDc::GetTextBackground, 469

---

---

wxDC::GetTextExtent, 469  
wxDC::GetTextForeground, 470  
wxDC::GetUserScale, 470  
wxDC::GradientFillConcentric, 470  
wxDC::GradientFillLinear, 470  
wxDC::IsOk, 472  
wxDC::LogicalToDeviceX, 471  
wxDC::LogicalToDeviceXRel, 471  
wxDC::LogicalToDeviceY, 471  
wxDC::LogicalToDeviceYRel, 471  
wxDC::MaxX, 471  
wxDC::MaxY, 471  
wxDC::MinX, 471  
wxDC::MinY, 471  
wxDC::ResetBoundingBox, 472  
wxDC::SetAxisOrientation, 472  
wxDC::SetBackground, 472  
wxDC::SetBackgroundMode, 472  
wxDC::SetBrush, 472  
wxDC::SetClippingRegion, 473  
wxDC::SetDeviceOrigin, 473  
wxDC::SetFont, 473  
wxDC::SetLayoutDirection, 473  
wxDC::SetLogicalFunction, 474  
wxDC::SetMapMode, 474  
wxDC::SetPalette, 475  
wxDC::SetPen, 475  
wxDC::SetTextBackground, 475  
wxDC::SetTextForeground, 475  
wxDC::SetUserScale, 476  
wxDC::StartDoc, 476  
wxDC::StartPage, 476  
wxDCCLipper, 477  
wxDCCLipper::wxDCCLipper, 477  
wxDD\_CHANGE\_DIR, 514  
wxDD\_DEFAULT\_STYLE, 513  
wxDD\_DIR\_MUST\_EXIST, 514  
wxDDECleanUp, 1954  
wxDDEClient, 477  
wxDDEClient::MakeConnection, 478  
wxDDEClient::OnMakeConnection, 478  
wxDDEClient::ValidHost, 478  
wxDDEClient::wxDDEClient, 477  
wxDDEConnection, 479  
wxDDEConnection::Advise, 480  
wxDDEConnection::Disconnect, 480  
wxDDEConnection::Execute, 480  
wxDDEConnection::OnAdvise, 480  
wxDDEConnection::OnDisconnect, 480  
wxDDEConnection::OnExecute, 480  
wxDDEConnection::OnPoke, 481  
wxDDEConnection::OnRequest, 481  
wxDDEConnection::OnStartAdvise, 481  
wxDDEConnection::OnStopAdvise, 481  
wxDDEConnection::Poke, 481  
wxDDEConnection::Request, 481  
wxDDEConnection::StartAdvise, 482  
wxDDEConnection::StopAdvise, 482  
wxDDEConnection::wxDDEConnection, 479  
wxDDEInitialize, 1954  
wxDDEServer, 482  
wxDDEServer::Create, 482  
wxDDEServer::OnAcceptConnection, 483  
wxDDEServer::wxDDEServer, 482  
WXDEBUG\_NEW, 1968, 1969  
wxDebugContext overview, 2074  
wxDebugContext::Check, 483  
wxDebugContext::Dump, 483  
wxDebugContext::GetCheckPrevious, 484  
wxDebugContext::GetDebugMode, 484  
wxDebugContext::GetLevel, 484  
wxDebugContext::GetStream, 484  
wxDebugContext::GetStreamBuf, 485  
wxDebugContext::HasStream, 485  
wxDebugContext::PrintClasses, 485  
wxDebugContext::PrintStatistics, 485  
wxDebugContext::SetCheckpoint, 486  
wxDebugContext::SetCheckPrevious, 486  
wxDebugContext::SetDebugMode, 486  
wxDebugContext::SetFile, 486  
wxDebugContext::SetLevel, 487  
wxDebugContext::SetStandardError, 487  
wxDebugContext::SetStream, 487  
wxDebugMsg, 1971  
wxDebugReport, 489  
wxDebugReport::~~wxDebugReport, 489  
wxDebugReport::AddAll, 489  
wxDebugReport::AddContext, 489  
wxDebugReport::AddCurrentContext, 489  
wxDebugReport::AddCurrentDump, 490  
wxDebugReport::AddDump, 490  
wxDebugReport::AddExceptionContext, 490  
wxDebugReport::AddExceptionDump, 490  
wxDebugReport::AddFile, 490  
wxDebugReport::AddText, 490  
wxDebugReport::DoAddCustomContext, 491  
wxDebugReport::DoAddExceptionInfo, 491  
wxDebugReport::DoAddLoadedModules, 491  
wxDebugReport::DoAddSystemInfo, 491  
wxDebugReport::GetDirectory, 491  
wxDebugReport::GetFile, 491  
wxDebugReport::GetFilesCount, 491  
wxDebugReport::GetReportName, 492  
wxDebugReport::IsOk, 492  
wxDebugReport::Process, 492  
wxDebugReport::RemoveFile, 492  
wxDebugReport::Reset, 492  
wxDebugReport::wxDebugReport, 489  
wxDebugReportCompress, 493  
wxDebugReportCompress::GetCompressedFileName, 493  
wxDebugReportCompress::wxDebugReportCompress, 493  
wxDebugReportPreview, 493  
wxDebugReportPreview::~~wxDebugReportPreview, 493  
wxDebugReportPreview::Show, 493  
wxDebugReportPreview::wxDebugReportPreview, 493  
wxDebugReportPreviewStd, 494  
wxDebugReportPreviewStd::Show, 494  
wxDebugReportPreviewStd::wxDebugReportPreviewStd, 494  
wxDebugReportUpload, 495

---

- 
- wxDebugReportUpload::OnServerReply, 495
  - wxDebugReportUpload::wxDebugReportUpload, 495
  - wxDEFAULT\_DIALOG\_STYLE, 497
  - wxDEFAULT\_FRAME\_STYLE, 682, 1047, 1052
  - wxDelegateRendererNative, 496
  - wxDelegateRendererNative::DrawXXX, 496
  - wxDelegateRendererNative::wxDelegateRenderNative, 495
  - wxDialog, 498
  - wxDialog::~wxDialog, 499
  - wxDialog::Centre, 499
  - wxDialog::Create, 499
  - wxDialog::CreateButtonSizer, 499
  - wxDialog::CreateSeparatedButtonSizer, 500
  - wxDialog::CreateStdDialogButtonSizer, 500
  - wxDialog::DoOK, 500
  - wxDialog::EndModal, 500
  - wxDialog::GetAffirmativeId, 501
  - wxDialog::GetEscapedId, 501
  - wxDialog::GetReturnCode, 501
  - wxDialog::GetToolBar, 501
  - wxDialog::Iconize, 501
  - wxDialog::IsIconized, 502
  - wxDialog::IsModal, 502
  - wxDialog::OnSysColourChanged, 502
  - wxDialog::SetAffirmativeId, 503
  - wxDialog::SetEscapedId, 503
  - wxDialog::SetIcon, 503
  - wxDialog::SetIcons, 503
  - wxDialog::SetModal, 504
  - wxDialog::SetReturnCode, 504
  - wxDialog::Show, 504
  - wxDialog::ShowModal, 505
  - wxDialog::wxDialog, 498
  - wxDIALOG\_EX\_CONTEXTHELP, 498
  - wxDIALOG\_EX\_METAL, 498
  - wxDIALOG\_NO\_PARENT, 498
  - wxDialUpEvent, 505
  - wxDialUpEvent::IsConnectedEvent, 505
  - wxDialUpEvent::IsOwnEvent, 506
  - wxDialUpEvent::wxDialUpEvent, 505
  - wxDialUpManager::~wxDialUpManager, 507
  - wxDialUpManager::CancelDialing, 508
  - wxDialUpManager::Create, 506
  - wxDialUpManager::Dial, 507
  - wxDialUpManager::DisableAutoCheckOnlineStatus, 509
  - wxDialUpManager::EnableAutoCheckOnlineStatus, 508
  - wxDialUpManager::GetISPNames, 507
  - wxDialUpManager::HangUp, 508
  - wxDialUpManager::IsAlwaysOnline, 508
  - wxDialUpManager::IsDialing, 507
  - wxDialUpManager::IsOk, 507
  - wxDialUpManager::IsOnline, 508
  - wxDialUpManager::SetConnectCommand, 509
  - wxDialUpManager::SetOnlineStatus, 508
  - wxDialUpManager::SetWellKnownHost, 509
  - wxDi, 510, 511
  - wxDi::~wxDi, 511
  - wxDi::Exists, 511
  - wxDi::FindFirst, 511
  - wxDi::GetAllFiles, 511
  - wxDi::GetFirst, 511
  - wxDi::GetName, 512
  - wxDi::GetNext, 512
  - wxDi::GetTotalSize, 512
  - wxDi::HasFiles, 512
  - wxDi::HasSubDirs, 512
  - wxDi::IsOpened, 512
  - wxDi::Open, 513
  - wxDi::Traverse, 513
  - wxDi::wxDi, 510
  - wxDIRCTRL\_3D\_INTERNAL, 710
  - wxDIRCTRL\_DIR\_ONLY, 710
  - wxDIRCTRL\_EDIT\_LABELS, 710
  - wxDIRCTRL\_SELECT\_FIRST, 710
  - wxDIRCTRL\_SHOW\_FILTERS, 710
  - wxDiDialog, 514
  - wxDiDialog overview, 2131
  - wxDiDialog::~wxDiDialog, 515
  - wxDiDialog::GetMessage, 515
  - wxDiDialog::GetPath, 515
  - wxDiDialog::SetMessage, 515
  - wxDiDialog::SetPath, 515
  - wxDiDialog::ShowModal, 515
  - wxDiDialog::wxDiDialog, 514
  - wDiExists, 1922
  - wDIRP\_CHANGE\_DIR, 516
  - wDIRP\_DEFAULT\_STYLE, 516
  - wDIRP\_DIR\_MUST\_EXIST, 516
  - wDIRP\_USE\_TEXTCTRL, 516
  - wDiPickerCtrl, 517
  - wDiPickerCtrl::Create, 517
  - wDiPickerCtrl::GetPath, 518
  - wDiPickerCtrl::SetPath, 518
  - wDiPickerCtrl::wDiPickerCtrl, 516
  - wDiSelector, 1935
  - wDiTraverser::OnDir, 519
  - wDiTraverser::OnFile, 519
  - wDiTraverser::OnOpenError, 519
  - wDisplay, 520
  - wDisplay::~wDisplay, 520
  - wDisplay::ChangeMode, 520
  - wDisplay::GetClientArea, 520
  - wDisplay::GetCount, 521
  - wDisplay::GetCurrentMode, 521
  - wDisplay::GetDepth, 521
  - wDisplay::GetFromPoint, 521
  - wDisplay::GetFromWindow, 521
  - wDisplay::GetGeometry, 522
  - wDisplay::GetModes, 522
  - wDisplay::GetName, 522
  - wDisplay::IsPrimary, 522
  - wDisplay::wDisplay, 520
  - wDisplayDepth, 1945
  - wDisplaySize, 1945
  - wDisplaySizeMM, 1945
  - wDllLoader::GetDllExt, 524
  - wDllLoader::GetProgramHandle, 524
  - wDllLoader::GetSymbol, 524
  - wDllLoader::LoadLibrary, 524
  - wDllLoader::UnloadLibrary, 525
-

- wxDocChildFrame, 526
- wxDocChildFrame::~wxDocChildFrame, 526
- wxDocChildFrame::GetDocument, 526
- wxDocChildFrame::GetView, 526
- wxDocChildFrame::m\_childDocument, 525
- wxDocChildFrame::m\_childView, 525
- wxDocChildFrame::OnActivate, 526
- wxDocChildFrame::OnCloseWindow, 526
- wxDocChildFrame::SetDocument, 526
- wxDocChildFrame::SetView, 526
- wxDocChildFrame::wxDocChildFrame, 526
- wxDocManager, 528
- wxDocManager overview, 2135
- wxDocManager::~wxDocManager, 528
- wxDocManager::ActivateView, 528
- wxDocManager::AddDocument, 529
- wxDocManager::AddFileToHistory, 529
- wxDocManager::AssociateTemplate, 529
- wxDocManager::CloseDocuments, 529
- wxDocManager::CreateDocument, 529
- wxDocManager::CreateView, 529
- wxDocManager::DisassociateTemplate, 530
- wxDocManager::FileHistoryAddFilesToMenu, 530
- wxDocManager::FileHistoryLoad, 530
- wxDocManager::FileHistoryRemoveMenu, 530
- wxDocManager::FileHistorySave, 530
- wxDocManager::FileHistoryUseMenu, 530
- wxDocManager::FindTemplateForPath, 531
- wxDocManager::GetCurrentDocument, 531
- wxDocManager::GetCurrentView, 531
- wxDocManager::GetDocuments, 531
- wxDocManager::GetFileHistory, 531
- wxDocManager::GetHistoryFilesCount, 531
- wxDocManager::GetLastDirectory, 531
- wxDocManager::GetMaxDocsOpen, 531
- wxDocManager::GetTemplates, 531
- wxDocManager::Initialize, 532
- wxDocManager::m\_currentView, 527
- wxDocManager::m\_defaultDocumentNameCounter, 527
- wxDocManager::m\_docs, 528
- wxDocManager::m\_fileHistory, 527
- wxDocManager::m\_flags, 528
- wxDocManager::m\_lastDirectory, 528
- wxDocManager::m\_maxDocsOpen, 527
- wxDocManager::m\_templates, 528
- wxDocManager::MakeDefaultName, 532
- wxDocManager::MakeNewDocumentName, 532
- wxDocManager::OnCreateFileHistory, 532
- wxDocManager::OnFileClose, 533
- wxDocManager::OnFileCloseAll, 533
- wxDocManager::OnFileNew, 533
- wxDocManager::OnFileOpen, 533
- wxDocManager::OnFileRevert, 533
- wxDocManager::OnFileSave, 533
- wxDocManager::OnFileSaveAs, 533
- wxDocManager::RemoveDocument, 533
- wxDocManager::SelectDocumentPath, 534
- wxDocManager::SelectDocumentType, 534
- wxDocManager::SelectViewType, 534
- wxDocManager::SetLastDirectory, 535
- wxDocManager::SetMaxDocsOpen, 535
- wxDocManager::wxDocManager, 528
- wxDocMDIChildFrame, 536
- wxDocMDIChildFrame::~wxDocMDIChildFrame, 536
- wxDocMDIChildFrame::GetDocument, 536
- wxDocMDIChildFrame::GetView, 536
- wxDocMDIChildFrame::m\_childDocument, 536
- wxDocMDIChildFrame::m\_childView, 536
- wxDocMDIChildFrame::OnActivate, 536
- wxDocMDIChildFrame::OnCloseWindow, 537
- wxDocMDIChildFrame::SetDocument, 537
- wxDocMDIChildFrame::SetView, 537
- wxDocMDIChildFrame::wxDocMDIChildFrame, 536
- wxDocMDIParentFrame, 538
- wxDocMDIParentFrame::~wxDocMDIParentFrame, 538
- wxDocMDIParentFrame::Create, 538
- wxDocMDIParentFrame::OnCloseWindow, 538
- wxDocMDIParentFrame::wxDocMDIParentFrame, 537
- wxDocParentFrame, 539
- wxDocParentFrame::~wxDocParentFrame, 539
- wxDocParentFrame::Create, 539
- wxDocParentFrame::GetDocumentManager, 539
- wxDocParentFrame::OnCloseWindow, 540
- wxDocParentFrame::wxDocParentFrame, 539
- wxDocTemplate, 542
- wxDocTemplate overview, 2134
- wxDocTemplate::~wxDocTemplate, 543
- wxDocTemplate::CreateDocument, 543
- wxDocTemplate::CreateView, 543
- wxDocTemplate::GetDefaultExtension, 543
- wxDocTemplate::GetDescription, 543
- wxDocTemplate::GetDirectory, 544
- wxDocTemplate::GetDocumentManager, 544
- wxDocTemplate::GetDocumentName, 544
- wxDocTemplate::GetFileFilter, 544
- wxDocTemplate::GetFlags, 544
- wxDocTemplate::GetViewName, 544
- wxDocTemplate::InitDocument, 544
- wxDocTemplate::IsVisible, 544
- wxDocTemplate::m\_defaultExt, 540
- wxDocTemplate::m\_description, 540
- wxDocTemplate::m\_directory, 540
- wxDocTemplate::m\_docClassInfo, 541
- wxDocTemplate::m\_docTypeName, 541
- wxDocTemplate::m\_documentManager, 541
- wxDocTemplate::m\_fileFilter, 541
- wxDocTemplate::m\_flags, 541
- wxDocTemplate::m\_viewClassInfo, 541
- wxDocTemplate::m\_viewTypeName, 541
- wxDocTemplate::SetDefaultExtension, 544
- wxDocTemplate::SetDescription, 545
- wxDocTemplate::SetDirectory, 545
- wxDocTemplate::SetDocumentManager, 545
- wxDocTemplate::SetFileFilter, 545
- wxDocTemplate::SetFlags, 545
- wxDocTemplate::wxDocTemplate, 541
- wxDocument, 547
- wxDocument overview, 2133
- wxDocument::~wxDocument, 547

---

wxDocument::AddView, 547  
wxDocument::Close, 547  
wxDocument::DeleteAllViews, 547  
wxDocument::GetCommandProcessor, 547  
wxDocument::GetDocumentManager, 548  
wxDocument::GetDocumentName, 548  
wxDocument::GetDocumentTemplate, 547  
wxDocument::GetDocumentWindow, 548  
wxDocument::GetFilename, 548  
wxDocument::GetFirstView, 548  
wxDocument::GetPrintableName, 548  
wxDocument::GetTitle, 548  
wxDocument::GetViews, 549  
wxDocument::IsModified, 549  
wxDocument::LoadObject, 549  
wxDocument::m\_commandProcessor, 546  
wxDocument::m\_documentFile, 546  
wxDocument::m\_documentModified, 546  
wxDocument::m\_documentTemplate, 546  
wxDocument::m\_documentTitle, 546  
wxDocument::m\_documentTypeName, 546  
wxDocument::m\_documentViews, 546  
wxDocument::Modify, 549  
wxDocument::OnChangedViewList, 549  
wxDocument::OnCloseDocument, 550  
wxDocument::OnCreate, 550  
wxDocument::OnCreateCommandProcessor, 550  
wxDocument::OnNewDocument, 550  
wxDocument::OnOpenDocument, 550  
wxDocument::OnSaveDocument, 550  
wxDocument::OnSaveModified, 551  
wxDocument::RemoveView, 551  
wxDocument::Save, 551  
wxDocument::SaveAs, 551  
wxDocument::SaveObject, 551  
wxDocument::SetCommandProcessor, 551  
wxDocument::SetDocumentName, 551  
wxDocument::SetDocumentTemplate, 552  
wxDocument::SetFilename, 552  
wxDocument::SetTitle, 552  
wxDocument::UpdateAllViews, 552  
wxDocument::wxDocument, 547  
wxDos2UnixFilename, 1920  
wxDP\_ALLOWNONE, 341  
wxDP\_DEFAULT, 340  
wxDP\_DROPDOWN, 340  
wxDP\_SHOWCENTURY, 341  
wxDP\_SPIN, 340  
wxDragImage, 553, 554  
wxDragImage::BeginDrag, 554  
wxDragImage::DoDrawImage, 555  
wxDragImage::EndDrag, 555  
wxDragImage::GetImageRect, 556  
wxDragImage::Hide, 556  
wxDragImage::Move, 556  
wxDragImage::Show, 556  
wxDragImage::UpdateBackingFromWindow, 556  
wxDragImage::wxDragImage, 553  
wxDragResult, 558, 561  
wxDROP\_ICON, 1945  
wxDropFilesEvent, 557  
wxDropFilesEvent::GetFiles, 558  
wxDropFilesEvent::GetNumberOfFiles, 558  
wxDropFilesEvent::GetPosition, 558  
wxDropFilesEvent::m\_files, 557  
wxDropFilesEvent::m\_noFiles, 557  
wxDropFilesEvent::m\_pos, 558  
wxDropFilesEvent::wxDropFilesEvent, 557  
wxDropSource, 559  
wxDropSource::~wxDropSource, 559  
wxDropSource::DoDragDrop, 560  
wxDropSource::GetDataObject, 560  
wxDropSource::GiveFeedback, 560  
wxDropSource::SetCursor, 561  
wxDropSource::SetData, 560  
wxDropSource::wxDropSource, 559  
wxDropTarget, 562  
wxDropTarget::~wxDropTarget, 562  
wxDropTarget::GetData, 562  
wxDropTarget::OnData, 562  
wxDropTarget::OnDragOver, 563  
wxDropTarget::OnDrop, 562  
wxDropTarget::OnEnter, 563  
wxDropTarget::OnLeave, 564  
wxDropTarget::SetDataObject, 564  
wxDropTarget::wxDropTarget, 562  
wxDV\_MULTIPLE, 317  
wxDV\_SINGLE, 317  
wxDynamicCast, 1969  
wxDynamicCastThis, 1969  
wxDynamicLibrary, 564  
wxDynamicLibrary::CanonicalizedName, 564  
wxDynamicLibrary::CanonicalizePluginName, 565  
wxDynamicLibrary::Detach, 565  
wxDynamicLibrary::GetProgramHandle, 566  
wxDynamicLibrary::GetSymbol, 565  
wxDynamicLibrary::GetSymbolAorW, 565  
wxDynamicLibrary::HasSymbol, 566  
wxDynamicLibrary::IsLoaded, 566  
wxDynamicLibrary::ListLoaded, 566  
wxDynamicLibrary::Load, 566  
wxDynamicLibrary::Unload, 567  
wxDynamicLibrary::wxDynamicLibrary, 564  
wxDynamicLibraryDetails::GetAddress, 568  
wxDynamicLibraryDetails::GetName, 567  
wxDynamicLibraryDetails::GetPath, 568  
wxDynamicLibraryDetails::GetVersion, 568  
wxDYNLIB\_FUNCTION, 1951  
wxEdge, 938  
wxEmptyClipboard, 1949  
wxEnableTopLevelWindows, 1954  
wxEncodingConverter, 569  
wxEncodingConverter::CanConvert, 569  
wxEncodingConverter::Convert, 570  
wxEncodingConverter::GetAllEquivalents, 571  
wxEncodingConverter::GetPlatformEquivalents, 570  
wxEncodingConverter::Init, 569  
wxEncodingConverter::wxEncodingConverter, 569  
wxEndBusyCursor, 1937  
wxENTER\_CRIT\_SECT, 1918  
wxEntry, 1909, 1910

wxEntryCleanup, 1910  
wxEntryStart, 1910  
wxEnumClipboardFormats, 1949  
wxEraseEvent, 572  
wxEraseEvent::GetDC, 572  
wxEraseEvent::wxEraseEvent, 572  
wxError, 1971  
wxEvent, 573  
wxEvent::Clone, 573  
wxEvent::GetEventObject, 574  
wxEvent::GetEventType, 574  
wxEvent::GetId, 574  
wxEvent::GetSkipped, 574  
wxEvent::GetTimestamp, 574  
wxEvent::IsCommandEvent, 574  
wxEvent::m\_propagationLevel, 573  
wxEvent::ResumePropagation, 575  
wxEvent::SetEventObject, 575  
wxEvent::SetEventType, 575  
wxEvent::SetId, 575  
wxEvent::SetTimestamp, 575  
wxEvent::ShouldPropagate, 575  
wxEvent::Skip, 575  
wxEvent::StopPropagation, 576  
wxEvent::widthSash, 1519  
wxEvent::wxEvent, 573  
wxEvtHandler, 576  
wxEvtHandler::~~wxEvtHandler, 576  
wxEvtHandler::AddPendingEvent, 576  
wxEvtHandler::Connect, 577  
wxEvtHandler::Disconnect, 578  
wxEvtHandler::GetClientData, 579  
wxEvtHandler::GetClientObject, 579  
wxEvtHandler::GetEvtHandlerEnabled, 579  
wxEvtHandler::GetNextHandler, 580  
wxEvtHandler::GetPreviousHandler, 580  
wxEvtHandler::ProcessEvent, 580  
wxEvtHandler::SearchEventTable, 581  
wxEvtHandler::SetClientData, 582  
wxEvtHandler::SetClientObject, 582  
wxEvtHandler::SetEvtHandlerEnabled, 583  
wxEvtHandler::SetNextHandler, 583  
wxEvtHandler::SetPreviousHandler, 583  
wxEvtHandler::wxEvtHandler, 576  
wxExecute, 1913  
wxExit, 1915  
wxEXPAND, 1447  
wxEXPLICIT, 1952  
wxFAIL, 1981  
wxFAIL\_MSG, 1981  
wxFatalError, 1972  
wxFD\_CHANGE\_DIR, 600  
wxFD\_DEFAULT\_STYLE, 600  
wxFD\_FILE\_MUST\_EXIST, 600  
wxFD\_MULTIPLE, 600  
wxFD\_OPEN, 600  
wxFD\_OVERWRITE\_PROMPT, 600  
wxFD\_PREVIEW, 600  
wxFD\_SAVE, 600  
wxFFile, 584  
wxFFile::~~wxFFile, 585  
wxFFile::Attach, 585  
wxFFile::Close, 585  
wxFFile::Detach, 585  
wxFFile::Eof, 585  
wxFFile::Error, 586  
wxFFile::Flush, 586  
wxFFile::fp, 585  
wxFFile::GetKind, 586  
wxFFile::IsOpened, 586  
wxFFile::Length, 586  
wxFFile::Open, 587  
wxFFile::Read, 587  
wxFFile::ReadAll, 587  
wxFFile::Seek, 588  
wxFFile::SeekEnd, 588  
wxFFile::Tell, 588  
wxFFile::Write, 588, 589  
wxFFile::wxFFile, 584  
wxFFileInputStream, 589  
wxFFileInputStream::~~wxFFileInputStream, 589  
wxFFileInputStream::IsOk, 590  
wxFFileInputStream::wxFFileInputStream, 589  
wxFFileOutputStream, 590  
wxFFileOutputStream::~~wxFFileOutputStream, 590  
wxFFileOutputStream::IsOk, 591  
wxFFileOutputStream::wxFFileOutputStream, 590  
wxFFileStream, 591  
wxFFileStream::wxFFileStream, 591  
wxFile, 593  
wxFile::~~wxFile, 593  
wxFile::Access, 593  
wxFile::Attach, 594  
wxFile::Close, 594  
wxFile::Create, 594  
wxFile::Detach, 594  
wxFile::Eof, 594  
wxFile::Exists, 595  
wxFile::fd, 594  
wxFile::Flush, 595  
wxFile::GetKind, 595  
wxFile::IsOpened, 595  
wxFile::Length, 595  
wxFile::Open, 595  
wxFile::Read, 596  
wxFile::Seek, 596  
wxFile::SeekEnd, 596  
wxFile::Tell, 597  
wxFile::Write, 597  
wxFile::wxFile, 593  
wxFileConfig, 598  
wxFileConfig::Save, 598  
wxFileConfig::SetUmask, 598  
wxFileConfig::wxFileConfig, 598  
wxFileDataObject, 599  
wxFileDataObject::AddFile, 599  
wxFileDataObject::GetFilenames, 599  
wxFileDialog, 601  
wxFileDialog in PocketPC, 2241  
wxFileDialog overview, 2130  
wxFileDialog::~~wxFileDialog, 602  
wxFileDialog::GetDirectory, 602

- wxFileDialog::GetFilename, 602
- wxFileDialog::GetFileNames, 602
- wxFileDialog::GetFilterIndex, 603
- wxFileDialog::GetMessage, 603
- wxFileDialog::GetPath, 603
- wxFileDialog::GetPaths, 603
- wxFileDialog::GetWildcard, 603
- wxFileDialog::SetDirectory, 603
- wxFileDialog::SetFilename, 603
- wxFileDialog::SetFilterIndex, 603
- wxFileDialog::SetMessage, 604
- wxFileDialog::SetPath, 604
- wxFileDialog::SetWildcard, 604
- wxFileDialog::ShowModal, 604
- wxFileDialog::wxFileDialog, 601
- wxFileDirPickerEvent, 632
- wxFileDirPickerEvent::GetPath, 633
- wxFileDirPickerEvent::SetPath, 633
- wxFileDirPickerEvent::wxFileDirPickerEvent, 632
- wxFileDropTarget, 605
- wxFileDropTarget::OnDrop, 605
- wxFileDropTarget::OnDropFiles, 605
- wxFileDropTarget::wxFileDropTarget, 605
- wxFileExists, 1920
- wxFileHistory, 606
- wxFileHistory overview, 2136
- wxFileHistory::~~wxFileHistory, 606
- wxFileHistory::AddFilesToMenu, 607
- wxFileHistory::AddFileToHistory, 606
- wxFileHistory::GetBaseId, 607
- wxFileHistory::GetCount, 607
- wxFileHistory::GetHistoryFile, 607
- wxFileHistory::GetMaxFiles, 607
- wxFileHistory::GetMenus, 607
- wxFileHistory::Load, 607
- wxFileHistory::m\_fileHistory, 606
- wxFileHistory::m\_fileHistoryN, 606
- wxFileHistory::m\_fileMaxFiles, 606
- wxFileHistory::m\_fileMenu, 606
- wxFileHistory::RemoveFileFromHistory, 608
- wxFileHistory::RemoveMenu, 608
- wxFileHistory::Save, 608
- wxFileHistory::SetBaseId, 608
- wxFileHistory::UseMenu, 608
- wxFileHistory::wxFileHistory, 606
- wxFileInputStream, 609
- wxFileInputStream::~~wxFileInputStream, 609
- wxFileInputStream::IsOk, 609
- wxFileInputStream::wxFileInputStream, 609
- wxFileModificationTime, 1920
- wxFileName, 613
- wxFileName::AppendDir, 613
- wxFileName::Assign, 613
- wxFileName::AssignCwd, 614
- wxFileName::AssignDir, 614
- wxFileName::AssignHomeDir, 614
- wxFileName::AssignTempFileName, 614
- wxFileName::Clear, 614
- wxFileName::ClearExt, 615
- wxFileName::CreateTempFileName, 615
- wxFileName::DirExists, 615
- wxFileName::DirName, 615
- wxFileName::FileExists, 616
- wxFileName::FileName, 616
- wxFileName::GetCwd, 616
- wxFileName::GetDirCount, 616
- wxFileName::GetDirs, 616
- wxFileName::GetExt, 617
- wxFileName::GetForbiddenChars, 617
- wxFileName::GetFormat, 617
- wxFileName::GetFullName, 617
- wxFileName::GetFullPath, 617
- wxFileName::GetHomeDir, 617
- wxFileName::GetHumanReadableSize, 619
- wxFileName::GetLongPath, 617
- wxFileName::GetModificationTime, 617
- wxFileName::GetName, 617
- wxFileName::GetPath, 618
- wxFileName::GetPathSeparator, 618
- wxFileName::GetPathSeparators, 618
- wxFileName::GetPathTerminators, 618
- wxFileName::GetPathWithSep, 619
- wxFileName::GetShortPath, 619
- wxFileName::GetSize, 619
- wxFileName::GetTempDir, 619
- wxFileName::GetTimes, 620
- wxFileName::GetVolume, 620
- wxFileName::GetVolumeSeparator, 620
- wxFileName::HasExt, 620
- wxFileName::HasName, 620
- wxFileName::HasVolume, 620
- wxFileName::InsertDir, 621
- wxFileName::IsAbsolute, 621
- wxFileName::IsCaseSensitive, 621
- wxFileName::IsDir, 622
- wxFileName::IsDirReadable, 621
- wxFileName::IsDirWritable, 621
- wxFileName::IsFileExecutable, 621
- wxFileName::IsFileReadable, 622
- wxFileName::IsFileWritable, 622
- wxFileName::IsOk, 622
- wxFileName::IsPathSeparator, 622
- wxFileName::IsRelative, 622
- wxFileName::MacFindDefaultTypeAndCreator, 622
- wxFileName::MacRegisterDefaultTypeAndCreator, 623
- wxFileName::MacSetDefaultTypeAndCreator, 623
- wxFileName::MakeAbsolute, 623
- wxFileName::MakeRelativeTo, 623
- wxFileName::Mkdir, 624
- wxFileName::Normalize, 624
- wxFileName::operator!=, 628
- wxFileName::operator=, 628
- wxFileName::operator==, 628
- wxFileName::PrependDir, 625
- wxFileName::RemoveDir, 625
- wxFileName::RemoveLastDir, 625
- wxFileName::Rmdir, 625
- wxFileName::SameAs, 625
- wxFileName::SetCwd, 626
- wxFileName::SetEmptyExt, 626
- wxFileName::SetExt, 626



- wxFileName::SetFullName, 626
- wxFileName::SetName, 626
- wxFileName::SetTimes, 626
- wxFileName::SetVolume, 627
- wxFileName::SplitPath, 627
- wxFileName::SplitVolume, 627
- wxFileName::Touch, 627
- wxFileName::wxFileName, 613
- wxFileNameFromPath, 1920
- wxFileOutputStream, 629
- wxFileOutputStream::~~wxFileOutputStream, 629
- wxFileOutputStream::IsOk, 629
- wxFileOutputStream::wxFileOutputStream, 629
- wxFilePickerCtrl, 630
- wxFilePickerCtrl::Create, 631
- wxFilePickerCtrl::GetPath, 632
- wxFilePickerCtrl::SetPath, 632
- wxFilePickerCtrl::wxFilePickerCtrl, 630
- wxFileSelector, 1936
- wxFileStream, 633
- wxFileStream::wxFileStream, 633
- wxFileSystem, 634
- wxFileSystem::AddHandler, 634
- wxFileSystem::ChangePathTo, 634
- wxFileSystem::FileNameToURL, 635
- wxFileSystem::FindFileInPath, 635
- wxFileSystem::FindFirst, 635
- wxFileSystem::FindNext, 636
- wxFileSystem::GetPath, 635
- wxFileSystem::HasHandlerForPath, 634
- wxFileSystem::OpenFile, 636
- wxFileSystem::URLToFileName, 636
- wxFileSystem::wxFileSystem, 634
- wxFileSystemHandler, 637
- wxFileSystemHandler::CanOpen, 637
- wxFileSystemHandler::FindFirst, 638
- wxFileSystemHandler::FindNext, 638
- wxFileSystemHandler::GetAnchor, 637
- wxFileSystemHandler::GetLeftLocation, 638
- wxFileSystemHandler::GetMimeTypeFromExt, 638
- wxFileSystemHandler::GetProtocol, 638
- wxFileSystemHandler::GetRightLocation, 638
- wxFileSystemHandler::OpenFile, 639
- wxFileSystemHandler::wxFileSystemHandler, 637
- wxFileType, 641
- wxFileType::~~wxFileType, 641
- wxFileType::ExpandCommand, 643
- wxFileType::GetDescription, 642
- wxFileType::GetExtensions, 641
- wxFileType::GetIcon, 642
- wxFileType::GetMimeType, 641
- wxFileType::GetMimeType, 641
- wxFileType::GetOpenCommand, 642
- wxFileType::GetPrintCommand, 642
- wxFileType::wxFileType, 641
- wxFilterClassFactory::CanHandle, 644
- wxFilterClassFactory::Find, 644
- wxFilterClassFactory::GetFirst/GetNext, 644
- wxFilterClassFactory::GetProtocol, 645
- wxFilterClassFactory::GetProtocols, 645
- wxFilterClassFactory::NewStream, 645
- wxFilterClassFactory::PopExtension, 646
- wxFilterClassFactory::PushFront, 646
- wxFilterClassFactory::Remove, 646
- wxFilterInputStream, 647
- wxFilterInputStream::wxFilterInputStream, 647
- wxFilterOutputStream, 647, 648
- wxFilterOutputStream::wxFilterOutputStream, 647
- wxFindDialogEvent, 648
- wxFindDialogEvent::GetDialog, 649
- wxFindDialogEvent::GetFindString, 649
- wxFindDialogEvent::GetFlags, 648
- wxFindDialogEvent::GetReplaceString, 649
- wxFindDialogEvent::wxFindDialogEvent, 648
- wxFindFirstFile, 1920
- wxFindMenuItemId, 1955
- wxFindNextFile, 1921
- wxFindReplaceData, 650
- wxFindReplaceData::GetFindString, 650
- wxFindReplaceData::GetFlags, 650
- wxFindReplaceData::GetReplaceString, 650
- wxFindReplaceData::SetFindString, 651
- wxFindReplaceData::SetFlags, 650
- wxFindReplaceData::SetReplaceString, 651
- wxFindReplaceData::wxFindReplaceData, 650
- wxFindReplaceDialog, 651
- wxFindReplaceDialog::~~wxFindReplaceDialog, 651
- wxFindReplaceDialog::Create, 651
- wxFindReplaceDialog::GetData, 652
- wxFindReplaceDialog::wxFindReplaceDialog, 651
- wxFindWindowAtPoint, 1955
- wxFindWindowAtPointer, 1956
- wxFindWindowByLabel, 1955
- wxFindWindowByName, 1955
- wxFinite, 1944
- wxFIXED\_MINSIZE, 1447
- wxFlexGridSizer, 652
- wxFlexGridSizer::AddGrowableCol, 653
- wxFlexGridSizer::AddGrowableRow, 653
- wxFlexGridSizer::GetFlexibleDirection, 653
- wxFlexGridSizer::GetNonFlexibleGrowMode, 653
- wxFlexGridSizer::RemoveGrowableCol, 654
- wxFlexGridSizer::RemoveGrowableRow, 654
- wxFlexGridSizer::SetFlexibleDirection, 654
- wxFlexGridSizer::SetNonFlexibleGrowMode, 654
- wxFlexGridSizer::wxFlexGridSizer, 652
- wxFLP\_CHANGE\_DIR, 630
- wxFLP\_DEFAULT\_STYLE, 629
- wxFLP\_FILE\_MUST\_EXIST, 630
- wxFLP\_OPEN, 630
- wxFLP\_OVERWRITE\_PROMPT, 630
- wxFLP\_SAVE, 630
- wxFLP\_USE\_TEXTCTRL, 630
- wxFNTP\_DEFAULT\_STYLE, 679
- wxFNTP\_FONTDESC\_AS\_LABEL, 679
- wxFNTP\_USE\_TEXTCTRL, 679
- wxFNTP\_USEFONT\_FOR\_LABEL, 679
- wxFocusEvent, 655
- wxFocusEvent::GetWindow, 655

---

wxFocusEvent::wxFocusEvent, 655  
wxFont, 658, 659  
wxFont::~~wxFont, 660  
wxFont::GetDefaultEncoding, 661  
wxFont::GetFaceName, 661  
wxFont::GetFamily, 661  
wxFont::GetNativeFontInfoDesc, 661  
wxFont::GetNativeFontInfoUserDesc, 662  
wxFont::GetPointSize, 662  
wxFont::GetStyle, 662  
wxFont::GetUnderlined, 662  
wxFont::GetWeight, 662  
wxFont::IsFixedWidth, 661  
wxFont::IsOk, 663  
wxFont::New, 663  
wxFont::operator !=, 668  
wxFont::operator =, 667  
wxFont::operator ==, 668  
wxFont::SetDefaultEncoding, 663  
wxFont::SetFaceName, 663  
wxFont::SetFamily, 664  
wxFont::SetNativeFontInfo, 664  
wxFont::SetNativeFontInfoUserDesc, 665  
wxFont::SetPointSize, 666  
wxFont::SetStyle, 666  
wxFont::SetUnderlined, 667  
wxFont::SetWeight, 667  
wxFont::wxFont, 658  
wxFontData, 668  
wxFontData::EnableEffects, 668  
wxFontData::GetAllowSymbols, 669  
wxFontData::GetChosenFont, 669  
wxFontData::GetColour, 669  
wxFontData::GetEnableEffects, 669  
wxFontData::GetInitialFont, 669  
wxFontData::GetShowHelp, 669  
wxFontData::operator =, 670  
wxFontData::SetAllowSymbols, 669  
wxFontData::SetChosenFont, 670  
wxFontData::SetColour, 670  
wxFontData::SetInitialFont, 670  
wxFontData::SetRange, 670  
wxFontData::SetShowHelp, 670  
wxFontData::wxFontData, 668  
wxFontDialog, 671  
wxFontDialog overview, 2129  
wxFontDialog::Create, 671  
wxFontDialog::GetFontData, 671  
wxFontDialog::ShowModal, 671  
wxFontDialog::wxFontDialog, 671  
wxFontEnumerator::EnumerateEncodings, 673  
wxFontEnumerator::EnumerateFacenames, 672  
wxFontEnumerator::GetEncodings, 673  
wxFontEnumerator::GetFacenames, 673  
wxFontEnumerator::IsValidFacename, 673  
wxFontEnumerator::OnFacename, 673  
wxFontEnumerator::OnFontEncoding, 673  
wxFontList, 674  
wxFontList::FindOrCreateFont, 674  
wxFontList::wxFontList, 674  
wxFontMapper, 675  
wxFontMapper::~~wxFontMapper, 675  
wxFontMapper::CharsetToEncoding, 676  
wxFontMapper::Get, 676  
wxFontMapper::GetAllEncodingNames, 676  
wxFontMapper::GetAltForEncoding, 676  
wxFontMapper::GetEncoding, 676  
wxFontMapper::GetEncodingDescription, 677  
wxFontMapper::GetEncodingFromName, 677  
wxFontMapper::GetEncodingName, 677  
wxFontMapper::GetSupportedEncodingsCount, 677  
wxFontMapper::IsEncodingAvailable, 677  
wxFontMapper::Set, 678  
wxFontMapper::SetConfig, 678  
wxFontMapper::SetConfigPath, 678  
wxFontMapper::SetDialogParent, 677  
wxFontMapper::SetDialogTitle, 678  
wxFontMapper::wxFontMapper, 675  
wxFontPickerCtrl, 679  
wxFontPickerCtrl::Create, 679  
wxFontPickerCtrl::GetMaxPointSize, 681  
wxFontPickerCtrl::GetSelectedFont, 680  
wxFontPickerCtrl::SetMaxPointSize, 681  
wxFontPickerCtrl::SetSelectedFont, 680  
wxFontPickerCtrl::wxFontPickerCtrl, 679  
wxFontPickerEvent, 681  
wxFontPickerEvent::GetFont, 681  
wxFontPickerEvent::SetFont, 682  
wxFontPickerEvent::wxFontPickerEvent, 681  
wxFrame, 684  
wxFrame::~~wxFrame, 685  
wxFrame::Centre, 685  
wxFrame::Create, 685  
wxFrame::CreateStatusBar, 685  
wxFrame::CreateToolBar, 686  
wxFrame::GetClientAreaOrigin, 687  
wxFrame::GetMenuBar, 687  
wxFrame::GetStatusBar, 687  
wxFrame::GetStatusBarPane, 688  
wxFrame::GetToolBar, 688  
wxFrame::OnCreateStatusBar, 688  
wxFrame::OnCreateToolBar, 689  
wxFrame::ProcessCommand, 689  
wxFrame::SendSizeEvent, 690  
wxFrame::SetMenuBar, 690  
wxFrame::SetStatusBar, 690  
wxFrame::SetStatusBarPane, 690  
wxFrame::SetStatusText, 691  
wxFrame::SetStatusWidths, 691  
wxFrame::SetToolBar, 692  
wxFrame::wxFrame, 684  
wxFRAME\_EX\_CONTEXTHELP, 683  
wxFRAME\_EX\_METAL, 683  
wxFRAME\_FLOAT\_ON\_PARENT, 683  
wxFRAME\_NO\_TASKBAR, 683  
wxFRAME\_NO\_WINDOW\_MENU, 1053  
wxFRAME\_SHAPED, 683  
wxFRAME\_TOOL\_WINDOW, 683  
wxFSFile, 692  
wxFSFile::DetachStream, 693  
wxFSFile::GetAnchor, 693  
wxFSFile::GetLocation, 694  
wxFSFile::GetMimeType, 694

---

wxFSFile::GetModificationTime, 694  
wxFSFile::GetStream, 694  
wxFSFile::wxFSFile, 692  
wxFTP, 696  
wxFTP::~wxFTP, 696  
wxFTP::Abort, 696  
wxFTP::ChDir, 697  
wxFTP::CheckCommand, 696  
wxFTP::FileExists, 698  
wxFTP::GetDirList, 699  
wxFTP::GetFileSize, 699  
wxFTP::GetFilesList, 699  
wxFTP::GetInputStream, 700  
wxFTP::GetLastResult, 696  
wxFTP::GetOutputStream, 699  
wxFTP::MkDir, 697  
wxFTP::Pwd, 697  
wxFTP::Rename, 697  
wxFTP::RmDir, 697  
wxFTP::RmFile, 697  
wxFTP::SendCommand, 696  
wxFTP::SetAscii, 697  
wxFTP::SetBinary, 697  
wxFTP::SetPassive, 698  
wxFTP::SetPassword, 698  
wxFTP::SetTransferMode, 698  
wxFTP::SetUser, 698  
wxFTP::wxFTP, 696  
wxFULL\_REPAINT\_ON\_RESIZE, 1797  
wxGA\_HORIZONTAL, 701  
wxGA\_SMOOTH, 701  
wxGA\_VERTICAL, 701  
wxGauge, 701, 702  
wxGauge::~wxGauge, 702  
wxGauge::Create, 702  
wxGauge::GetBezelFace, 703  
wxGauge::GetRange, 703  
wxGauge::GetShadowWidth, 703  
wxGauge::GetValue, 703  
wxGauge::IsVertical, 704  
wxGauge::Pulse, 705  
wxGauge::SetBezelFace, 704  
wxGauge::SetRange, 704  
wxGauge::SetShadowWidth, 704  
wxGauge::SetValue, 704  
wxGauge::wxGauge, 701  
wxGBPosition, 705  
wxGBPosition::GetCol, 705  
wxGBPosition::GetRow, 705  
wxGBPosition::operator!, 706  
wxGBPosition::operator==, 706  
wxGBPosition::SetCol, 705  
wxGBPosition::SetRow, 706  
wxGBPosition::wxGBPosition, 705  
wxGBSizerItem, 706, 707  
wxGBSizerItem::GetEndPos, 707  
wxGBSizerItem::GetPos, 707  
wxGBSizerItem::GetSpan, 707  
wxGBSizerItem::Intersects, 707  
wxGBSizerItem::SetPos, 707  
wxGBSizerItem::SetSpan, 707  
wxGBSizerItem::wxGBSizerItem, 706  
wxGBSpan, 708  
wxGBSpan::GetColspan, 708  
wxGBSpan::GetRowspan, 708  
wxGBSpan::operator!, 709  
wxGBSpan::operator==, 709  
wxGBSpan::SetColspan, 708  
wxGBSpan::SetRowspan, 708  
wxGBSpan::wxGBSpan, 708  
wxGDIObject, 709  
wxGDIObject::wxGDIObject, 709  
wxGenericAboutBox, 1937  
wxGenericDirCtrl, 710  
wxGenericDirCtrl::~wxGenericDirCtrl, 711  
wxGenericDirCtrl::CollapsePath, 712  
wxGenericDirCtrl::CollapseTree, 711  
wxGenericDirCtrl::Create, 711  
wxGenericDirCtrl::ExpandPath, 712  
wxGenericDirCtrl::GetDefaultPath, 712  
wxGenericDirCtrl::GetFilePath, 712  
wxGenericDirCtrl::GetFilter, 712  
wxGenericDirCtrl::GetFilterIndex, 712  
wxGenericDirCtrl::GetFilterListCtrl, 712  
wxGenericDirCtrl::GetPath, 712  
wxGenericDirCtrl::GetRootId, 713  
wxGenericDirCtrl::GetTreeCtrl, 713  
wxGenericDirCtrl::Init, 711  
wxGenericDirCtrl::ReCreateTree, 713  
wxGenericDirCtrl::SetDefaultPath, 713  
wxGenericDirCtrl::SetFilter, 713  
wxGenericDirCtrl::SetFilterIndex, 713  
wxGenericDirCtrl::SetPath, 713  
wxGenericDirCtrl::ShowHidden, 713  
wxGenericDirCtrl::wxGenericDirCtrl, 710  
wxGenericValidator, 714, 715  
wxGenericValidator::~wxGenericValidator, 715  
wxGenericValidator::Clone, 715  
wxGenericValidator::TransferFromWindow, 715  
wxGenericValidator::TransferToWindow, 715  
wxGenericValidator::wxGenericValidator, 714  
wxGetActiveWindow, 1956  
wxGetApp, 1911  
wxGetBatteryState, 1956  
wxGetClientDisplayRect, 1945  
wxGetClipboardData, 1950  
wxGetClipboardFormatName, 1950  
wxGetColourFromUser, 1937  
wxGetCwd, 1923  
wxGetDiskSpace, 1921  
wxGetDisplayName, 1956  
wxGetDisplaySize, 1945  
wxGetDisplaySizeMM, 1945  
wxGetElapsedTime, 1977  
wxGetEmailAddress, 1926  
wxGetenv, 1983  
wxGetEnv, 1983  
wxGetFileKind, 1921  
wxGetFontFromUser, 1938  
wxGetFreeMemory, 1926  
wxGetFullHostName, 1926  
wxGetHomeDir, 1926  
wxGetHostName, 1927  
wxGetKeyState, 1952

- 
- wxGetLocalTime, 1977
  - wxGetLocalTimeMillis, 1978
  - wxGetMousePosition, 1957
  - wxGetMouseState, 1957
  - wxGetMultipleChoice, 1940
  - wxGetMultipleChoices, 1938, 1939
  - wxGetNumberFromUser, 1939
  - wxGetOsDescription, 1927
  - wxGetOSDirectory, 1922
  - wxGetOsVersion, 1927
  - wxGetPasswordFromUser, 1939
  - wxGetPowerType, 1956
  - wxGetPrinterCommand, 1947
  - wxGetPrinterFile, 1947
  - wxGetPrinterMode, 1947
  - wxGetPrinterOptions, 1947
  - wxGetPrinterOrientation, 1947
  - wxGetPrinterPreviewCommand, 1947
  - wxGetPrinterScaling, 1948
  - wxGetPrinterTranslation, 1948
  - wxGetProcessId, 1916
  - wxGetResource, 1958
  - wxGetSingleChoice, 1941
  - wxGetSingleChoiceData, 1942
  - wxGetSingleChoiceIndex, 1941
  - wxGetStockLabel, 1958
  - wxGetTempFileName, 1923
  - wxGetTextFromUser, 1940
  - wxGetTopLevelParent, 1959
  - wxGetTranslation, 1930
  - wxGetUserHome, 1928
  - wxGetUserId, 1928, 1929
  - wxGetUserName, 1929
  - wxGetUTCTime, 1978
  - wxGetVariantCast, 1779
  - wxGetWorkingDirectory, 1923
  - wxGLCanvas, 717
  - wxGLCanvas::GetContext, 719
  - wxGLCanvas::SetColour, 719
  - wxGLCanvas::SetCurrent, 719
  - wxGLCanvas::SwapBuffers, 719
  - wxGLCanvas::wxGLCanvas, 717
  - wxGLContext, 720
  - wxGLContext::SetCurrent, 721
  - wxGLContext::wxGLContext, 720
  - wxGraphicsContext::Clip, 723
  - wxGraphicsContext::ConcatTransform, 726
  - wxGraphicsContext::Create, 721
  - wxGraphicsContext::CreateBrush, 722
  - wxGraphicsContext::CreateFont, 723
  - wxGraphicsContext::CreateFromNative, 722
  - wxGraphicsContext::CreateFromNativeWindow, 722
  - wxGraphicsContext::CreateLinearGradientBrush, 722
  - wxGraphicsContext::CreateMatrix, 723
  - wxGraphicsContext::CreatePath, 723
  - wxGraphicsContext::CreatePen, 722
  - wxGraphicsContext::CreateRadialGradientBrush, 722
  - wxGraphicsContext::DrawBitmap, 723
  - wxGraphicsContext::DrawEllipse, 723
  - wxGraphicsContext::DrawIcon, 724
  - wxGraphicsContext::DrawLines, 724
  - wxGraphicsContext::DrawPath, 724
  - wxGraphicsContext::DrawRectangle, 724
  - wxGraphicsContext::DrawRoundedRectangle, 724
  - wxGraphicsContext::DrawText, 724
  - wxGraphicsContext::FillPath, 724
  - wxGraphicsContext::GetNativeContext, 725
  - wxGraphicsContext::GetPartialTextExtents, 725
  - wxGraphicsContext::GetTextExtent, 725
  - wxGraphicsContext::GetTransform, 725
  - wxGraphicsContext::ResetClip, 723
  - wxGraphicsContext::Rotate, 725
  - wxGraphicsContext::Scale, 725
  - wxGraphicsContext::SetBrush, 726
  - wxGraphicsContext::SetFont, 726
  - wxGraphicsContext::SetPen, 726
  - wxGraphicsContext::SetTransform, 726
  - wxGraphicsContext::StrokeLine, 726
  - wxGraphicsContext::StrokeLines, 726
  - wxGraphicsContext::StrokePath, 725
  - wxGraphicsContext::Translate, 725
  - wxGraphicsMatrix::Concat, 727
  - wxGraphicsMatrix::Get, 727
  - wxGraphicsMatrix::GetNativeMatrix, 728
  - wxGraphicsMatrix::Invert, 728
  - wxGraphicsMatrix::IsEqual, 728
  - wxGraphicsMatrix::IsIdentity, 728
  - wxGraphicsMatrix::Rotate, 728
  - wxGraphicsMatrix::Scale, 728
  - wxGraphicsMatrix::Set, 728
  - wxGraphicsMatrix::TransformDistance, 729
  - wxGraphicsMatrix::TransformPoint, 729
  - wxGraphicsMatrix::Translate, 728
  - wxGraphicsObject::GetRenderer, 729
  - wxGraphicsObject::IsNull, 729
  - wxGraphicsPath::AddArc, 730
  - wxGraphicsPath::AddArcToPoint, 730
  - wxGraphicsPath::AddCircle, 730
  - wxGraphicsPath::AddCurveToPoint, 730
  - wxGraphicsPath::AddEllipse, 731
  - wxGraphicsPath::AddLineToPoint, 731
  - wxGraphicsPath::AddPath, 731
  - wxGraphicsPath::AddQuadCurveToPoint, 731
  - wxGraphicsPath::AddRectangle, 731
  - wxGraphicsPath::AddRoundedRectangle, 731
  - wxGraphicsPath::CloseSubpath, 731
  - wxGraphicsPath::Contains, 732
  - wxGraphicsPath::GetBox, 732
  - wxGraphicsPath::GetCurrentPoint, 732
  - wxGraphicsPath::GetNativePath, 732
  - wxGraphicsPath::MoveToPoint, 730
  - wxGraphicsPath::Transform, 732
  - wxGraphicsPath::UnGetNativePath, 732
  - wxGraphicsRenderer::CreateBrush, 734
  - wxGraphicsRenderer::CreateContext, 733
  - wxGraphicsRenderer::CreateContextFromNativeContext, 733
  - wxGraphicsRenderer::CreateContextFromNativeWindow, 734
-

wxGraphicsRenderer::CreateFont, 734  
wxGraphicsRenderer::CreateLinearGradientBrush, 734  
wxGraphicsRenderer::CreateMatrix, 734  
wxGraphicsRenderer::CreatePath, 735  
wxGraphicsRenderer::CreatePen, 734  
wxGraphicsRenderer::CreateRadialGradientBrush, 734  
wxGraphicsRenderer::GetDefaultRenderer, 733  
wxGrid, 739  
wxGrid::~~wxGrid, 739  
wxGrid::AppendCols, 739  
wxGrid::AppendRows, 739  
wxGrid::AutoSize, 739  
wxGrid::AutoSizeColumn, 740  
wxGrid::AutoSizeColumns, 740  
wxGrid::AutoSizeRow, 740  
wxGrid::AutoSizeRows, 740  
wxGrid::BeginBatch, 740  
wxGrid::BlockToDeviceRect, 741  
wxGrid::CanDragColMove, 741  
wxGrid::CanDragColSize, 741  
wxGrid::CanDragGridSize, 741  
wxGrid::CanDragRowSize, 741  
wxGrid::CanEnableCellControl, 741  
wxGrid::CanHaveAttributes, 742  
wxGrid::CellToRect, 742  
wxGrid::ClearGrid, 742  
wxGrid::ClearSelection, 742  
wxGrid::CreateGrid, 742  
wxGrid::DeleteCols, 742  
wxGrid::DeleteRows, 743  
wxGrid::DisableCellEditControl, 743  
wxGrid::DisableDragColMove, 743  
wxGrid::DisableDragColSize, 743  
wxGrid::DisableDragGridSize, 743  
wxGrid::DisableDragRowSize, 743  
wxGrid::EnableCellEditControl, 743  
wxGrid::EnableDragColMove, 744  
wxGrid::EnableDragColSize, 744  
wxGrid::EnableDragGridSize, 744  
wxGrid::EnableDragRowSize, 744  
wxGrid::EnableEditing, 744  
wxGrid::EnableGridLines, 744  
wxGrid::EndBatch, 744  
wxGrid::Fit, 745  
wxGrid::ForceRefresh, 745  
wxGrid::GetBatchCount, 745  
wxGrid::GetCellAlignment, 745  
wxGrid::GetCellBackgroundColour, 745  
wxGrid::GetCellEditor, 745  
wxGrid::GetCellFont, 746  
wxGrid::GetCellRenderer, 746  
wxGrid::GetCellTextColour, 746  
wxGrid::GetCellValue, 746  
wxGrid::GetColAt, 746  
wxGrid::GetColGridLinePen, 751  
wxGrid::GetColLabelAlignment, 747  
wxGrid::GetColLabelSize, 747  
wxGrid::GetColLabelValue, 747  
wxGrid::GetColLeft, 747  
wxGrid::GetColMinimalAcceptableWidth, 747  
wxGrid::GetColMinimalWidth, 747  
wxGrid::GetColPos, 747  
wxGrid::GetColRight, 748  
wxGrid::GetColSize, 748  
wxGrid::GetDefaultCellAlignment, 748  
wxGrid::GetDefaultCellBackgroundColour, 748  
wxGrid::GetDefaultCellFont, 748  
wxGrid::GetDefaultCellTextColour, 748  
wxGrid::GetDefaultColLabelSize, 748  
wxGrid::GetDefaultColSize, 748  
wxGrid::GetDefaultEditor, 749  
wxGrid::GetDefaultEditorForCell, 749  
wxGrid::GetDefaultEditorForType, 749  
wxGrid::GetDefaultGridLinePen, 750  
wxGrid::GetDefaultRenderer, 749  
wxGrid::GetDefaultRendererForCell, 749  
wxGrid::GetDefaultRendererForType, 749  
wxGrid::GetDefaultRowLabelSize, 749  
wxGrid::GetDefaultRowSize, 749  
wxGrid::GetGridCursorCol, 749  
wxGrid::GetGridCursorRow, 750  
wxGrid::GetGridLineColour, 750  
wxGrid::GetLabelBackgroundColour, 751  
wxGrid::GetLabelFont, 751  
wxGrid::GetLabelTextColour, 751  
wxGrid::GetNumberCols, 751  
wxGrid::GetNumberRows, 751  
wxGrid::GetOrCreateCellAttr, 752  
wxGrid::GetRowGridLinePen, 750  
wxGrid::GetRowLabelAlignment, 752  
wxGrid::GetRowLabelSize, 752  
wxGrid::GetRowLabelValue, 752  
wxGrid::GetRowMinimalAcceptableHeight, 752  
wxGrid::GetRowMinimalHeight, 752  
wxGrid::GetRowSize, 752  
wxGrid::GetScrollLineX, 752  
wxGrid::GetScrollLineY, 753  
wxGrid::GetSelectedCells, 753  
wxGrid::GetSelectedCols, 753  
wxGrid::GetSelectedRows, 753  
wxGrid::GetSelectionBackground, 753  
wxGrid::GetSelectionBlockBottomRight, 754  
wxGrid::GetSelectionBlockTopLeft, 753  
wxGrid::GetSelectionForeground, 754  
wxGrid::GetSelectionMode, 753  
wxGrid::GetTable, 754  
wxGrid::GetViewWidth, 754  
wxGrid::GridLinesEnabled, 751  
wxGrid::HideCellEditControl, 754  
wxGrid::InitColWidths, 754  
wxGrid::InitRowHeights, 754  
wxGrid::InsertCols, 754  
wxGrid::InsertRows, 755  
wxGrid::IsCellEditControlEnabled, 755  
wxGrid::IsCurrentCellReadOnly, 755  
wxGrid::IsEditable, 755  
wxGrid::IsInSelection, 755  
wxGrid::IsReadOnly, 756  
wxGrid::IsSelection, 756  
wxGrid::IsVisible, 756  
wxGrid::MakeCellVisible, 756  
wxGrid::MoveCursorDown, 756

- 
- wxGrid::MoveCursorDownBlock, 757
  - wxGrid::MoveCursorLeft, 756
  - wxGrid::MoveCursorLeftBlock, 757
  - wxGrid::MoveCursorRight, 757
  - wxGrid::MoveCursorRightBlock, 757
  - wxGrid::MoveCursorUp, 757
  - wxGrid::MoveCursorUpBlock, 758
  - wxGrid::MovePageDown, 758
  - wxGrid::MovePageUp, 758
  - wxGrid::RegisterDataType, 758
  - wxGrid::SaveEditControlValue, 758
  - wxGrid::SelectAll, 759
  - wxGrid::SelectBlock, 759
  - wxGrid::SelectCol, 759
  - wxGrid::SelectionToDeviceRect, 759
  - wxGrid::SelectRow, 759
  - wxGrid::SetCellAlignment, 759
  - wxGrid::SetCellBackgroundColour, 760
  - wxGrid::SetCellEditor, 760
  - wxGrid::SetCellFont, 760
  - wxGrid::SetCellRenderer, 760
  - wxGrid::SetCellTextColour, 760
  - wxGrid::SetCellValue, 760
  - wxGrid::SetColAttr, 761
  - wxGrid::SetColFormatBool, 761
  - wxGrid::SetColFormatCustom, 761
  - wxGrid::SetColFormatFloat, 761
  - wxGrid::SetColFormatNumber, 761
  - wxGrid::SetColLabelAlignment, 761
  - wxGrid::SetColLabelSize, 762
  - wxGrid::SetColLabelValue, 762
  - wxGrid::SetColMinimalAcceptableWidth, 762
  - wxGrid::SetColMinimalWidth, 762
  - wxGrid::SetColPos, 762
  - wxGrid::SetColSize, 763
  - wxGrid::SetDefaultCellAlignment, 763
  - wxGrid::SetDefaultCellBackgroundColour, 763
  - wxGrid::SetDefaultCellFont, 763
  - wxGrid::SetDefaultCellTextColour, 763
  - wxGrid::SetDefaultColSize, 764
  - wxGrid::SetDefaultEditor, 763
  - wxGrid::SetDefaultRenderer, 764
  - wxGrid::SetDefaultRowSize, 764
  - wxGrid::SetGridCursor, 764
  - wxGrid::SetGridLineColour, 764
  - wxGrid::SetLabelBackgroundColour, 764
  - wxGrid::SetLabelFont, 764
  - wxGrid::SetLabelTextColour, 765
  - wxGrid::SetMargins, 765
  - wxGrid::SetOrCalcColumnSizes, 765
  - wxGrid::SetOrCalcRowSizes, 765
  - wxGrid::SetReadOnly, 765
  - wxGrid::SetRowAttr, 765
  - wxGrid::SetRowLabelAlignment, 765
  - wxGrid::SetRowLabelSize, 766
  - wxGrid::SetRowLabelValue, 766
  - wxGrid::SetRowMinimalAcceptableHeight, 766
  - wxGrid::SetRowMinimalHeight, 766
  - wxGrid::SetRowSize, 766
  - wxGrid::SetScrollLineX, 767
  - wxGrid::SetScrollLineY, 767
  - wxGrid::SetSelectionBackground, 767
  - wxGrid::SetSelectionForeground, 767
  - wxGrid::SetSelectionMode, 767
  - wxGrid::SetTable, 768
  - wxGrid::ShowCellEditControl, 768
  - wxGrid::wxGrid, 738
  - wxGrid::XToCol, 768
  - wxGrid::XToEdgeOfCol, 768
  - wxGrid::YToEdgeOfRow, 768
  - wxGrid::YToRow, 769
  - wxGridBagSizer, 772
  - wxGridBagSizer::Add, 772
  - wxGridBagSizer::CalcMin, 772
  - wxGridBagSizer::CheckForIntersection, 773
  - wxGridBagSizer::FindItem, 773
  - wxGridBagSizer::FindItemAtPoint, 773
  - wxGridBagSizer::FindItemAtPosition, 773
  - wxGridBagSizer::FindItemWithData, 773
  - wxGridBagSizer::GetCellSize, 773
  - wxGridBagSizer::GetEmptyCellSize, 774
  - wxGridBagSizer::GetItemPosition, 774
  - wxGridBagSizer::GetItemSpan, 774
  - wxGridBagSizer::RecalcSizes, 774
  - wxGridBagSizer::SetEmptyCellSize, 774
  - wxGridBagSizer::SetItemPosition, 774
  - wxGridBagSizer::SetItemSpan, 774
  - wxGridBagSizer::wxGridBagSizer, 772
  - wxGridCellAttr, 769
  - wxGridCellAttr::Clone, 769
  - wxGridCellAttr::DecRef, 769
  - wxGridCellAttr::GetAlignment, 771
  - wxGridCellAttr::GetBackgroundColour, 771
  - wxGridCellAttr::GetEditor, 771
  - wxGridCellAttr::GetFont, 771
  - wxGridCellAttr::GetRenderer, 771
  - wxGridCellAttr::GetTextColour, 771
  - wxGridCellAttr::HasAlignment, 771
  - wxGridCellAttr::HasBackgroundColour, 770
  - wxGridCellAttr::HasEditor, 771
  - wxGridCellAttr::HasFont, 771
  - wxGridCellAttr::HasRenderer, 771
  - wxGridCellAttr::HasTextColour, 770
  - wxGridCellAttr::IncRef, 769
  - wxGridCellAttr::IsReadOnly, 771
  - wxGridCellAttr::SetAlignment, 770
  - wxGridCellAttr::SetBackgroundColour, 770
  - wxGridCellAttr::SetDefAttr, 772
  - wxGridCellAttr::SetEditor, 770
  - wxGridCellAttr::SetFont, 770
  - wxGridCellAttr::SetReadOnly, 770
  - wxGridCellAttr::SetRenderer, 770
  - wxGridCellAttr::SetTextColour, 770
  - wxGridCellAttr::wxGridCellAttr, 769
  - wxGridCellBoolEditor, 775
  - wxGridCellBoolEditor::IsTrueValue, 775
  - wxGridCellBoolEditor::UseStringValue, 775
  - wxGridCellBoolEditor::wxGridCellBoolEditor, 775
  - wxGridCellBoolRenderer, 789
  - wxGridCellBoolRenderer::wxGridCellBoolRender  
er, 789
  - wxGridCellChoiceEditor, 776
  - wxGridCellChoiceEditor::SetParameters, 776
  - wxGridCellChoiceEditor::wxGridCellChoiceEditor,
-

- 776
- wxGridCellEditor, 777
- wxGridCellEditor::~wxGridCellEditor, 778
- wxGridCellEditor::BeginEdit, 777
- wxGridCellEditor::Clone, 778
- wxGridCellEditor::Create, 777
- wxGridCellEditor::Destroy, 778
- wxGridCellEditor::EndEdit, 777
- wxGridCellEditor::HandleReturn, 778
- wxGridCellEditor::IsCreated, 777
- wxGridCellEditor::PaintBackground, 777
- wxGridCellEditor::Reset, 778
- wxGridCellEditor::SetSize, 777
- wxGridCellEditor::Show, 777
- wxGridCellEditor::StartingClick, 778
- wxGridCellEditor::StartingKey, 778
- wxGridCellEditor::wxGridCellEditor, 777
- wxGridCellFloatEditor, 779
- wxGridCellFloatEditor::SetParameters, 779
- wxGridCellFloatEditor::wxGridCellFloatEditor, 779
- wxGridCellFloatRenderer, 790
- wxGridCellFloatRenderer::GetPrecision, 790
- wxGridCellFloatRenderer::GetWidth, 790
- wxGridCellFloatRenderer::SetParameters, 790
- wxGridCellFloatRenderer::SetPrecision, 790
- wxGridCellFloatRenderer::SetWidth, 791
- wxGridCellFloatRenderer::wxGridCellFloatRenderer, 790
- wxGridCellNumberEditor, 780
- wxGridCellNumberEditor::GetString, 780
- wxGridCellNumberEditor::HasRange, 780
- wxGridCellNumberEditor::SetParameters, 780
- wxGridCellNumberEditor::wxGridCellNumberEditor, 780
- wxGridCellNumberRenderer, 791
- wxGridCellNumberRenderer::wxGridCellNumberRenderer, 791
- wxGridCellRenderer::Clone, 792
- wxGridCellRenderer::Draw, 792
- wxGridCellRenderer::GetBestSize, 792
- wxGridCellStringRenderer, 792
- wxGridCellStringRenderer::wxGridCellStringRenderer, 792
- wxGridCellTextEditor, 780
- wxGridCellTextEditor::SetParameters, 781
- wxGridCellTextEditor::wxGridCellTextEditor, 780
- wxGridEditorCreatedEvent, 781
- wxGridEditorCreatedEvent::GetCol, 781
- wxGridEditorCreatedEvent::GetControl, 781
- wxGridEditorCreatedEvent::GetRow, 782
- wxGridEditorCreatedEvent::SetCol, 782
- wxGridEditorCreatedEvent::SetControl, 782
- wxGridEditorCreatedEvent::SetRow, 782
- wxGridEditorCreatedEvent::wxGridEditorCreatedEvent, 781
- wxGridEvent, 784
- wxGridEvent::AltDown, 785
- wxGridEvent::ControlDown, 785
- wxGridEvent::GetCol, 785
- wxGridEvent::GetPosition, 785
- wxGridEvent::GetRow, 785
- wxGridEvent::MetaDown, 785
- wxGridEvent::Selecting, 785
- wxGridEvent::ShiftDown, 785
- wxGridEvent::wxGridEvent, 784
- wxGridRangeSelectEvent, 786
- wxGridRangeSelectEvent::AltDown, 786
- wxGridRangeSelectEvent::ControlDown, 786
- wxGridRangeSelectEvent::GetBottomRightCoords, 786
- wxGridRangeSelectEvent::GetBottomRow, 787
- wxGridRangeSelectEvent::GetLeftCol, 787
- wxGridRangeSelectEvent::GetRightCol, 787
- wxGridRangeSelectEvent::GetTopLeftCoords, 787
- wxGridRangeSelectEvent::GetTopRow, 787
- wxGridRangeSelectEvent::MetaDown, 787
- wxGridRangeSelectEvent::Selecting, 787
- wxGridRangeSelectEvent::ShiftDown, 787
- wxGridRangeSelectEvent::wxGridRangeSelectEvent, 786
- wxGridSizeEvent, 788
- wxGridSizeEvent::AltDown, 788
- wxGridSizeEvent::ControlDown, 788
- wxGridSizeEvent::GetPosition, 789
- wxGridSizeEvent::GetRowOrCol, 789
- wxGridSizeEvent::MetaDown, 789
- wxGridSizeEvent::ShiftDown, 789
- wxGridSizeEvent::wxGridSizeEvent, 788
- wxGridSizer, 797
- wxGridSizer::GetCols, 797
- wxGridSizer::GetHGap, 797
- wxGridSizer::GetRows, 797
- wxGridSizer::GetVGap, 798
- wxGridSizer::SetCols, 798
- wxGridSizer::SetHGap, 798
- wxGridSizer::SetRows, 798
- wxGridSizer::SetVGap, 798
- wxGridSizer::wxGridSizer, 797
- wxGridTableBase, 793
- wxGridTableBase::~wxGridTableBase, 793
- wxGridTableBase::AppendCols, 795
- wxGridTableBase::AppendRows, 795
- wxGridTableBase::CanGetValueAs, 794
- wxGridTableBase::CanHaveAttributes, 796
- wxGridTableBase::CanSetValueAs, 794
- wxGridTableBase::Clear, 795
- wxGridTableBase::DeleteCols, 795
- wxGridTableBase::DeleteRows, 795
- wxGridTableBase::GetAttr, 796
- wxGridTableBase::GetAttrProvider, 796
- wxGridTableBase::GetColLabelValue, 795
- wxGridTableBase::GetNumberCols, 793
- wxGridTableBase::GetNumberRows, 793
- wxGridTableBase::GetRowLabelValue, 795
- wxGridTableBase::GetTypeName, 793
- wxGridTableBase::GetValue, 793
- wxGridTableBase::GetValueAsBool, 794
- wxGridTableBase::GetValueAsCustom, 794
- wxGridTableBase::GetValueAsDouble, 794
- wxGridTableBase::GetValueAsLong, 794
- wxGridTableBase::GetView, 795
- wxGridTableBase::InsertCols, 795

- wxGridTableBase::InsertRows, 795
- wxGridTableBase::IsEmptyCell, 793
- wxGridTableBase::SetAttr, 796
- wxGridTableBase::SetAttrProvider, 796
- wxGridTableBase::SetColAttr, 796
- wxGridTableBase::SetColLabelValue, 795
- wxGridTableBase::SetRowAttr, 796
- wxGridTableBase::SetRowLabelValue, 795
- wxGridTableBase::SetValue, 793
- wxGridTableBase::SetValueAsBool, 794
- wxGridTableBase::SetValueAsCustom, 794
- wxGridTableBase::SetValueAsDouble, 794
- wxGridTableBase::SetValueAsLong, 794
- wxGridTableBase::SetView, 794
- wxGridTableBase::UpdateAttrCols, 796
- wxGridTableBase::UpdateAttrRows, 796
- wxGridTableBase::wxGridTableBase, 793
- wxHandleFatalExceptions, 1911
- wxHashMap, 801
- wxHashMap::begin, 801
- wxHashMap::clear, 801
- wxHashMap::count, 801
- wxHashMap::empty, 801
- wxHashMap::end, 802
- wxHashMap::erase, 802
- wxHashMap::find, 802
- wxHashMap::insert, 802
- wxHashMap::operator[], 802
- wxHashMap::size, 802
- wxHashMap::wxHashMap, 801
- wxHashSet, 805
- wxHashSet::begin, 805
- wxHashSet::clear, 805
- wxHashSet::count, 805
- wxHashSet::empty, 806
- wxHashSet::end, 806
- wxHashSet::erase, 806
- wxHashSet::find, 806
- wxHashSet::insert, 806
- wxHashSet::size, 807
- wxHashSet::wxHashSet, 805
- wxHashTable, 808
- wxHashTable::~wxHashTable, 808
- wxHashTable::BeginFind, 808
- wxHashTable::Clear, 808
- wxHashTable::Delete, 808
- wxHashTable::DeleteContents, 808
- wxHashTable::Get, 808
- wxHashTable::GetCount, 809
- wxHashTable::MakeKey, 808
- wxHashTable::Next, 809
- wxHashTable::Put, 809
- wxHashTable::wxHashTable, 808
- wxHelpController, 811
- wxHelpController::~wxHelpController, 811
- wxHelpController::DisplayBlock, 811
- wxHelpController::DisplayContents, 812
- wxHelpController::DisplayContextPopup, 812
- wxHelpController::DisplaySection, 812
- wxHelpController::DisplayTextPopup, 812
- wxHelpController::GetFrameParameters, 813
- wxHelpController::GetParentWindow, 813
- wxHelpController::Initialize, 811
- wxHelpController::KeywordSearch, 813
- wxHelpController::LoadFile, 813
- wxHelpController::OnQuit, 814
- wxHelpController::Quit, 814
- wxHelpController::SetFrameParameters, 814
- wxHelpController::SetParentWindow, 814
- wxHelpController::SetViewer, 814
- wxHelpController::wxHelpController, 811
- wxHelpControllerHelpProvider, 815
- wxHelpControllerHelpProvider::GetHelpController, 815
- wxHelpControllerHelpProvider::SetHelpController, 815
- wxHelpControllerHelpProvider::wxHelpControllerHelpProvider, 815
- wxHelpEvent, 816
- wxHelpEvent::GetOrigin, 816
- wxHelpEvent::GetPosition, 817
- wxHelpEvent::SetOrigin, 817
- wxHelpEvent::SetPosition, 817
- wxHelpEvent::wxHelpEvent, 816
- wxHelpProvider::~wxHelpProvider, 818
- wxHelpProvider::AddHelp, 818
- wxHelpProvider::Get, 818
- wxHelpProvider::GetHelp, 818
- wxHelpProvider::RemoveHelp, 818
- wxHelpProvider::Set, 818
- wxHelpProvider::ShowHelp, 819
- wxHelpProvider::ShowHelpAtPoint, 819
- wxHF\_BOOKMARKS, 839
- wxHF\_CONTENTS, 839
- wxHF\_DEFAULT\_STYLE, 840
- wxHF\_DIALOG, 840
- wxHF\_EMBEDDED, 840
- wxHF\_FLAT\_TOOLBAR, 839
- wxHF\_FRAME, 840
- wxHF\_ICONS\_BOOK, 839
- wxHF\_ICONS\_BOOK\_CHAPTER, 839
- wxHF\_ICONS\_FOLDER, 839
- wxHF\_INDEX, 839
- wxHF\_MERGE\_BOOKS, 839
- wxHF\_MODAL, 840
- wxHF\_OPEN\_FILES, 839
- wxHF\_PRINT, 839
- wxHF\_SEARCH, 839
- wxHF\_TOOLBAR, 839
- wxHL\_ALIGN\_CENTRE, 891
- wxHL\_ALIGN\_LEFT, 891
- wxHL\_ALIGN\_RIGHT, 891
- wxHL\_CONTEXTMENU, 891
- wxHL\_DEFAULT\_STYLE, 891
- wxHLB\_DEFAULT\_STYLE, 856
- wxHLB\_MULTIPLE, 856
- wxHSCROLL, 1052, 1634, 1797
- wxHTML quick start, 2179
- wxHtmlCell, 820
- wxHtmlCell::AdjustPagebreak, 820
- wxHtmlCell::Draw, 820
- wxHtmlCell::DrawInvisible, 821
- wxHtmlCell::Find, 821
- wxHtmlCell::GetDescent, 821



---

wxHtmlCell::GetFirstChild, 822  
wxHtmlCell::GetHeight, 822  
wxHtmlCell::GetId, 822  
wxHtmlCell::GetLink, 822  
wxHtmlCell::GetMouseCursor, 822  
wxHtmlCell::GetNext, 823  
wxHtmlCell::GetParent, 823  
wxHtmlCell::GetPosX, 823  
wxHtmlCell::GetPosY, 823  
wxHtmlCell::GetWidth, 823  
wxHtmlCell::Layout, 823  
wxHtmlCell::ProcessMouseClicked, 824  
wxHtmlCell::SetId, 824  
wxHtmlCell::SetLink, 824  
wxHtmlCell::SetNext, 825  
wxHtmlCell::SetParent, 825  
wxHtmlCell::SetPos, 825  
wxHtmlCell::wxHtmlCell, 820  
wxHtmlCellEvent, 882  
wxHtmlCellEvent::GetCell, 882  
wxHtmlCellEvent::GetLinkClicked, 883  
wxHtmlCellEvent::GetPoint, 883  
wxHtmlCellEvent::SetLinkClicked, 883  
wxHtmlCellEvent::wxHtmlCellEvent, 882  
wxHtmlColourCell, 825  
wxHtmlColourCell::wxHtmlColourCell, 825  
wxHtmlContainerCell, 826  
wxHtmlContainerCell::GetAlignHor, 826  
wxHtmlContainerCell::GetAlignVer, 826  
wxHtmlContainerCell::GetBackgroundColour, 826  
wxHtmlContainerCell::GetIndent, 827  
wxHtmlContainerCell::GetIndentUnits, 827  
wxHtmlContainerCell::InsertCell, 827  
wxHtmlContainerCell::SetAlign, 827  
wxHtmlContainerCell::SetAlignHor, 827  
wxHtmlContainerCell::SetAlignVer, 828  
wxHtmlContainerCell::SetBackgroundColour, 828  
wxHtmlContainerCell::SetBorder, 828  
wxHtmlContainerCell::SetIndent, 829  
wxHtmlContainerCell::SetMinHeight, 830  
wxHtmlContainerCell::SetWidthFloat, 830  
wxHtmlContainerCell::wxHtmlContainerCell, 826  
wxHtmlDCRenderer, 832  
wxHtmlDCRenderer::GetTotalHeight, 833  
wxHtmlDCRenderer::Render, 833  
wxHtmlDCRenderer::SetDC, 832  
wxHtmlDCRenderer::SetFont, 832  
wxHtmlDCRenderer::SetHtmlText, 832  
wxHtmlDCRenderer::SetSize, 832  
wxHtmlDCRenderer::wxHtmlDCRenderer, 832  
wxHtmlEasyPrinting, 834  
wxHtmlEasyPrinting::GetPageSetupData, 834  
wxHtmlEasyPrinting::GetParentWindow, 834  
wxHtmlEasyPrinting::GetPrintData, 834  
wxHtmlEasyPrinting::PageSetup, 836  
wxHtmlEasyPrinting::PreviewFile, 835  
wxHtmlEasyPrinting::PreviewText, 835  
wxHtmlEasyPrinting::PrintFile, 835  
wxHtmlEasyPrinting::PrintText, 835  
wxHtmlEasyPrinting::SetFont, 836  
wxHtmlEasyPrinting::SetFooter, 836  
wxHtmlEasyPrinting::SetHeader, 836  
wxHtmlEasyPrinting::SetParentWindow, 837  
wxHtmlEasyPrinting::wxHtmlEasyPrinting, 834  
wxHtmlFilter, 837  
wxHtmlFilter::CanRead, 837  
wxHtmlFilter::ReadFile, 838  
wxHtmlFilter::wxHtmlFilter, 837  
wxHtmlHelpController, 839  
wxHtmlHelpController::AddBook, 840  
wxHtmlHelpController::CreateHelpDialog, 841  
wxHtmlHelpController::CreateHelpFrame, 841  
wxHtmlHelpController::Display, 841  
wxHtmlHelpController::DisplayContents, 842  
wxHtmlHelpController::DisplayIndex, 842  
wxHtmlHelpController::KeywordSearch, 842  
wxHtmlHelpController::ReadCustomization, 842  
wxHtmlHelpController::SetTempDir, 842  
wxHtmlHelpController::SetTitleFormat, 842  
wxHtmlHelpController::UseConfig, 843  
wxHtmlHelpController::WriteCustomization, 843  
wxHtmlHelpController::wxHtmlHelpController, 839  
wxHtmlHelpData, 843  
wxHtmlHelpData::AddBook, 843  
wxHtmlHelpData::FindPageById, 843  
wxHtmlHelpData::FindPageByName, 844  
wxHtmlHelpData::GetBookRecArray, 844  
wxHtmlHelpData::GetContentsArray, 844  
wxHtmlHelpData::GetIndexArray, 844  
wxHtmlHelpData::SetTempDir, 844  
wxHtmlHelpData::wxHtmlHelpData, 843  
wxHtmlHelpDialog, 845  
wxHtmlHelpDialog::AddToolBarButtons, 845  
wxHtmlHelpDialog::Create, 845  
wxHtmlHelpDialog::GetController, 845  
wxHtmlHelpDialog::ReadCustomization, 845  
wxHtmlHelpDialog::SetController, 845  
wxHtmlHelpDialog::SetTitleFormat, 845  
wxHtmlHelpDialog::WriteCustomization, 846  
wxHtmlHelpDialog::wxHtmlHelpDialog, 845  
wxHtmlHelpFrame, 846  
wxHtmlHelpFrame::AddToolBarButtons, 846  
wxHtmlHelpFrame::Create, 846  
wxHtmlHelpFrame::GetController, 847  
wxHtmlHelpFrame::ReadCustomization, 847  
wxHtmlHelpFrame::SetController, 847  
wxHtmlHelpFrame::SetTitleFormat, 847  
wxHtmlHelpFrame::WriteCustomization, 847  
wxHtmlHelpFrame::wxHtmlHelpFrame, 846  
wxHtmlHelpWindow, 848  
wxHtmlHelpWindow::AddToolBarButtons, 850  
wxHtmlHelpWindow::Create, 848  
wxHtmlHelpWindow::CreateContents, 848  
wxHtmlHelpWindow::CreateIndex, 849  
wxHtmlHelpWindow::CreateSearch, 849  
wxHtmlHelpWindow::Display, 849  
wxHtmlHelpWindow::DisplayContents, 849  
wxHtmlHelpWindow::DisplayIndex, 849  
wxHtmlHelpWindow::GetData, 849  
wxHtmlHelpWindow::KeywordSearch, 850  
wxHtmlHelpWindow::ReadCustomization, 850  
wxHtmlHelpWindow::RefreshLists, 850  
wxHtmlHelpWindow::SetTitleFormat, 850

---

wxHtmlHelpWindow::UseConfig, 850  
 wxHtmlHelpWindow::WriteCustomization, 850  
 wxHtmlHelpWindow::wxHtmlHelpWindow, 848  
 wxHtmlLinkEvent::GetLinkInfo, 882  
 wxHtmlLinkEvent::wxHtmlLinkEvent, 881  
 wxHtmlLinkInfo, 852  
 wxHtmlLinkInfo::GetEvent, 852  
 wxHtmlLinkInfo::GetHref, 852  
 wxHtmlLinkInfo::GetHtmlCell, 852  
 wxHtmlLinkInfo::GetTarget, 852  
 wxHtmlLinkInfo::wxHtmlLinkInfo, 852  
 wxHtmlListBox, 853, 857  
 wxHtmlListBox::~wxHtmlListBox, 853  
 wxHtmlListBox::Create, 854  
 wxHtmlListBox::GetFileSystem, 854  
 wxHtmlListBox::GetSelectedTextBgColour, 854  
 wxHtmlListBox::GetSelectedTextColour, 854  
 wxHtmlListBox::OnGetItem, 855  
 wxHtmlListBox::OnGetItemMarkup, 855  
 wxHtmlListBox::OnLinkClicked, 855  
 wxHtmlListBox::wxHtmlListBox, 853  
 wxHtmlModalHelp, 851  
 wxHtmlModalHelp::wxHtmlModalHelp, 851  
 wxHtmlParser, 859  
 wxHtmlParser::AddTag, 859  
 wxHtmlParser::AddTagHandler, 859  
 wxHtmlParser::AddText, 860  
 wxHtmlParser::DoneParser, 860  
 wxHtmlParser::DoParsing, 860  
 wxHtmlParser::GetFS, 860  
 wxHtmlParser::GetProduct, 860  
 wxHtmlParser::GetSource, 860  
 wxHtmlParser::InitParser, 861  
 wxHtmlParser::OpenURL, 861  
 wxHtmlParser::Parse, 861  
 wxHtmlParser::PopTagHandler, 863  
 wxHtmlParser::PushTagHandler, 862  
 wxHtmlParser::SetFS, 863  
 wxHtmlParser::StopParsing, 863  
 wxHtmlParser::wxHtmlParser, 859  
 wxHtmlPrintout, 863  
 wxHtmlPrintout::AddFilter, 863  
 wxHtmlPrintout::SetFont, 864  
 wxHtmlPrintout::SetFooter, 864  
 wxHtmlPrintout::SetHeader, 864  
 wxHtmlPrintout::SetHtmlFile, 865  
 wxHtmlPrintout::SetHtmlText, 865  
 wxHtmlPrintout::SetMargins, 865  
 wxHtmlPrintout::wxHtmlPrintout, 863  
 wxHtmlTag, 866  
 wxHtmlTag::GetAllParams, 866  
 wxHtmlTag::GetBeginPos, 866  
 wxHtmlTag::GetEndPos1, 866  
 wxHtmlTag::GetEndPos2, 866  
 wxHtmlTag::GetName, 867  
 wxHtmlTag::GetParam, 867  
 wxHtmlTag::GetParamAsColour, 867  
 wxHtmlTag::GetParamAsInt, 868  
 wxHtmlTag::HasEnding, 868  
 wxHtmlTag::HasParam, 868  
 wxHtmlTag::ScanParam, 868  
 wxHtmlTag::wxHtmlTag, 866  
 wxHtmlTagHandler, 869  
 wxHtmlTagHandler::GetSupportedTags, 869  
 wxHtmlTagHandler::HandleTag, 869  
 wxHtmlTagHandler::m\_Parser, 869  
 wxHtmlTagHandler::ParseInner, 870  
 wxHtmlTagHandler::SetParser, 870  
 wxHtmlTagHandler::wxHtmlTagHandler, 869  
 wxHtmlTagsModule::FillHandlersTable, 871  
 wxHtmlWidgetCell, 871  
 wxHtmlWidgetCell::wxHtmlWidgetCell, 871  
 wxHtmlWindow, 873  
 wxHtmlWindow::AddFilter, 873  
 wxHtmlWindow::AppendToPage, 873  
 wxHtmlWindow::GetInternalRepresentation, 873  
 wxHtmlWindow::GetOpenedAnchor, 874  
 wxHtmlWindow::GetOpenedPage, 874  
 wxHtmlWindow::GetOpenedPageTitle, 874  
 wxHtmlWindow::GetRelatedFrame, 874  
 wxHtmlWindow::HistoryBack, 874  
 wxHtmlWindow::HistoryCanBack, 874  
 wxHtmlWindow::HistoryCanForward, 874  
 wxHtmlWindow::HistoryClear, 874  
 wxHtmlWindow::HistoryForward, 875  
 wxHtmlWindow::LoadFile, 875  
 wxHtmlWindow::LoadPage, 875  
 wxHtmlWindow::OnCellClicked, 875  
 wxHtmlWindow::OnCellMouseHover, 876  
 wxHtmlWindow::OnLinkClicked, 876  
 wxHtmlWindow::OnOpeningURL, 877  
 wxHtmlWindow::OnSetTitle, 877  
 wxHtmlWindow::ReadCustomization, 877  
 wxHtmlWindow::SelectAll, 878  
 wxHtmlWindow::SelectionToText, 878  
 wxHtmlWindow::SelectLine, 878  
 wxHtmlWindow::SelectWord, 878  
 wxHtmlWindow::SetBorders, 879  
 wxHtmlWindow::SetFont, 879  
 wxHtmlWindow::SetPage, 880  
 wxHtmlWindow::SetRelatedFrame, 880  
 wxHtmlWindow::SetRelatedStatusBar, 880  
 wxHtmlWindow::ToText, 881  
 wxHtmlWindow::WriteCustomization, 881  
 wxHtmlWindow::wxHtmlWindow, 873  
 wxHtmlWinParser, 883  
 wxHtmlWinParser::AddModule, 884  
 wxHtmlWinParser::CloseContainer, 884  
 wxHtmlWinParser::CreateCurrentFont, 884  
 wxHtmlWinParser::GetActualColor, 884  
 wxHtmlWinParser::GetAlign, 884  
 wxHtmlWinParser::GetCharHeight, 884  
 wxHtmlWinParser::GetCharWidth, 884  
 wxHtmlWinParser::GetContainer, 885  
 wxHtmlWinParser::GetDC, 885  
 wxHtmlWinParser::GetEncodingConverter, 885  
 wxHtmlWinParser::GetFontBold, 885  
 wxHtmlWinParser::GetFontFace, 885  
 wxHtmlWinParser::GetFontFixed, 885  
 wxHtmlWinParser::GetFontItalic, 885  
 wxHtmlWinParser::GetFontSize, 886  
 wxHtmlWinParser::GetFontUnderlined, 886  
 wxHtmlWinParser::GetInputEncoding, 886  
 wxHtmlWinParser::GetLink, 886

---

wxHtmlWinParser::GetLinkColor, 886  
wxHtmlWinParser::GetOutputEncoding, 886  
wxHtmlWinParser::GetWindow, 886  
wxHtmlWinParser::OpenContainer, 886  
wxHtmlWinParser::SetActualColor, 887  
wxHtmlWinParser::SetAlign, 887  
wxHtmlWinParser::SetContainer, 887  
wxHtmlWinParser::SetDC, 887  
wxHtmlWinParser::SetFontBold, 887  
wxHtmlWinParser::SetFontFace, 887  
wxHtmlWinParser::SetFontFixed, 887  
wxHtmlWinParser::SetFontItalic, 888  
wxHtmlWinParser::SetFonts, 888  
wxHtmlWinParser::SetFontSize, 888  
wxHtmlWinParser::SetFontUnderlined, 888  
wxHtmlWinParser::SetInputEncoding, 888  
wxHtmlWinParser::SetLink, 888  
wxHtmlWinParser::SetLinkColor, 888  
wxHtmlWinParser::wxHtmlWinParser, 883  
wxHtmlWinTagHandler::m\_WParser, 889  
wxHTTP::GetHeader, 890  
wxHTTP::GetInputStream, 889  
wxHTTP::GetResponse, 889  
wxHTTP::SetHeader, 890  
wxHW\_NO\_SELECTION, 872  
wxHW\_SCROLLBAR\_AUTO, 872  
wxHW\_SCROLLBAR\_NEVER, 872  
wxHyperLink, 891  
wxHyperlinkCtrl::Create, 891  
wxHyperlinkCtrl::GetHoverColour, 892  
wxHyperlinkCtrl::GetNormalColour, 893  
wxHyperlinkCtrl::GetURL, 893  
wxHyperlinkCtrl::GetVisited, 893  
wxHyperlinkCtrl::GetVisitedColour, 893  
wxHyperlinkCtrl::SetHoverColour, 892  
wxHyperlinkCtrl::SetNormalColour, 893  
wxHyperlinkCtrl::SetURL, 894  
wxHyperlinkCtrl::SetVisited, 893  
wxHyperlinkCtrl::SetVisitedColour, 893  
wxHyperlinkCtrl::wxHyperLink, 891  
wxHyperlinkEvent, 881, 894  
wxHyperlinkEvent::GetURL, 894  
wxHyperlinkEvent::SetURL, 894  
wxHyperlinkEvent::wxHyperlinkEvent, 894  
wxICOHandler, 907  
wxIcon, 895, 896  
wxICON, 1946  
wxIcon::~~wxIcon, 898  
wxIcon::CopyFromBitmap, 898  
wxIcon::GetDepth, 899  
wxIcon::GetHeight, 899  
wxIcon::GetWidth, 899  
wxIcon::IsOk, 900  
wxIcon::LoadFile, 899  
wxIcon::operator =, 901  
wxIcon::SetDepth, 900  
wxIcon::SetHeight, 900  
wxIcon::SetWidth, 900  
wxIcon::wxIcon, 895  
wxIconBundle, 901  
wxIconBundle::~~wxIconBundle, 901  
wxIconBundle::AddIcon, 901  
wxIconBundle::GetIcon, 902  
wxIconBundle::operator =, 902  
wxIconBundle::wxIconBundle, 901  
wxICONIZE, 682, 1047, 1052, 1113  
wxIconizeEvent, 903  
wxIconizeEvent::Iconized, 903  
wxIconizeEvent::wxIconizeEvent, 903  
wxIconLocation::IsOk, 903  
wxID, 2082  
wxIdleEvent, 904  
wxIdleEvent::CanSend, 904  
wxIdleEvent::GetMode, 905  
wxIdleEvent::MoreRequested, 905  
wxIdleEvent::RequestMore, 905  
wxIdleEvent::SetMode, 905  
wxIdleEvent::wxIdleEvent, 904  
wxIFFHandler, 907  
wxImage, 907, 908  
wxImage::~~wxImage, 910  
wxImage::AddHandler, 910  
wxImage::Blur, 910  
wxImage::BlurHorizontal, 911  
wxImage::BlurVertical, 911  
wxImage::CleanUpHandlers, 911  
wxImage::ComputeHistogram, 911  
wxImage::ConvertAlphaToMask, 912  
wxImage::ConvertToBitmap, 912  
wxImage::ConvertToGreyscale, 912  
wxImage::ConvertToMono, 912  
wxImage::Copy, 912  
wxImage::Create, 912  
wxImage::Destroy, 913  
wxImage::FindFirstUnusedColour, 913  
wxImage::FindHandler, 914  
wxImage::GetAlpha, 915  
wxImage::GetBlue, 915  
wxImage::GetData, 915  
wxImage::GetGreen, 915  
wxImage::GetHandlers, 916  
wxImage::GetHeight, 917  
wxImage::GetImageCount, 915  
wxImage::GetImageExtWildcard, 914  
wxImage::GetMaskBlue, 917  
wxImage::GetMaskGreen, 917  
wxImage::GetMaskRed, 917  
wxImage::GetOption, 919  
wxImage::GetOptionInt, 919  
wxImage::GetOrFindMaskColour, 917  
wxImage::GetPalette, 917  
wxImage::GetRed, 917  
wxImage::GetSubImage, 918  
wxImage::GetWidth, 918  
wxImage::HasAlpha, 918  
wxImage::HasMask, 918  
wxImage::HasOption, 919  
wxImage::HSVtoRGB, 918  
wxImage::InitAlpha, 919  
wxImage::InitStandardHandlers, 920  
wxImage::InsertHandler, 920  
wxImage::IsOk, 922  
wxImage::IsTransparent, 920  
wxImage::LoadFile, 920

---

- wxImage::Mirror, 923
- wxImage::operator =, 929
- wxImage::RemoveHandler, 923
- wxImage::Replace, 923
- wxImage::Rescale, 923
- wxImage::Resize, 923
- wxImage::RGBtoHSV, 922
- wxImage::Rotate, 924
- wxImage::Rotate90, 924
- wxImage::RotateHue, 924
- wxImage::SaveFile, 924
- wxImage::Scale, 926
- wxImage::SetAlpha, 927
- wxImage::SetData, 927
- wxImage::SetMask, 928
- wxImage::SetMaskColour, 928
- wxImage::SetMaskFromImage, 928
- wxImage::SetOption, 929
- wxImage::SetPalette, 929
- wxImage::SetRGB, 929
- wxImage::Size, 927
- wxImage::wxImage, 907
- wxIMAGE\_LIST\_NORMAL, 987
- wxIMAGE\_LIST\_SMALL, 987
- wxIMAGE\_LIST\_STATE, 987
- wxImageHandler, 930
- wxImageHandler::~~wxImageHandler, 930
- wxImageHandler::GetExtension, 931
- wxImageHandler::GetImageCount, 931
- wxImageHandler::GetMimeType, 931
- wxImageHandler::GetName, 930
- wxImageHandler::GetType, 931
- wxImageHandler::LoadFile, 931
- wxImageHandler::SaveFile, 932
- wxImageHandler::SetExtension, 933
- wxImageHandler::SetMimeType, 933
- wxImageHandler::SetName, 932
- wxImageHandler::SetType, 933
- wxImageHandler::wxImageHandler, 930
- wxImageList, 934
- wxImageList::Add, 934
- wxImageList::Create, 935
- wxImageList::Draw, 935
- wxImageList::GetBitmap, 936
- wxImageList::GetIcon, 936
- wxImageList::GetImageCount, 936
- wxImageList::GetSize, 937
- wxImageList::Remove, 937
- wxImageList::RemoveAll, 937
- wxImageList::Replace, 937
- wxImageList::wxImageList, 934
- wxIndividualLayoutConstraint, 939
- wxIndividualLayoutConstraint::Above, 939
- wxIndividualLayoutConstraint::Absolute, 939
- wxIndividualLayoutConstraint::AsIs, 939
- wxIndividualLayoutConstraint::Below, 940
- wxIndividualLayoutConstraint::LeftOf, 940
- wxIndividualLayoutConstraint::PercentOf, 940
- wxIndividualLayoutConstraint::RightOf, 940
- wxIndividualLayoutConstraint::SameAs, 940
- wxIndividualLayoutConstraint::Set, 940
- wxIndividualLayoutConstraint::Unconstrained, 940
- wxIndividualLayoutConstraint::wxIndividualLayoutConstraint, 939
- wxInitAllImageHandlers, 1911
- wxInitDialogEvent, 941
- wxInitDialogEvent::wxInitDialogEvent, 941
- wxInitialize, 1911
- wxInputStream, 942
- wxInputStream::~~wxInputStream, 942
- wxInputStream::CanRead, 942
- wxInputStream::Eof, 942
- wxInputStream::GetC, 942
- wxInputStream::LastRead, 942
- wxInputStream::Peek, 943
- wxInputStream::Read, 943
- wxInputStream::Seekl, 943
- wxInputStream::Telll, 944
- wxInputStream::Ungetch, 944
- wxInputStream::wxInputStream, 942
- wxINT16\_SWAP\_ALWAYS, 1963
- wxINT16\_SWAP\_ON\_BE, 1964
- wxINT16\_SWAP\_ON\_LE, 1964
- wxINT32\_SWAP\_ALWAYS, 1963
- wxINT32\_SWAP\_ON\_BE, 1964
- wxINT32\_SWAP\_ON\_LE, 1964
- wxINTXX\_SWAP\_ALWAYS, 1963
- wxINTXX\_SWAP\_ON\_BE, 1963
- wxINTXX\_SWAP\_ON\_LE, 1964
- wxIPAddress::AnyAddress, 945
- wxIPAddress::Hostname, 944
- wxIPAddress::IPAddress, 945
- wxIPAddress::IsLocalHost, 946
- wxIPAddress::LocalHost, 946
- wxIPAddress::Service, 945
- wxIPCFormat, 277, 479, 1611
- wxIPv4address::AnyAddress, 947
- wxIPv4address::Hostname, 946
- wxIPv4address::IPAddress, 947
- wxIPv4address::LocalHost, 947
- wxIPv4address::Service, 947
- wxIsAbsolutePath, 1922
- wxIsBusy, 1942
- wxIsClipboardFormatAvailable, 1950
- wxIsDebuggerRunning, 1983
- wxIsEmpty, 1930
- wxIsMainThread, 1918
- wxIsNaN, 1944
- wxIsPlatform64Bit, 1928
- wxIsPlatformLittleEndian, 1928
- wxIsWild, 1923
- wxJoystick, 948
- wxJoystick::~~wxJoystick, 948
- wxJoystick::GetButtonState, 948
- wxJoystick::GetManufacturerId, 948
- wxJoystick::GetMovementThreshold, 948
- wxJoystick::GetNumberAxes, 949
- wxJoystick::GetNumberButtons, 949
- wxJoystick::GetNumberJoysticks, 949
- wxJoystick::GetPollingMax, 949
- wxJoystick::GetPollingMin, 949
- wxJoystick::GetPosition, 949
- wxJoystick::GetPOVCTSPosition, 950

---

wxJoystick::GetPOVPosition, 950  
wxJoystick::GetProductId, 949  
wxJoystick::GetProductName, 949  
wxJoystick::GetRudderMax, 950  
wxJoystick::GetRudderMin, 950  
wxJoystick::GetRudderPosition, 950  
wxJoystick::GetUMax, 950  
wxJoystick::GetUMin, 950  
wxJoystick::GetUPosition, 950  
wxJoystick::GetVMax, 951  
wxJoystick::GetVMin, 951  
wxJoystick::GetVPosition, 951  
wxJoystick::GetXMax, 951  
wxJoystick::GetXMin, 951  
wxJoystick::GetYMax, 951  
wxJoystick::GetYMin, 951  
wxJoystick::GetZMax, 951  
wxJoystick::GetZMin, 951  
wxJoystick::GetZPosition, 952  
wxJoystick::HasPOV, 952  
wxJoystick::HasPOV4Dir, 952  
wxJoystick::HasPOVCTS, 952  
wxJoystick::HasRudder, 952  
wxJoystick::HasU, 952  
wxJoystick::HasV, 952  
wxJoystick::HasZ, 952  
wxJoystick::IsOk, 953  
wxJoystick::ReleaseCapture, 953  
wxJoystick::SetCapture, 953  
wxJoystick::SetMovementThreshold, 953  
wxJoystick::wxJoystick, 948  
wxJoystickEvent, 954  
wxJoystickEvent::ButtonDown, 954  
wxJoystickEvent::ButtonIsDown, 955  
wxJoystickEvent::ButtonUp, 955  
wxJoystickEvent::GetButtonChange, 955  
wxJoystickEvent::GetButtonState, 955  
wxJoystickEvent::GetJoystick, 955  
wxJoystickEvent::GetPosition, 955  
wxJoystickEvent::GetZPosition, 956  
wxJoystickEvent::IsButton, 956  
wxJoystickEvent::IsMove, 956  
wxJoystickEvent::IsZMove, 956  
wxJoystickEvent::wxJoystickEvent, 954  
wxJPEGHandler, 906  
wxKeyEvent, 958  
wxKeyEvent::AltDown, 959  
wxKeyEvent::CmdDown, 959  
wxKeyEvent::ControlDown, 959  
wxKeyEvent::GetKeyCode, 959  
wxKeyEvent::GetModifiers, 959  
wxKeyEvent::GetPosition, 960  
wxKeyEvent::GetRawKeyCode, 960  
wxKeyEvent::GetRawKeyFlags, 960  
wxKeyEvent::GetUnicodeKey, 960  
wxKeyEvent::GetX, 960  
wxKeyEvent::GetY, 961  
wxKeyEvent::HasModifiers, 961  
wxKeyEvent::m\_altDown, 957  
wxKeyEvent::m\_controlDown, 957  
wxKeyEvent::m\_keyCode, 958  
wxKeyEvent::m\_metaDown, 958  
wxKeyEvent::m\_shiftDown, 958  
wxKeyEvent::m\_x, 958  
wxKeyEvent::m\_y, 958  
wxKeyEvent::MetaDown, 961  
wxKeyEvent::ShiftDown, 961  
wxKeyEvent::wxKeyEvent, 958  
wxKill, 1915  
wxLaunchDefaultBrowser, 1959  
wxLayoutAlgorithm, 963  
wxLayoutAlgorithm::~wxLayoutAlgorithm, 963  
wxLayoutAlgorithm::LayoutFrame, 963  
wxLayoutAlgorithm::LayoutMDIFrame, 964  
wxLayoutAlgorithm::LayoutWindow, 964  
wxLayoutAlgorithm::wxLayoutAlgorithm, 963  
wxLayoutConstraints, 965  
wxLayoutConstraints::bottom, 965  
wxLayoutConstraints::centreX, 965  
wxLayoutConstraints::centreY, 965  
wxLayoutConstraints::height, 965  
wxLayoutConstraints::left, 966  
wxLayoutConstraints::right, 966  
wxLayoutConstraints::top, 966  
wxLayoutConstraints::width, 966  
wxLayoutConstraints::wxLayoutConstraints, 965  
wxLB\_ALWAYS\_SB, 975  
wxLB\_BOTTOM, 974  
wxLB\_DEFAULT, 974  
wxLB\_EXTENDED, 975  
wxLB\_HSCROLL, 975  
wxLB\_LEFT, 974  
wxLB\_MULTIPLE, 975  
wxLB\_NEEDED\_SB, 975  
wxLB\_RIGHT, 974  
wxLB\_SINGLE, 975  
wxLB\_SORT, 975  
wxLB\_TOP, 974  
wxLC\_ALIGN\_LEFT, 981  
wxLC\_ALIGN\_TOP, 981  
wxLC\_AUTOARRANGE, 981  
wxLC\_EDIT\_LABELS, 981  
wxLC\_HRULES, 981  
wxLC\_ICON, 981  
wxLC\_LIST, 981  
wxLC\_NO\_HEADER, 981  
wxLC\_REPORT, 981  
wxLC\_SINGLE\_SEL, 981  
wxLC\_SMALL\_ICON, 981  
wxLC\_SORT\_ASCENDING, 981  
wxLC\_SORT\_DESCENDING, 981  
wxLC\_VIRTUAL, 981  
wxLC\_VRULES, 981  
wxLEAVE\_CRIT\_SECT, 1918  
wxLEFT, 1447  
wxLI\_HORIZONTAL, 1533  
wxLI\_VERTICAL, 1533  
wxList<T>, 968  
wxList<T>::~wxList<T>, 968  
wxList<T>::Append, 968  
wxList<T>::assign, 971  
wxList<T>::back, 971  
wxList<T>::begin, 971  
wxList<T>::clear, 971

wxList<T>::Clear, 968  
wxList<T>::DeleteContents, 968  
wxList<T>::DeleteNode, 968  
wxList<T>::DeleteObject, 968  
wxList<T>::empty, 971  
wxList<T>::end, 971  
wxList<T>::erase, 971  
wxList<T>::Erase, 969  
wxList<T>::Find, 969  
wxList<T>::front, 972  
wxList<T>::GetCount, 969  
wxList<T>::GetFirst, 969  
wxList<T>::GetLast, 969  
wxList<T>::IndexOf, 969  
wxList<T>::insert, 972  
wxList<T>::Insert, 969  
wxList<T>::IsEmpty, 970  
wxList<T>::Item, 970  
wxList<T>::max\_size, 972  
wxList<T>::Member, 970  
wxList<T>::Nth, 970  
wxList<T>::Number, 970  
wxList<T>::pop\_back, 972  
wxList<T>::pop\_front, 972  
wxList<T>::push\_back, 972  
wxList<T>::push\_front, 972  
wxList<T>::rbegin, 973  
wxList<T>::remove, 973  
wxList<T>::rend, 973  
wxList<T>::resize, 973  
wxList<T>::reverse, 973  
wxList<T>::size, 973  
wxList<T>::Sort, 970  
wxList<T>::splice, 973  
wxList<T>::wxList<T>, 968  
wxListBox, 976  
wxListBox::~~wxListBox, 977  
wxListBox::Create, 977  
wxListBox::Deselect, 977  
wxListBox::GetSelections, 977  
wxListBox::HitTest, 978  
wxListBox::InsertItems, 978  
wxListBox::IsSelected, 979  
wxListBox::Set, 979  
wxListBox::SetFirstItem, 980  
wxListBox::wxListBox, 975  
wxListCtrl, 982, 983  
wxListCtrl::~~wxListCtrl, 983  
wxListCtrl::Arrange, 983  
wxListCtrl::AssignImageList, 984  
wxListCtrl::ClearAll, 984  
wxListCtrl::Create, 984  
wxListCtrl::DeleteAllItems, 984  
wxListCtrl::DeleteColumn, 984  
wxListCtrl::DeleteItem, 985  
wxListCtrl::EditLabel, 985  
wxListCtrl::EnsureVisible, 985  
wxListCtrl::FindItem, 985  
wxListCtrl::GetColumn, 986  
wxListCtrl::GetColumnCount, 986  
wxListCtrl::GetColumnWidth, 986  
wxListCtrl::GetCountPerPage, 986  
wxListCtrl::GetEditControl, 986  
wxListCtrl::GetImageList, 986  
wxListCtrl::GetItem, 987  
wxListCtrl::GetItemBackgroundColour, 987  
wxListCtrl::GetItemCount, 987  
wxListCtrl::GetItemData, 987  
wxListCtrl::GetItemFont, 987  
wxListCtrl::GetItemPosition, 987  
wxListCtrl::GetItemRect, 988  
wxListCtrl::GetItemSpacing, 988  
wxListCtrl::GetItemState, 988  
wxListCtrl::GetItemText, 989  
wxListCtrl::GetItemTextColour, 989  
wxListCtrl::GetNextItem, 989  
wxListCtrl::GetSelectedItemCount, 990  
wxListCtrl::GetSubItemRect, 988  
wxListCtrl::GetTextColour, 990  
wxListCtrl::GetTopItem, 990  
wxListCtrl::GetViewRect, 990  
wxListCtrl::HitTest, 990  
wxListCtrl::InsertColumn, 991  
wxListCtrl::InsertItem, 992  
wxListCtrl::OnGetItemAttr, 993  
wxListCtrl::OnGetItemColumnImage, 993  
wxListCtrl::OnGetItemImage, 993  
wxListCtrl::OnGetItemText, 994  
wxListCtrl::RefreshItem, 994  
wxListCtrl::RefreshItems, 994  
wxListCtrl::ScrollList, 994  
wxListCtrl::SetBackgroundColour, 995  
wxListCtrl::SetColumn, 995  
wxListCtrl::SetColumnWidth, 995  
wxListCtrl::SetImageList, 995  
wxListCtrl::SetItem, 995  
wxListCtrl::SetItemBackgroundColour, 997  
wxListCtrl::SetItemColumnImage, 998  
wxListCtrl::SetItemCount, 997  
wxListCtrl::SetItemData, 997  
wxListCtrl::SetItemFont, 997  
wxListCtrl::SetItemImage, 997  
wxListCtrl::SetItemPosition, 998  
wxListCtrl::SetItemPtrData, 998  
wxListCtrl::SetItemState, 998  
wxListCtrl::SetItemText, 998  
wxListCtrl::SetItemTextColour, 998  
wxListCtrl::SetSingleStyle, 998  
wxListCtrl::SetTextColour, 999  
wxListCtrl::SetWindowStyleFlag, 999  
wxListCtrl::SortItems, 999  
wxListCtrl::wxListCtrl, 982  
wxListEvent, 1001  
wxListEvent::GetCacheFrom, 1001  
wxListEvent::GetCacheTo, 1001  
wxListEvent::GetColumn, 1002  
wxListEvent::GetData, 1002  
wxListEvent::GetImage, 1002  
wxListEvent::GetIndex, 1001  
wxListEvent::GetItem, 1002  
wxListEvent::GetKeyCode, 1001  
wxListEvent::GetLabel, 1002  
wxListEvent::GetMask, 1002  
wxListEvent::GetPoint, 1002

wxCommandEvent::GetText, 1002  
wxCommandEvent::IsEditCancelled, 1003  
wxCommandEvent::wxCommandEvent, 1001  
wxListItem, 1003  
wxListItem::Clear, 1003  
wxListItem::GetAlign, 1003  
wxListItem::GetBackgroundColour, 1003  
wxListItem::GetColumn, 1003  
wxListItem::GetData, 1004  
wxListItem::GetFont, 1004  
wxListItem::GetId, 1004  
wxListItem::GetImage, 1004  
wxListItem::GetMask, 1004  
wxListItem::GetState, 1004  
wxListItem::GetText, 1005  
wxListItem::GetTextColour, 1005  
wxListItem::GetWidth, 1005  
wxListItem::SetAlign, 1005  
wxListItem::SetBackgroundColour, 1005  
wxListItem::SetColumn, 1005  
wxListItem::SetData, 1005  
wxListItem::SetFont, 1006  
wxListItem::SetId, 1006  
wxListItem::SetImage, 1006  
wxListItem::SetMask, 1006  
wxListItem::SetState, 1006  
wxListItem::SetStateMask, 1006  
wxListItem::SetText, 1006  
wxListItem::SetTextColour, 1007  
wxListItem::SetWidth, 1007  
wxListItem::wxListItem, 1003  
wxListItemAttr, 1007  
wxListItemAttr::GetBackgroundColour, 1007  
wxListItemAttr::GetFont, 1007  
wxListItemAttr::GetTextColour, 1008  
wxListItemAttr::HasBackgroundColour, 1008  
wxListItemAttr::HasFont, 1008  
wxListItemAttr::HasTextColour, 1008  
wxListItemAttr::SetBackgroundColour, 1008  
wxListItemAttr::SetFont, 1008  
wxListItemAttr::SetTextColour, 1008  
wxListItemAttr::wxListItemAttr, 1007  
wxListView::ClearColumnImage, 1009  
wxListView::Focus, 1009  
wxListView::GetFirstSelected, 1009  
wxListView::GetFocusedItem, 1009  
wxListView::GetNextSelected, 1010  
wxListView::IsSelected, 1010  
wxListView::Select, 1010  
wxListView::SetColumnImage, 1010  
wxLL, 1952  
wxLoadUserResource, 1959  
wxLocale, 1012  
wxLocale::~~wxLocale, 1012  
wxLocale::AddCatalog, 1012  
wxLocale::AddCatalogLookupPathPrefix, 1013  
wxLocale::AddLanguage, 1013  
wxLocale::FindLanguageInfo, 1014  
wxLocale::GetCanonicalName, 1014  
wxLocale::GetHeaderValue, 1016  
wxLocale::GetLanguage, 1014  
wxLocale::GetLanguageInfo, 1015  
wxLocale::GetLanguageName, 1015  
wxLocale::GetLocale, 1015  
wxLocale::GetName, 1015  
wxLocale::GetString, 1015  
wxLocale::GetSysName, 1016  
wxLocale::GetSystemEncoding, 1016  
wxLocale::GetSystemEncodingName, 1016  
wxLocale::GetSystemLanguage, 1016  
wxLocale::Init, 1017  
wxLocale::IsAvailable, 1018  
wxLocale::IsLoaded, 1018  
wxLocale::IsOk, 1018  
wxLocale::wxLocale, 1012  
wxLOCALE\_CONV\_ENCODING, 1017  
wxLOCALE\_LOAD\_DEFAULT, 1017  
wxLog::AddTraceMask, 1021  
wxLog::ClearTraceMasks, 1021  
wxLog::DoLog, 1023  
wxLog::DoLogString, 1023  
wxLog::DontCreateOnDemand, 1023  
wxLog::Flush, 1023  
wxLog::FlushActive, 1023  
wxLog::GetActiveTarget, 1022  
wxLog::GetLogLevel, 1024  
wxLog::GetRepetitionCounting, 1024  
wxLog::GetTimestamp, 1024  
wxLog::GetTraceMask, 1025  
wxLog::GetTraceMasks, 1021  
wxLog::GetVerbose, 1024  
wxLog::IsAllowedTraceMask, 1025  
wxLog::OnLog, 1022  
wxLog::RemoveTraceMask, 1025  
wxLog::Resume, 1022  
wxLog::SetActiveTarget, 1022  
wxLog::SetLogLevel, 1024  
wxLog::SetRepetitionCounting, 1024  
wxLog::SetTimestamp, 1024  
wxLog::SetTraceMask, 1025  
wxLog::SetVerbose, 1023  
wxLog::Suspend, 1022  
wxLogChain, 1026  
wxLogChain::~~wxLogChain, 1026  
wxLogChain::DetachOldLog, 1026  
wxLogChain::GetOldLog, 1026  
wxLogChain::IsPassingMessages, 1026  
wxLogChain::PassMessages, 1026  
wxLogChain::SetLog, 1026  
wxLogChain::wxLogChain, 1026  
wxLogDebug, 1973  
wxLogError, 1972  
wxLogFatalError, 1972  
wxLogGui, 1027  
wxLogGui::wxLogGui, 1027  
wxLogMessage, 1973  
wxLogNull, 1028  
wxLogNull::~~wxLogNull, 1028  
wxLogNull::wxLogNull, 1028  
wxLogPassThrough::wxLogPassThrough, 1029  
wxLogStatus, 1973  
wxLogStderr, 1029  
wxLogStderr::wxLogStderr, 1029  
wxLogStream, 1030

wxLogStream::wxLogStream, 1030  
wxLogSysError, 1973  
wxLogTextCtrl, 1030  
wxLogTextCtrl::wxLogTextCtrl, 1030  
wxLogTrace, 1974  
wxLogVerbose, 1973  
wxLogWarning, 1972  
wxLogWindow, 1031  
wxLogWindow::GetFrame, 1031  
wxLogWindow::OnFrameClose, 1032  
wxLogWindow::OnFrameCreate, 1031  
wxLogWindow::OnFrameDelete, 1032  
wxLogWindow::Show, 1031  
wxLogWindow::wxLogWindow, 1031  
wxLongLong, 1033  
wxLongLong::Abs, 1034  
wxLongLong::Assign, 1034  
wxLongLong::GetHi, 1034  
wxLongLong::GetLo, 1034  
wxLongLong::GetValue, 1034  
wxLongLong::operator-, 1035  
wxLongLong::operator--, 1035  
wxLongLong::operator+, 1035  
wxLongLong::operator++, 1035  
wxLongLong::operator+=", 1035  
wxLongLong::operator=, 1033, 1034  
wxLongLong::operator=, 1035  
wxLongLong::ToDouble, 1034  
wxLongLong::ToLong, 1034  
wxLongLong::ToString, 1035  
wxLongLong::wxLongLong, 1033  
wxLongLongFmtSpec, 1952  
wxMakeMetafilePlaceable, 1946  
wxMask, 1036  
wxMask::~wxMask, 1037  
wxMask::Create, 1037  
wxMask::wxMask, 1036  
wxMatchWild, 1923  
wxMAXIMIZE, 682, 1047, 1052, 1113  
wxMAXIMIZE\_BOX, 497, 682, 1047, 1052, 1113  
wxMaximizeEvent, 1038  
wxMaximizeEvent::wxMaximizeEvent, 1038  
wxMBConv, 1039  
wxMBConv classes, 2060  
wxMBConv objects, 2060  
wxMBConv::cMB2WC, 1040  
wxMBConv::cMB2WX, 1041  
wxMBConv::cWC2MB, 1041  
wxMBConv::cWC2WX, 1041  
wxMBConv::cWX2MB, 1041  
wxMBConv::cWX2WC, 1041  
wxMBConv::FromWChar, 1042  
wxMBConv::GetMaxMByteLen, 1042  
wxMBConv::GetMByteLen, 1042  
wxMBConv::MB2WC, 1039  
wxMBConv::ToWChar, 1042  
wxMBConv::WC2MB, 1040  
wxMBConv::wxMBConv, 1039  
wxMBConvFile::MB2WC, 1043  
wxMBConvFile::WC2MB, 1043  
wxMBConvUTF16::MB2WC, 1046  
wxMBConvUTF16::WC2MB, 1046  
wxMBConvUTF32::MB2WC, 1046  
wxMBConvUTF32::WC2MB, 1047  
wxMBConvUTF7::MB2WC, 1044  
wxMBConvUTF7::WC2MB, 1044  
wxMBConvUTF8::MB2WC, 1045  
wxMBConvUTF8::WC2MB, 1045  
wxMDIChildFrame, 1048  
wxMDIChildFrame::~wxMDIChildFrame, 1049  
wxMDIChildFrame::Activate, 1049  
wxMDIChildFrame::Create, 1049  
wxMDIChildFrame::Maximize, 1049  
wxMDIChildFrame::Restore, 1050  
wxMDIChildFrame::wxMDIChildFrame, 1048  
wxMDIClientWindow, 1050  
wxMDIClientWindow::~wxMDIClientWindow, 1051  
wxMDIClientWindow::CreateClient, 1051  
wxMDIClientWindow::wxMDIClientWindow, 1050  
wxMDIParentFrame, 1053  
wxMDIParentFrame::~wxMDIParentFrame, 1054  
wxMDIParentFrame::ActivateNext, 1054  
wxMDIParentFrame::ActivatePrevious, 1054  
wxMDIParentFrame::ArrangeIcons, 1054  
wxMDIParentFrame::Cascade, 1055  
wxMDIParentFrame::Create, 1055  
wxMDIParentFrame::GetActiveChild, 1056  
wxMDIParentFrame::GetClientSize, 1055  
wxMDIParentFrame::GetClientWindow, 1056  
wxMDIParentFrame::GetToolBar, 1056  
wxMDIParentFrame::GetWindowMenu, 1056  
wxMDIParentFrame::OnCreateClient, 1056  
wxMDIParentFrame::SetToolBar, 1057  
wxMDIParentFrame::SetWindowMenu, 1057  
wxMDIParentFrame::Tile, 1058  
wxMDIParentFrame::wxMDIParentFrame, 1053  
wxMediaCtrl, 1061  
wxMediaCtrl::Create, 1062  
wxMediaCtrl::GetBestSize, 1063  
wxMediaCtrl::GetPlaybackRate, 1063  
wxMediaCtrl::GetState, 1063  
wxMediaCtrl::GetVolume, 1063  
wxMediaCtrl::Length, 1063  
wxMediaCtrl::Load, 1063, 1064  
wxMediaCtrl::LoadURI, 1064  
wxMediaCtrl::LoadURIWithProxy, 1064  
wxMediaCtrl::Pause, 1064  
wxMediaCtrl::Play, 1064  
wxMediaCtrl::Seek, 1064  
wxMediaCtrl::SetPlaybackRate, 1064  
wxMediaCtrl::SetVolume, 1065  
wxMediaCtrl::ShowPlayerControls, 1065  
wxMediaCtrl::Stop, 1065  
wxMediaCtrl::Tell, 1066  
wxMediaCtrl::wxMediaCtrl, 1061  
wxMemoryBuffer, 1067  
wxMemoryBuffer::AppendByte, 1068  
wxMemoryBuffer::AppendData, 1069  
wxMemoryBuffer::GetAppendBuf, 1068  
wxMemoryBuffer::GetBufSize, 1067  
wxMemoryBuffer::GetData, 1067  
wxMemoryBuffer::GetDataLen, 1067  
wxMemoryBuffer::GetWriteBuf, 1068



---

wxMemoryBuffer::SetBufSize, 1067  
wxMemoryBuffer::SetDataLen, 1068  
wxMemoryBuffer::UngetAppendBuf, 1068  
wxMemoryBuffer::UngetWriteBuf, 1068  
wxMemoryBuffer::wxMemoryBuffer, 1067  
wxMemoryDC, 1070  
wxMemoryDC::SelectObject, 1070  
wxMemoryDC::SelectObjectAsSource, 1070  
wxMemoryDC::wxMemoryDC, 1070  
wxMemoryFSHandler, 1072  
wxMemoryFSHandler::AddFile, 1072  
wxMemoryFSHandler::AddFileWithMimeType, 1072  
wxMemoryFSHandler::RemoveFile, 1072  
wxMemoryFSHandler::wxMemoryFSHandler, 1072  
wxMemoryInputStream, 1073  
wxMemoryInputStream::~wxMemoryInputStream, 1073  
wxMemoryInputStream::GetInputStreamBuffer, 1073  
wxMemoryInputStream::wxMemoryInputStream, 1073  
wxMemoryOutputStream, 1074  
wxMemoryOutputStream::~wxMemoryOutputStream, 1074  
wxMemoryOutputStream::CopyTo, 1074  
wxMemoryOutputStream::GetOutputStreamBuffer, 1074  
wxMemoryOutputStream::wxMemoryOutputStream, 1074  
wxMenu, 1076  
wxMenu::~wxMenu, 1076  
wxMenu::Append, 1076  
wxMenu::AppendCheckItem, 1078  
wxMenu::AppendRadioItem, 1079  
wxMenu::AppendSeparator, 1079  
wxMenu::AppendSubMenu, 1079  
wxMenu::Break, 1079  
wxMenu::Check, 1079  
wxMenu::Delete, 1080  
wxMenu::Destroy, 1080  
wxMenu::Enable, 1080  
wxMenu::FindItem, 1081  
wxMenu::FindItemByPosition, 1082  
wxMenu::GetHelpString, 1082  
wxMenu::GetLabel, 1082  
wxMenu::GetLabelText, 1082  
wxMenu::GetMenuItemCount, 1083  
wxMenu::GetMenuItems, 1083  
wxMenu::GetTitle, 1083  
wxMenu::Insert, 1083  
wxMenu::InsertCheckItem, 1084  
wxMenu::InsertRadioItem, 1084  
wxMenu::InsertSeparator, 1084  
wxMenu::IsChecked, 1084  
wxMenu::IsEnabled, 1085  
wxMenu::Prepend, 1085  
wxMenu::PrependCheckItem, 1085  
wxMenu::PrependRadioItem, 1085  
wxMenu::PrependSeparator, 1086  
wxMenu::Remove, 1086  
wxMenu::SetHelpString, 1086  
wxMenu::SetLabel, 1087  
wxMenu::SetTitle, 1087  
wxMenu::UpdateUI, 1087  
wxMenu::wxMenu, 1076  
wxMenuBar, 1088  
wxMenuBar::~wxMenuBar, 1089  
wxMenuBar::Append, 1089  
wxMenuBar::Check, 1089  
wxMenuBar::Enable, 1090  
wxMenuBar::EnableTop, 1090  
wxMenuBar::FindItem, 1091  
wxMenuBar::FindMenu, 1091  
wxMenuBar::FindMenuItem, 1091  
wxMenuBar::GetHelpString, 1092  
wxMenuBar::GetLabel, 1092  
wxMenuBar::GetLabelText, 1092  
wxMenuBar::GetMenu, 1093  
wxMenuBar::GetMenuItemCount, 1093  
wxMenuBar::GetMenuItemLabel, 1093  
wxMenuBar::GetMenuItemLabelText, 1094  
wxMenuBar::Insert, 1094  
wxMenuBar::IsChecked, 1095  
wxMenuBar::IsEnabled, 1095  
wxMenuBar::Refresh, 1095  
wxMenuBar::Remove, 1095  
wxMenuBar::Replace, 1095  
wxMenuBar::SetHelpString, 1096  
wxMenuBar::SetLabel, 1096  
wxMenuBar::SetLabelText, 1097  
wxMenuBar::SetMenuItemLabel, 1097  
wxMenuBar::wxMenuBar, 1088  
wxMenuEvent, 1098  
wxMenuEvent::GetMenu, 1099  
wxMenuEvent::GetMenuId, 1099  
wxMenuEvent::IsPopup, 1099  
wxMenuEvent::wxMenuEvent, 1098  
wxMenuItem, 1100  
wxMenuItem::~wxMenuItem, 1101  
wxMenuItem::Check, 1101  
wxMenuItem::Enable, 1101  
wxMenuItem::GetBackgroundColour, 1101  
wxMenuItem::GetBitmap, 1101  
wxMenuItem::GetFont, 1101  
wxMenuItem::GetHelp, 1101  
wxMenuItem::GetId, 1102  
wxMenuItem::GetItemLabel, 1102  
wxMenuItem::GetItemLabelText, 1102  
wxMenuItem::GetKind, 1102  
wxMenuItem::GetLabel, 1102  
wxMenuItem::GetLabelFromText, 1103  
wxMenuItem::GetLabelText, 1102  
wxMenuItem::GetMarginWidth, 1103  
wxMenuItem::GetMenu, 1103  
wxMenuItem::GetName, 1103  
wxMenuItem::GetSubMenu, 1104  
wxMenuItem::GetText, 1103  
wxMenuItem::GetTextColour, 1104  
wxMenuItem::IsCheckable, 1104  
wxMenuItem::IsChecked, 1104  
wxMenuItem::IsEnabled, 1104  
wxMenuItem::IsSeparator, 1104

- 
- wxMenuItem::IsSubMenu, 1104
  - wxMenuItem::SetBackgroundColour, 1104
  - wxMenuItem::SetBitmap, 1105
  - wxMenuItem::SetBitmaps, 1105
  - wxMenuItem::SetFont, 1105
  - wxMenuItem::SetHelp, 1105
  - wxMenuItem::SetItemLabel, 1105
  - wxMenuItem::SetMarginWidth, 1105
  - wxMenuItem::SetMenu, 1105
  - wxMenuItem::SetSubMenu, 1105
  - wxMenuItem::SetText, 1106
  - wxMenuItem::SetTextColour, 1106
  - wxMenuItem::wxMenuItem, 1099
  - wxMessageBox, 1942
  - wxMessageDialog, 1106
  - wxMessageDialog overview, 2131
  - wxMessageDialog::~wxMessageDialog, 1107
  - wxMessageDialog::ShowModal, 1107
  - wxMessageDialog::wxMessageDialog, 1106
  - wxMetafile, 1108
  - wxMetafile::~wxMetafile, 1108
  - wxMetafile::IsOk, 1108
  - wxMetafile::Play, 1108
  - wxMetafile::SetClipboard, 1108
  - wxMetafile::wxMetafile, 1108
  - wxMetafileDC, 1110
  - wxMetafileDC::~wxMetafileDC, 1110
  - wxMetafileDC::Close, 1110
  - wxMetafileDC::wxMetafileDC, 1110
  - wxMicroSleep, 1978
  - wxMilliSleep, 1978
  - wxMimeTypesManager, 1112
  - wxMimeTypesManager::~wxMimeTypesManager, 1112
  - wxMimeTypesManager::AddFallbacks, 1112
  - wxMimeTypesManager::GetFileTypeFromExtension, 1112
  - wxMimeTypesManager::GetFileTypeFromMimeType, 1112
  - wxMimeTypesManager::IsOfType, 1112
  - wxMimeTypesManager::ReadMailcap, 1113
  - wxMimeTypesManager::ReadMimeTypes, 1113
  - wxMimeTypesManager::wxMimeTypesManager, 1112
  - wxMiniFrame, 1114
  - wxMiniFrame::~wxMiniFrame, 1115
  - wxMiniFrame::Create, 1115
  - wxMiniFrame::wxMiniFrame, 1114
  - wxMINIMIZE, 682, 1047, 1052, 1113
  - wxMINIMIZE\_BOX, 497, 682, 1047, 1052, 1113
  - wxMirrorDC, 1116
  - wxMirrorDC::wxMirrorDC, 1116
  - wxMkdir, 1924
  - wxModule, 1117
  - wxModule::~wxModule, 1117
  - wxModule::AddDependency, 1117
  - wxModule::OnExit, 1117
  - wxModule::OnInit, 1118
  - wxModule::wxModule, 1117
  - wxMouseCaptureChangedEvent, 1118
  - wxMouseCaptureChangedEvent::wxMouseCaptureChangedEvent, 1118
  - wxMouseCaptureLostEvent, 1119
  - wxMouseCaptureLostEvent::wxMouseCaptureLostEvent, 1119
  - wxMouseEvent, 1122
  - wxMouseEvent::AltDown, 1123
  - wxMouseEvent::Button, 1123
  - wxMouseEvent::ButtonDClick, 1123
  - wxMouseEvent::ButtonDown, 1123
  - wxMouseEvent::ButtonUp, 1123
  - wxMouseEvent::CmdDown, 1124
  - wxMouseEvent::ControlDown, 1124
  - wxMouseEvent::Dragging, 1124
  - wxMouseEvent::Entering, 1124
  - wxMouseEvent::GetButton, 1124
  - wxMouseEvent::GetLinesPerAction, 1125
  - wxMouseEvent::GetLogicalPosition, 1125
  - wxMouseEvent::GetPosition, 1124
  - wxMouseEvent::GetWheelDelta, 1125
  - wxMouseEvent::GetWheelRotation, 1125
  - wxMouseEvent::GetX, 1125
  - wxMouseEvent::GetY, 1125
  - wxMouseEvent::IsButton, 1126
  - wxMouseEvent::IsPageScroll, 1126
  - wxMouseEvent::Leaving, 1126
  - wxMouseEvent::LeftDClick, 1126
  - wxMouseEvent::LeftDown, 1126
  - wxMouseEvent::LeftIsDown, 1126
  - wxMouseEvent::LeftUp, 1127
  - wxMouseEvent::m\_altDown, 1121
  - wxMouseEvent::m\_controlDown, 1121
  - wxMouseEvent::m\_leftDown, 1121
  - wxMouseEvent::m\_linesPerAction, 1122
  - wxMouseEvent::m\_metaDown, 1121
  - wxMouseEvent::m\_middleDown, 1121
  - wxMouseEvent::m\_rightDown, 1121
  - wxMouseEvent::m\_shiftDown, 1121
  - wxMouseEvent::m\_wheelDelta, 1122
  - wxMouseEvent::m\_wheelRotation, 1122
  - wxMouseEvent::m\_x, 1122
  - wxMouseEvent::m\_y, 1122
  - wxMouseEvent::MetaDown, 1127
  - wxMouseEvent::MiddleDClick, 1127
  - wxMouseEvent::MiddleDown, 1127
  - wxMouseEvent::MiddleIsDown, 1127
  - wxMouseEvent::MiddleUp, 1127
  - wxMouseEvent::Moving, 1127
  - wxMouseEvent::RightDClick, 1127
  - wxMouseEvent::RightDown, 1128
  - wxMouseEvent::RightIsDown, 1128
  - wxMouseEvent::RightUp, 1128
  - wxMouseEvent::ShiftDown, 1128
  - wxMouseEvent::wxMouseEvent, 1122
  - wxMoveEvent, 1129
  - wxMoveEvent::GetPosition, 1129
  - wxMoveEvent::wxMoveEvent, 1129
  - wxMultiChoiceDialog, 1129
  - wxMultiChoiceDialog overview, 2131
  - wxMultiChoiceDialog::GetSelections, 1130
  - wxMultiChoiceDialog::SetSelections, 1130
  - wxMultiChoiceDialog::ShowModal, 1131
  - wxMultiChoiceDialog::wxMultiChoiceDialog, 1129
  - wxMutex, 1132
-

- wxMutex::~wxMutex, 1133
- wxMutex::Lock, 1133
- wxMutex::TryLock, 1133
- wxMutex::Unlock, 1133
- wxMutex::wxMutex, 1132
- wxMutexGuiEnter, 1919
- wxMutexGuiLeave, 1919
- wxMutexLocker, 1134
- wxMutexLocker::~wxMutexLocker, 1134
- wxMutexLocker::IsOk, 1134
- wxMutexLocker::wxMutexLocker, 1134
- wxNB\_BOTTOM, 1136
- wxNB\_FIXEDWIDTH, 1136
- wxNB\_FLAT, 1136
- wxNB\_LEFT, 1136
- wxNB\_MULTILINE, 1136
- wxNB\_NOPAGETHHEME, 1136
- wxNB\_RIGHT, 1136
- wxNB\_TOP, 1136
- wxNewId, 1953
- wxNO\_3D, 498
- wxNO\_BORDER, 164
- wxNO\_FULL\_REPAINT\_ON\_RESIZE, 1797
- wxNode<T>::GetData, 1135
- wxNode<T>::GetNext, 1135
- wxNode<T>::GetPrevious, 1135
- wxNode<T>::IndexOf, 1135
- wxNode<T>::SetData, 1135
- wxNotebook, 1137
- wxNotebook::~wxNotebook, 1138
- wxNotebook::AddPage, 1138
- wxNotebook::AdvanceSelection, 1139
- wxNotebook::AssignImageList, 1139
- wxNotebook::ChangeSelection, 1144
- wxNotebook::Create, 1139
- wxNotebook::DeleteAllPages, 1139
- wxNotebook::DeletePage, 1139
- wxNotebook::GetCurrentPage, 1140
- wxNotebook::GetImageList, 1140
- wxNotebook::GetPage, 1140
- wxNotebook::GetPageCount, 1140
- wxNotebook::GetPageImage, 1140
- wxNotebook::GetPageText, 1140
- wxNotebook::GetRowCount, 1140
- wxNotebook::GetSelection, 1140
- wxNotebook::GetThemeBackgroundColour, 1141
- wxNotebook::HitTest, 1141
- wxNotebook::InsertPage, 1142
- wxNotebook::OnSelChange, 1142
- wxNotebook::RemovePage, 1142
- wxNotebook::SetImageList, 1143
- wxNotebook::SetPadding, 1143
- wxNotebook::SetPageImage, 1143
- wxNotebook::SetPageSize, 1143
- wxNotebook::SetPageText, 1143
- wxNotebook::SetSelection, 1143
- wxNotebook::wxNotebook, 1137
- wxNotebookEvent, 1145
- wxNotebookEvent::GetOldSelection, 1145
- wxNotebookEvent::GetSelection, 1145
- wxNotebookEvent::SetOldSelection, 1145
- wxNotebookEvent::SetSelection, 1145
- wxNotebookEvent::wxNotebookEvent, 1145
- wxNotebookSizer, 1146
- wxNotebookSizer::GetNotebook, 1146
- wxNotebookSizer::wxNotebookSizer, 1146
- wxNotifyEvent, 1147
- wxNotifyEvent::Allow, 1147
- wxNotifyEvent::IsAllowed, 1147
- wxNotifyEvent::Veto, 1147
- wxNotifyEvent::wxNotifyEvent, 1147
- wxNow, 1979
- wxObjArray, 78
- wxObjArray::Detach, 80
- wxObject, 1148
- wxObject::~wxObject, 1148
- wxObject::Dump, 1148
- wxObject::GetClassInfo, 1149
- wxObject::GetRefData, 1149
- wxObject::IsKindOf, 1149
- wxObject::IsSameAs, 1150
- wxObject::m\_refData, 1148
- wxObject::operator delete, 1151
- wxObject::operator new, 1151
- wxObject::Ref, 1150
- wxObject::SetRefData, 1150
- wxObject::UnRef, 1150
- wxObject::UnShare, 1151
- wxObject::wxObject, 1148
- wxObjectRefData, 1152
- wxObjectRefData::~wxObjectRefData, 1152
- wxObjectRefData::GetRefCount, 1152
- wxObjectRefData::wxObjectRefData, 1151
- wxODBC - Basic Step-By-Step Guide, 2158
- wxODBC - Compiling, 2157
- wxODBC - Configuring your system for ODBC use, 2157
- wxODBC - Known Issues, 2167
- wxODBC - Sample Code, 2169
- wxODBC Where To Start, 2154
- wxODCB\_DCLICK\_CYCLES, 1152
- wxODCB\_STD\_CONTROL\_PAINT, 1152
- wxON\_BLOCK\_EXIT, 1953
- wxON\_BLOCK\_EXIT\_OBJ, 1953
- wxON\_BLOCK\_EXIT\_OBJ0, 1953
- wxON\_BLOCK\_EXIT\_OBJ1, 1953
- wxON\_BLOCK\_EXIT\_OBJ2, 1953
- wxON\_BLOCK\_EXIT0, 1953
- wxON\_BLOCK\_EXIT1, 1953
- wxON\_BLOCK\_EXIT2, 1953
- wxOnAssert, 1980
- wxOpenClipboard, 1950
- wxOutputStream, 1156
- wxOutputStream::~wxOutputStream, 1156
- wxOutputStream::Close, 1156
- wxOutputStream::LastWrite, 1157
- wxOutputStream::PutC, 1157
- wxOutputStream::SeekO, 1157
- wxOutputStream::TellO, 1157
- wxOutputStream::Write, 1157
- wxOutputStream::wxOutputStream, 1156
- wxOwnerDrawnComboBox, 1153
- wxOwnerDrawnComboBox::~wxOwnerDrawnCo  
mboBox, 1154

- wxOwnerDrawnComboBox::Create, 1154
- wxOwnerDrawnComboBox::GetWidestItem, 1154
- wxOwnerDrawnComboBox::GetWidestItemWidth, 1155
- wxOwnerDrawnComboBox::OnDrawBackground, 1155
- wxOwnerDrawnComboBox::OnDrawItem, 1155
- wxOwnerDrawnComboBox::OnMeasureItem, 1156
- wxOwnerDrawnComboBox::OnMeasureItemWidth, 1156
- wxOwnerDrawnComboBox::wxOwnerDrawnComboBox, 1153
- wxPageSetupDialog, 1159
- wxPageSetupDialog* (p. 1155), 2148
- wxPageSetupDialog::~wxPageSetupDialog, 1159
- wxPageSetupDialog::GetPageSetupData, 1159
- wxPageSetupDialog::ShowModal, 1159
- wxPageSetupDialog::wxPageSetupDialog, 1159
- wxPageSetupDialogData, 1159, 1160
- wxPageSetupDialogData* (p. 1156), 2148
- wxPageSetupDialogData::~wxPageSetupDialogData, 1160
- wxPageSetupDialogData::EnableHelp, 1160
- wxPageSetupDialogData::EnableMargins, 1160
- wxPageSetupDialogData::EnableOrientation, 1160
- wxPageSetupDialogData::EnablePaper, 1160
- wxPageSetupDialogData::EnablePrinter, 1160
- wxPageSetupDialogData::GetDefaultInfo, 1161
- wxPageSetupDialogData::GetDefaultMinMargins, 1160
- wxPageSetupDialogData::GetEnableHelp, 1161
- wxPageSetupDialogData::GetEnableMargins, 1161
- wxPageSetupDialogData::GetEnableOrientation, 1161
- wxPageSetupDialogData::GetEnablePaper, 1161
- wxPageSetupDialogData::GetEnablePrinter, 1161
- wxPageSetupDialogData::GetMarginBottomRight, 1161
- wxPageSetupDialogData::GetMarginTopLeft, 1161
- wxPageSetupDialogData::GetMinMarginBottomRight, 1162
- wxPageSetupDialogData::GetMinMarginTopLeft, 1162
- wxPageSetupDialogData::GetPaperId, 1162
- wxPageSetupDialogData::GetPaperSize, 1162
- wxPageSetupDialogData::GetPrintData, 1162
- wxPageSetupDialogData::IsOk, 1162
- wxPageSetupDialogData::operator =, 1164
- wxPageSetupDialogData::SetDefaultInfo, 1162
- wxPageSetupDialogData::SetDefaultMinMargins, 1162
- wxPageSetupDialogData::SetMarginBottomRight, 1163
- wxPageSetupDialogData::SetMarginTopLeft, 1163
- wxPageSetupDialogData::SetMinMarginBottomRight, 1163
- wxPageSetupDialogData::SetMinMarginTopLeft, 1163
- wxPageSetupDialogData::SetPaperId, 1163
- wxPageSetupDialogData::SetPaperSize, 1163
- wxPageSetupDialogData::SetPrintData, 1163
- wxPageSetupDialogData::wxPageSetupDialogData, 1159
- wxPaintDC, 1164
- wxPaintDC::wxPaintDC, 1164
- wxPaintEvent, 1166
- wxPaintEvent::wxPaintEvent, 1166
- wxPalette, 1167
- wxPalette::~wxPalette, 1168
- wxPalette::Create, 1168
- wxPalette::GetColoursCount, 1168
- wxPalette::GetPixel, 1168
- wxPalette::GetRGB, 1169
- wxPalette::IsOk, 1169
- wxPalette::operator =, 1170
- wxPalette::wxPalette, 1167
- wxPanel, 1170, 1171
- wxPanel::~wxPanel, 1171
- wxPanel::Create, 1171
- wxPanel::InitDialog, 1172
- wxPanel::OnSysColourChanged, 1172
- wxPanel::SetFocus, 1172
- wxPanel::SetFocusIgnoringChildren, 1172
- wxPanel::wxPanel, 1170
- wxPaperSize, 1204
- wxParseCommonDialogsFilter, 1924
- wxPasswordEntryDialog, 1173
- wxPasswordEntryDialog overview, 2131
- wxPasswordEntryDialog::wxPasswordEntryDialog, 1173
- wxPathList, 1174
- wxPathList::Add, 1174
- wxPathList::AddEnvList, 1174
- wxPathList::EnsureFileAccessible, 1175
- wxPathList::FindAbsoluteValidPath, 1175
- wxPathList::FindValidPath, 1175
- wxPathList::wxPathList, 1174
- wxPathOnly, 1922
- wxPB\_USE\_TEXTCTRL, 1184, 1185
- wxPCXHandler, 906
- wxPD\_APP\_MODAL, 1230
- wxPD\_AUTO\_HIDE, 1230
- wxPD\_CAN\_ABORT, 1231
- wxPD\_CAN\_SKIP, 1231
- wxPD\_ELAPSED\_TIME, 1231
- wxPD\_ESTIMATED\_TIME, 1231
- wxPD\_REMAINING\_TIME, 1231
- wxPD\_SMOOTH, 1231
- wxPen, 1176, 1177
- wxPen::~wxPen, 1178
- wxPen::GetCap, 1179
- wxPen::GetColour, 1179
- wxPen::GetDashes, 1179
- wxPen::GetJoin, 1179
- wxPen::GetStipple, 1179
- wxPen::GetStyle, 1180
- wxPen::GetWidth, 1180
- wxPen::IsOk, 1180

---

wxPen::operator !=, 1182  
wxPen::operator =, 1181  
wxPen::operator ==, 1182  
wxPen::SetCap, 1180  
wxPen::SetColour, 1180  
wxPen::SetDashes, 1180  
wxPen::SetJoin, 1181  
wxPen::SetStipple, 1181  
wxPen::SetStyle, 1181  
wxPen::SetWidth, 1181  
wxPen::wxPen, 1176  
wxPenList, 1182  
wxPenList::FindOrCreatePen, 1183  
wxPenList::wxPenList, 1182  
wxPickerBase::GetInternalMargin, 1184  
wxPickerBase::GetPickerCtrlProportion, 1184  
wxPickerBase::GetTextCtrl, 1185  
wxPickerBase::GetTextCtrlProportion, 1184  
wxPickerBase::HasTextCtrl, 1185  
wxPickerBase::IsPickerCtrlGrowable, 1185  
wxPickerBase::IsTextCtrlGrowable, 1185  
wxPickerBase::SetInternalMargin, 1184  
wxPickerBase::SetPickerCtrlGrowable, 1185  
wxPickerBase::SetPickerCtrlProportion, 1184  
wxPickerBase::SetTextCtrlGrowable, 1185  
wxPickerBase::SetTextCtrlProportion, 1184  
wxPlatformInfo, 1188  
wxPlatformInfo::CheckOSVersion, 1188  
wxPlatformInfo::CheckToolkitVersion, 1188  
wxPlatformInfo::Get, 1188  
wxPlatformInfo::GetArch, 1188  
wxPlatformInfo::GetArchitecture, 1189  
wxPlatformInfo::GetArchName, 1189  
wxPlatformInfo::GetEndianness, 1189  
wxPlatformInfo::GetEndiannessName, 1189  
wxPlatformInfo::GetOperatingSystemFamilyName, 1190  
wxPlatformInfo::GetOperatingSystemId, 1190  
wxPlatformInfo::GetOperatingSystemIdName, 1190  
wxPlatformInfo::GetOSMajorVersion, 1189  
wxPlatformInfo::GetOSMinorVersion, 1190  
wxPlatformInfo::GetPortId, 1190  
wxPlatformInfo::GetPortIdName, 1191  
wxPlatformInfo::GetPortIdShortName, 1191  
wxPlatformInfo::GetToolkitMajorVersion, 1191  
wxPlatformInfo::GetToolkitMinorVersion, 1191  
wxPlatformInfo::IsOk, 1192  
wxPlatformInfo::IsUsingUniversalWidgets, 1192  
wxPlatformInfo::operator !=, 1193  
wxPlatformInfo::operator ==, 1193  
wxPlatformInfo::SetArchitecture, 1192  
wxPlatformInfo::SetEndianness, 1192  
wxPlatformInfo::SetOperatingSystemId, 1192  
wxPlatformInfo::SetOSVersion, 1192  
wxPlatformInfo::SetPortId, 1192  
wxPlatformInfo::SetToolkitVersion, 1193  
wxPlatformInfo::wxPlatformInfo, 1188  
wxPLURAL, 1933  
wxPNGHandler, 906  
wxPNMHandler, 906  
wxPoint, 1193  
wxPoint::wxPoint, 1193  
wxPoint::x, 1194  
wxPoint::y, 1194  
wxPostDelete, 1960  
wxPostEvent, 1960  
wxPostScriptDC, 1195  
wxPostScriptDC (p. 1191), 2147  
wxPostScriptDC::GetResolution, 1195  
wxPostScriptDC::SetResolution, 1195  
wxPostScriptDC::wxPostScriptDC, 1195  
wxPowerEvent::Veto, 1196  
wxPreviewCanvas, 1197  
wxPreviewCanvas::~~wxPreviewCanvas, 1197  
wxPreviewCanvas::OnPaint, 1197  
wxPreviewCanvas::wxPreviewCanvas, 1197  
wxPreviewControlBar, 1198  
wxPreviewControlBar::~~wxPreviewControlBar, 1198  
wxPreviewControlBar::CreateButtons, 1198  
wxPreviewControlBar::GetPrintPreview, 1198  
wxPreviewControlBar::GetZoomControl, 1198  
wxPreviewControlBar::SetZoomControl, 1199  
wxPreviewControlBar::wxPreviewControlBar, 1198  
wxPreviewFrame, 1199  
wxPreviewFrame::~~wxPreviewFrame, 1199  
wxPreviewFrame::CreateCanvas, 1200  
wxPreviewFrame::CreateControlBar, 1199  
wxPreviewFrame::Initialize, 1200  
wxPreviewFrame::OnCloseWindow, 1200  
wxPreviewFrame::wxPreviewFrame, 1199  
wxPrintData, 1201  
wxPrintData (p. 1197), 2148  
wxPrintData::~~wxPrintData, 1201  
wxPrintData::GetBin, 1201  
wxPrintData::GetCollate, 1201  
wxPrintData::GetColour, 1202  
wxPrintData::GetDuplex, 1202  
wxPrintData::GetNoCopies, 1202  
wxPrintData::GetOrientation, 1202  
wxPrintData::GetPaperId, 1202  
wxPrintData::GetPrinterName, 1202  
wxPrintData::GetQuality, 1202  
wxPrintData::IsOk, 1203  
wxPrintData::operator =, 1206  
wxPrintData::SetBin, 1203  
wxPrintData::SetCollate, 1203  
wxPrintData::SetColour, 1203  
wxPrintData::SetDuplex, 1204  
wxPrintData::SetNoCopies, 1204  
wxPrintData::SetOrientation, 1204  
wxPrintData::SetPaperId, 1204  
wxPrintData::SetPrinterName, 1205  
wxPrintData::SetQuality, 1206  
wxPrintData::wxPrintData, 1201  
wxPrintDialog, 1207  
wxPrintDialog (p. 1203), 2147  
wxPrintDialog overview, 2130  
wxPrintDialog::~~wxPrintDialog, 1207  
wxPrintDialog::GetPrintDC, 1207  
wxPrintDialog::GetPrintDialogData, 1207  
wxPrintDialog::ShowModal, 1207

---

wxPrintDialog::wxPrintDialog, 1206  
wxPrintDialogData, 1208  
  *wxPrintDialogData* (p. 1204), 2148  
wxPrintDialogData::~~wxPrintDialogData, 1208  
wxPrintDialogData::EnableHelp, 1208  
wxPrintDialogData::EnablePageNumbers, 1208  
wxPrintDialogData::EnablePrintToFile, 1208  
wxPrintDialogData::EnableSelection, 1208  
wxPrintDialogData::GetAllPages, 1209  
wxPrintDialogData::GetCollate, 1209  
wxPrintDialogData::GetFromPage, 1209  
wxPrintDialogData::GetMaxPage, 1209  
wxPrintDialogData::GetMinPage, 1209  
wxPrintDialogData::GetNoCopies, 1209  
wxPrintDialogData::GetPrintData, 1209  
wxPrintDialogData::GetPrintToFile, 1209  
wxPrintDialogData::GetSelection, 1209  
wxPrintDialogData::GetToPage, 1210  
wxPrintDialogData::IsOk, 1210  
wxPrintDialogData::operator =, 1211  
wxPrintDialogData::SetCollate, 1210  
wxPrintDialogData::SetFromPage, 1210  
wxPrintDialogData::SetMaxPage, 1210  
wxPrintDialogData::SetMinPage, 1210  
wxPrintDialogData::SetNoCopies, 1210  
wxPrintDialogData::SetPrintData, 1210  
wxPrintDialogData::SetPrintToFile, 1211  
wxPrintDialogData::SetSelection, 1211  
wxPrintDialogData::SetSetupDialog, 1211  
wxPrintDialogData::SetToPage, 1211  
wxPrintDialogData::wxPrintDialogData, 1208  
wxPrinter, 1212  
  *wxPrinter* (p. 1208), 2147  
wxPrinter::CreateAbortWindow, 1212  
wxPrinter::GetAbort, 1212  
wxPrinter::GetLastError, 1212  
wxPrinter::GetPrintDialogData, 1213  
wxPrinter::Print, 1213  
wxPrinter::PrintDialog, 1213  
wxPrinter::ReportError, 1213  
wxPrinter::Setup, 1213  
wxPrinter::wxPrinter, 1212  
wxPrinterDC, 1214  
  *wxPrinterDC* (p. 1210), 2147  
wxPrinterDC::GetPaperRect, 1214  
wxPrinterDC::wxPrinterDC, 1214  
wxPrintout, 1215  
  *wxPrintout* (p. 1211), 2146  
wxPrintout::~~wxPrintout, 1215  
wxPrintout::FitThisSizeToPage, 1218  
wxPrintout::FitThisSizeToPageMargins, 1218  
wxPrintout::FitThisSizeToPaper, 1218  
wxPrintout::GetDC, 1215  
wxPrintout::GetLogicalPageMarginsRect, 1220  
wxPrintout::GetLogicalPageRect, 1219  
wxPrintout::GetLogicalPaperRect, 1219  
wxPrintout::GetPageInfo, 1216  
wxPrintout::GetPageSizeMM, 1216  
wxPrintout::GetPageSizePixels, 1216  
wxPrintout::GetPaperRectPixels, 1216  
wxPrintout::GetPPIPrinter, 1217  
wxPrintout::GetPPIScreen, 1217  
wxPrintout::GetTitle, 1217  
wxPrintout::HasPage, 1217  
wxPrintout::IsPreview, 1218  
wxPrintout::MapScreenSizeToDevice, 1219  
wxPrintout::MapScreenSizeToPage, 1219  
wxPrintout::MapScreenSizeToPageMargins, 1219  
wxPrintout::MapScreenSizeToPaper, 1218  
wxPrintout::OffsetLogicalOrigin, 1220  
wxPrintout::OnBeginDocument, 1220  
wxPrintout::OnBeginPrinting, 1220  
wxPrintout::OnEndDocument, 1220  
wxPrintout::OnEndPrinting, 1221  
wxPrintout::OnPreparePrinting, 1221  
wxPrintout::OnPrintPage, 1221  
wxPrintout::SetLogicalOrigin, 1220  
wxPrintout::wxPrintout, 1215  
wxPrintPreview, 1221  
  *wxPrintPreview* (p. 1218), 2147  
wxPrintPreview::~~wxPrintPreview, 1222  
wxPrintPreview::GetCanvas, 1222  
wxPrintPreview::GetCurrentPage, 1222  
wxPrintPreview::GetFrame, 1222  
wxPrintPreview::GetMaxPage, 1222  
wxPrintPreview::GetMinPage, 1222  
wxPrintPreview::GetPrintout, 1223  
wxPrintPreview::GetPrintoutForPrinting, 1223  
wxPrintPreview::IsOk, 1223  
wxPrintPreview::PaintPage, 1223  
wxPrintPreview::Print, 1223  
wxPrintPreview::RenderPage, 1223  
wxPrintPreview::SetCanvas, 1223  
wxPrintPreview::SetCurrentPage, 1224  
wxPrintPreview::SetFrame, 1224  
wxPrintPreview::SetPrintout, 1224  
wxPrintPreview::SetZoom, 1224  
wxPrintPreview::wxPrintPreview, 1221  
wxProcess, 1225  
wxProcess::~~wxProcess, 1225  
wxProcess::CloseOutput, 1225  
wxProcess::Detach, 1226  
wxProcess::Exists, 1228  
wxProcess::GetErrorStream, 1226  
wxProcess::GetInputStream, 1226  
wxProcess::GetOutputStream, 1226  
wxProcess::GetPid, 1229  
wxProcess::IsErrorAvailable, 1226  
wxProcess::IsInputAvailable, 1226  
wxProcess::IsInputOpened, 1227  
wxProcess::Kill, 1227  
wxProcess::OnTerminate, 1228  
wxProcess::Open, 1228  
wxProcess::Redirect, 1229  
wxProcess::wxProcess, 1225  
wxProcessEvent, 1230  
wxProcessEvent::GetExitCode, 1230  
wxProcessEvent::GetPid, 1230  
wxProcessEvent::wxProcessEvent, 1230  
wxProgressDialog, 1231  
wxProgressDialog::~~wxProgressDialog, 1232  
wxProgressDialog::Pulse, 1232  
wxProgressDialog::Resume, 1232

- wxProgressDialog::Update, 1232
- wxProgressDialog::wxProgressDialog, 1231
- wxPropertySheetDialog, 1233
- wxPropertySheetDialog::AddBookCtrl, 1233
- wxPropertySheetDialog::Create, 1234
- wxPropertySheetDialog::CreateBookCtrl, 1234
- wxPropertySheetDialog::CreateButtons, 1234
- wxPropertySheetDialog::GetBookCtrl, 1234
- wxPropertySheetDialog::GetInnerSizer, 1234
- wxPropertySheetDialog::GetSheetStyle, 1234
- wxPropertySheetDialog::LayoutDialog, 1234
- wxPropertySheetDialog::SetBookCtrl, 1235
- wxPropertySheetDialog::SetInnerSizer, 1235
- wxPropertySheetDialog::SetSheetStyle, 1235
- wxPropertySheetDialog::wxPropertySheetDialog, 1233
- wxProtocol::Abort, 1236
- wxProtocol::GetContentType, 1237
- wxProtocol::GetError, 1236
- wxProtocol::GetInputStream, 1236
- wxProtocol::Reconnect, 1236
- wxProtocol::SetPassword, 1237
- wxProtocol::SetUser, 1237
- wxQuantize, 1238
- wxQuantize::DoQuantize, 1238
- wxQuantize::Quantize, 1238
- wxQuantize::wxQuantize, 1238
- wxQueryLayoutInfoEvent, 1239
- wxQueryLayoutInfoEvent::GetAlignment, 1239
- wxQueryLayoutInfoEvent::GetFlags, 1239
- wxQueryLayoutInfoEvent::GetOrientation, 1240
- wxQueryLayoutInfoEvent::GetRequestedLength, 1240
- wxQueryLayoutInfoEvent::GetSize, 1240
- wxQueryLayoutInfoEvent::SetAlignment, 1240
- wxQueryLayoutInfoEvent::SetFlags, 1240
- wxQueryLayoutInfoEvent::SetOrientation, 1240
- wxQueryLayoutInfoEvent::SetRequestedLength, 1240
- wxQueryLayoutInfoEvent::SetSize, 1240
- wxQueryLayoutInfoEvent::wxQueryLayoutInfoEvent, 1239
- wxRA\_SPECIFY\_COLS, 1241
- wxRA\_SPECIFY\_ROWS, 1241
- wxRA\_USE\_CHECKBOX, 1241
- wxRadioBox, 1241, 1242
- wxRadioBox::~wxRadioBox, 1243
- wxRadioBox::Create, 1243
- wxRadioBox::Enable, 1243
- wxRadioBox::FindString, 1244
- wxRadioBox::GetColumnCount, 1244
- wxRadioBox::GetItemFromPoint, 1245
- wxRadioBox::GetItemHelpText, 1244
- wxRadioBox::GetItemToolTip, 1244
- wxRadioBox::GetLabel, 1245
- wxRadioBox::GetRowCount, 1245
- wxRadioBox::GetSelection, 1245
- wxRadioBox::GetString, 1246
- wxRadioBox::GetStringSelection, 1246
- wxRadioBox::IsItemEnabled, 1246
- wxRadioBox::IsItemShown, 1246
- wxRadioBox::SetItemHelpText, 1247
- wxRadioBox::SetItemToolTip, 1248
- wxRadioBox::SetLabel, 1247
- wxRadioBox::SetSelection, 1247
- wxRadioBox::SetStringSelection, 1248
- wxRadioBox::Show, 1248
- wxRadioBox::wxRadioBox, 1241
- wxRadioButton, 1250
- wxRadioButton::~wxRadioButton, 1251
- wxRadioButton::Create, 1251
- wxRadioButton::GetValue, 1251
- wxRadioButton::SetValue, 1251
- wxRadioButton::wxRadioButton, 1250
- wxB\_GROUP, 1249
- wxB\_SINGLE, 1249
- wxB\_USE\_CHECKBOX, 1250
- wxRealPoint, 1252
- wxRealPoint::wxRealPoint, 1252
- wxRect, 1252, 1253
- wxRect::CentreIn, 1253
- wxRect::Contains, 1253
- wxRect::Deflate, 1254
- wxRect::GetBottom, 1254
- wxRect::GetBottomLeft, 1255
- wxRect::GetBottomRight, 1255
- wxRect::GetHeight, 1254
- wxRect::GetLeft, 1254
- wxRect::GetPosition, 1254
- wxRect::GetRight, 1255
- wxRect::GetSize, 1255
- wxRect::GetTop, 1255
- wxRect::GetTopLeft, 1255
- wxRect::GetTopRight, 1255
- wxRect::GetWidth, 1255
- wxRect::GetX, 1256
- wxRect::GetY, 1256
- wxRect::height, 1253
- wxRect::Inflate, 1256
- wxRect::Intersect, 1257
- wxRect::Intersects, 1257
- wxRect::IsEmpty, 1257
- wxRect::Offset, 1257
- wxRect::operator !=, 1258
- wxRect::operator =, 1258
- wxRect::operator ==, 1258
- wxRect::SetHeight, 1257
- wxRect::SetSize, 1257
- wxRect::SetWidth, 1258
- wxRect::SetX, 1258
- wxRect::SetY, 1258
- wxRect::Union, 1258
- wxRect::width, 1253
- wxRect::wxRect, 1252
- wxRect::x, 1253
- wxRect::y, 1253
- wxRecursionGuard, 1259
- wxRecursionGuard::~wxRecursionGuard, 1259
- wxRecursionGuard::IsInside, 1260
- wxRecursionGuard::wxRecursionGuard, 1259
- wxRegEx, 1262
- wxRegEx::~wxRegEx, 1262
- wxRegEx::Compile, 1262
- wxRegEx::GetMatch, 1263

- wxRegEx::GetMatchCount, 1263
- wxRegEx::IsValid, 1262
- wxRegEx::Matches, 1263
- wxRegEx::Replace, 1264
- wxRegEx::ReplaceAll, 1264
- wxRegEx::ReplaceFirst, 1264
- wxRegEx::wxRegEx, 1262
- wxRegion, 1265
- wxRegion::~wxRegion, 1265
- wxRegion::Clear, 1265
- wxRegion::Contains, 1266
- wxRegion::ConvertToBitmap, 1266
- wxRegion::GetBox, 1266
- wxRegion::Intersect, 1266
- wxRegion::IsEmpty, 1267
- wxRegion::IsEqual, 1267
- wxRegion::Offset, 1267
- wxRegion::operator =, 1269
- wxRegion::Subtract, 1267
- wxRegion::Union, 1268
- wxRegion::wxRegion, 1265
- wxRegion::Xor, 1268
- wxRegionIterator, 1269
- wxRegionIterator::GetH, 1270
- wxRegionIterator::GetHeight, 1270
- wxRegionIterator::GetRect, 1270
- wxRegionIterator::GetW, 1270
- wxRegionIterator::GetWidth, 1270
- wxRegionIterator::GetX, 1270
- wxRegionIterator::GetY, 1270
- wxRegionIterator::HaveRects, 1270
- wxRegionIterator::operator ++, 1271
- wxRegionIterator::operator bool, 1271
- wxRegionIterator::Reset, 1271
- wxRegionIterator::wxRegionIterator, 1269
- wxRegisterClipboardFormat, 1950
- wxRegisterId, 1954
- wxRegKey, 1272
- wxRegKey::Close, 1273
- wxRegKey::Create, 1273
- wxRegKey::DeleteKey, 1273
- wxRegKey::DeleteSelf, 1273
- wxRegKey::DeleteValue, 1273
- wxRegKey::Exists, 1273
- wxRegKey::GetFirstKey, 1273
- wxRegKey::GetFirstValue, 1273
- wxRegKey::GetKeyInfo, 1274
- wxRegKey::GetName, 1273
- wxRegKey::GetNextKey, 1274
- wxRegKey::GetNextValue, 1274
- wxRegKey::HasSubKey, 1274
- wxRegKey::HasSubKeys, 1275
- wxRegKey::HasValue, 1274
- wxRegKey::HasValues, 1274
- wxRegKey::IsEmpty, 1275
- wxRegKey::IsOpened, 1275
- wxRegKey::Open, 1275
- wxRegKey::QueryValue, 1275
- wxRegKey::Rename, 1275
- wxRegKey::RenameValue, 1275
- wxRegKey::SetValue, 1275
- wxRegKey::wxRegKey, 1272
- wxRelationship, 939
- wxRemoveFile, 1924
- wxRenameFile, 1924
- wxRendererNative::~wxRendererNative, 1277
- wxRendererNative::DrawCheckBox, 1277
- wxRendererNative::DrawComboBoxDropDown, 1277
- wxRendererNative::DrawDropArrow, 1278
- wxRendererNative::DrawHeaderButton, 1278
- wxRendererNative::DrawItemSelectionRect, 1278
- wxRendererNative::DrawPushButton, 1278
- wxRendererNative::DrawSplitterBorder, 1279
- wxRendererNative::DrawSplitterSash, 1279
- wxRendererNative::DrawTreeItemButton, 1279
- wxRendererNative::Get, 1279
- wxRendererNative::GetDefault, 1279
- wxRendererNative::GetGeneric, 1279
- wxRendererNative::GetHeaderButtonHeight, 1279
- wxRendererNative::GetSplitterParams, 1280
- wxRendererNative::GetVersion, 1280
- wxRendererNative::Load, 1280
- wxRendererNative::Set, 1280
- wxRendererVersion::age, 1281
- wxRendererVersion::IsCompatible, 1281
- wxRendererVersion::version, 1281
- wxRESERVE\_SPACE\_EVEN\_IF\_HIDDEN, 1447
- wxRESIZE\_BORDER, 497, 683, 1047, 1052, 1114
- wxRETAINED, 1416
- wxRICHTEXT\_HANDLER\_INCLUDE\_STYLESHEET, 1390
- wxRICHTEXT\_HANDLER\_NO\_HEADER\_FOOTER, 1363
- wxRICHTEXT\_HANDLER\_SAVE\_IMAGES\_TO\_BASE64, 1363
- wxRICHTEXT\_HANDLER\_SAVE\_IMAGES\_TO\_FILES, 1363
- wxRICHTEXT\_HANDLER\_SAVE\_IMAGES\_TO\_MEMORY, 1363
- wxRICHTEXT\_ORGANISER\_APPLY\_STYLES, 1384
- wxRICHTEXT\_ORGANISER\_BROWSE, 1385
- wxRICHTEXT\_ORGANISER\_BROWSE\_NUMBERING, 1385
- wxRICHTEXT\_ORGANISER\_CREATE\_STYLES, 1384
- wxRICHTEXT\_ORGANISER\_DELETE\_STYLES, 1384
- wxRICHTEXT\_ORGANISER\_EDIT\_STYLES, 1384
- wxRICHTEXT\_ORGANISER\_OK\_CANCEL, 1384
- wxRICHTEXT\_ORGANISER\_ORGANISE, 1385
- wxRICHTEXT\_ORGANISER\_RENAME\_STYLES, 1384
- wxRICHTEXT\_ORGANISER\_RENUMBER, 1384
- wxRICHTEXT\_ORGANISER\_SHOW\_ALL, 1385
- wxRICHTEXT\_ORGANISER\_SHOW\_CHARACTER, 1384
- wxRICHTEXT\_ORGANISER\_SHOW\_LIST, 1385
- wxRICHTEXT\_ORGANISER\_SHOW\_PARAGRA



- PH, 1385
- wxRichTextAttr, 1284
- wxRichTextAttr::Apply, 1284
- wxRichTextAttr::Combine, 1284
- wxRichTextAttr::CreateFont, 1284
- wxRichTextAttr::GetAlignment, 1284
- wxRichTextAttr::GetBackgroundColour, 1285
- wxRichTextAttr::GetBulletFont, 1285
- wxRichTextAttr::GetBulletName, 1285
- wxRichTextAttr::GetBulletNumber, 1285
- wxRichTextAttr::GetBulletStyle, 1285
- wxRichTextAttr::GetBulletText, 1286
- wxRichTextAttr::GetCharacterStyleName, 1286
- wxRichTextAttr::GetFlags, 1286
- wxRichTextAttr::GetFontAttributes, 1286
- wxRichTextAttr::GetFontFaceName, 1286
- wxRichTextAttr::GetFontSize, 1286
- wxRichTextAttr::GetFontStyle, 1286
- wxRichTextAttr::GetFontUnderlined, 1286
- wxRichTextAttr::GetFontWeight, 1286
- wxRichTextAttr::GetLeftIndent, 1287
- wxRichTextAttr::GetLeftSubIndent, 1287
- wxRichTextAttr::GetLineSpacing, 1287
- wxRichTextAttr::GetListStyleName, 1287
- wxRichTextAttr::GetOutlineLevel, 1287
- wxRichTextAttr::GetParagraphSpacingAfter, 1287
- wxRichTextAttr::GetParagraphSpacingBefore, 1287
- wxRichTextAttr::GetParagraphStyleName, 1287
- wxRichTextAttr::GetRightIndent, 1288
- wxRichTextAttr::GetTabs, 1288
- wxRichTextAttr::GetTextColour, 1288
- wxRichTextAttr::GetTextEffectFlags, 1288
- wxRichTextAttr::GetTextEffects, 1288
- wxRichTextAttr::GetURL, 1288
- wxRichTextAttr::HasAlignment, 1288
- wxRichTextAttr::HasBackgroundColour, 1288
- wxRichTextAttr::HasBulletName, 1289
- wxRichTextAttr::HasBulletNumber, 1289
- wxRichTextAttr::HasBulletStyle, 1289
- wxRichTextAttr::HasBulletText, 1289
- wxRichTextAttr::HasCharacterStyleName, 1289
- wxRichTextAttr::HasFlag, 1289
- wxRichTextAttr::HasFont, 1289
- wxRichTextAttr::HasFontFaceName, 1289
- wxRichTextAttr::HasFontItalic, 1290
- wxRichTextAttr::HasFontSize, 1291
- wxRichTextAttr::HasFontUnderlined, 1291
- wxRichTextAttr::HasFontWeight, 1291
- wxRichTextAttr::HasLeftIndent, 1290
- wxRichTextAttr::HasLineSpacing, 1290
- wxRichTextAttr::HasListStyleName, 1290
- wxRichTextAttr::HasOutlineLevel, 1290
- wxRichTextAttr::HasPageBreak, 1290
- wxRichTextAttr::HasParagraphSpacingAfter, 1290
- wxRichTextAttr::HasParagraphSpacingBefore, 1290
- wxRichTextAttr::HasParagraphStyleName, 1291
- wxRichTextAttr::HasRightIndent, 1291
- wxRichTextAttr::HasTabs, 1291
- wxRichTextAttr::HasTextColour, 1291
- wxRichTextAttr::HasTextEffects, 1291
- wxRichTextAttr::HasURL, 1291
- wxRichTextAttr::IsCharacterStyle, 1292
- wxRichTextAttr::IsDefault, 1292
- wxRichTextAttr::IsParagraphStyle, 1292
- wxRichTextAttr::operator=, 1298
- wxRichTextAttr::SetAlignment, 1292
- wxRichTextAttr::SetBackgroundColour, 1292
- wxRichTextAttr::SetBulletFont, 1293
- wxRichTextAttr::SetBulletName, 1293
- wxRichTextAttr::SetBulletNumber, 1293
- wxRichTextAttr::SetBulletStyle, 1293
- wxRichTextAttr::SetBulletText, 1293
- wxRichTextAttr::SetCharacterStyleName, 1294
- wxRichTextAttr::SetFlags, 1294
- wxRichTextAttr::SetFontFaceName, 1294
- wxRichTextAttr::SetFontSize, 1295
- wxRichTextAttr::SetFontStyle, 1295
- wxRichTextAttr::SetFontUnderlined, 1295
- wxRichTextAttr::SetFontWeight, 1295
- wxRichTextAttr::SetLeftIndent, 1295
- wxRichTextAttr::SetLineSpacing, 1295
- wxRichTextAttr::SetListStyleName, 1296
- wxRichTextAttr::SetOutlineLevel, 1296
- wxRichTextAttr::SetPageBreak, 1296
- wxRichTextAttr::SetParagraphSpacingAfter, 1296
- wxRichTextAttr::SetParagraphSpacingBefore, 1296
- wxRichTextAttr::SetParagraphStyleName, 1296
- wxRichTextAttr::SetRightIndent, 1296
- wxRichTextAttr::SetTabs, 1296
- wxRichTextAttr::SetTextColour, 1297
- wxRichTextAttr::SetTextEffectFlags, 1297
- wxRichTextAttr::SetTextEffects, 1297
- wxRichTextAttr::SetURL, 1297
- wxRichTextAttr::wxRichTextAttr, 1284
- wxRichTextAttr::wxTextAttrEx, 1298
- wxRichTextBuffer, 1298
- wxRichTextBuffer::~wxRichTextBuffer, 1299
- wxRichTextBuffer::AddEventHandler, 1299
- wxRichTextBuffer::AddHandler, 1299
- wxRichTextBuffer::AddParagraph, 1299
- wxRichTextBuffer::BatchingUndo, 1299
- wxRichTextBuffer::BeginAlignment, 1299
- wxRichTextBuffer::BeginBatchUndo, 1299
- wxRichTextBuffer::BeginBold, 1300
- wxRichTextBuffer::BeginCharacterStyle, 1300
- wxRichTextBuffer::BeginFont, 1300
- wxRichTextBuffer::BeginFontSize, 1300
- wxRichTextBuffer::BeginItalic, 1300
- wxRichTextBuffer::BeginLeftIndent, 1300
- wxRichTextBuffer::BeginLineSpacing, 1300
- wxRichTextBuffer::BeginListStyle, 1301
- wxRichTextBuffer::BeginNumberedBullet, 1301
- wxRichTextBuffer::BeginParagraphSpacing, 1302
- wxRichTextBuffer::BeginParagraphStyle, 1302
- wxRichTextBuffer::BeginRightIndent, 1302
- wxRichTextBuffer::BeginStandardBullet, 1302
- wxRichTextBuffer::BeginStyle, 1302
- wxRichTextBuffer::BeginSuppressUndo, 1302
- wxRichTextBuffer::BeginSymbolBullet, 1302

- wxRichTextBuffer::BeginTextColour, 1303
- wxRichTextBuffer::BeginUnderline, 1303
- wxRichTextBuffer::BeginURL, 1303
- wxRichTextBuffer::CanPasteFromClipboard, 1303
- wxRichTextBuffer::CleanUpHandlers, 1303
- wxRichTextBuffer::Clear, 1303
- wxRichTextBuffer::ClearListStyle, 1303
- wxRichTextBuffer::ClearStyleStack, 1304
- wxRichTextBuffer::Clone, 1304
- wxRichTextBuffer::Copy, 1304
- wxRichTextBuffer::CopyToClipboard, 1304
- wxRichTextBuffer::DeleteRangeWithUndo, 1304
- wxRichTextBuffer::Dump, 1304
- wxRichTextBuffer::EndAlignment, 1304
- wxRichTextBuffer::EndAllStyles, 1305
- wxRichTextBuffer::EndBatchUndo, 1305
- wxRichTextBuffer::EndBold, 1305
- wxRichTextBuffer::EndCharacterStyle, 1305
- wxRichTextBuffer::EndFont, 1305
- wxRichTextBuffer::EndFontSize, 1305
- wxRichTextBuffer::EndItalic, 1305
- wxRichTextBuffer::EndLeftIndent, 1305
- wxRichTextBuffer::EndLineSpacing, 1306
- wxRichTextBuffer::EndListStyle, 1306
- wxRichTextBuffer::EndNumberedBullet, 1306
- wxRichTextBuffer::EndParagraphSpacing, 1306
- wxRichTextBuffer::EndParagraphStyle, 1306
- wxRichTextBuffer::EndRightIndent, 1306
- wxRichTextBuffer::EndStandardBullet, 1307
- wxRichTextBuffer::EndStyle, 1306
- wxRichTextBuffer::EndSuppressUndo, 1306
- wxRichTextBuffer::EndSymbolBullet, 1306
- wxRichTextBuffer::EndTextColour, 1307
- wxRichTextBuffer::EndUnderline, 1307
- wxRichTextBuffer::EndURL, 1307
- wxRichTextBuffer::FindHandler, 1307
- wxRichTextBuffer::FindHandlerFilenameOrType, 1307
- wxRichTextBuffer::GetBasicStyle, 1307
- wxRichTextBuffer::GetBatchedCommand, 1308
- wxRichTextBuffer::GetCommandProcessor, 1308
- wxRichTextBuffer::GetDefaultStyle, 1308
- wxRichTextBuffer::GetExtWildcard, 1308
- wxRichTextBuffer::GetHandlers, 1308
- wxRichTextBuffer::GetRenderer, 1308
- wxRichTextBuffer::GetStyle, 1308
- wxRichTextBuffer::GetStyleForRange, 1309
- wxRichTextBuffer::GetStyleSheet, 1309
- wxRichTextBuffer::GetStyleStackSize, 1309
- wxRichTextBuffer::GetUncombinedStyle, 1309
- wxRichTextBuffer::HitTest, 1310
- wxRichTextBuffer::Init, 1310
- wxRichTextBuffer::InitStandardHandlers, 1310
- wxRichTextBuffer::InsertHandler, 1310
- wxRichTextBuffer::InsertImageWithUndo, 1310
- wxRichTextBuffer::InsertNewlineWithUndo, 1311
- wxRichTextBuffer::InsertTextWithUndo, 1311
- wxRichTextBuffer::IsModified, 1311
- wxRichTextBuffer::LoadFile, 1311
- wxRichTextBuffer::Modify, 1311
- wxRichTextBuffer::NumberList, 1311
- wxRichTextBuffer::PasteFromClipboard, 1312
- wxRichTextBuffer::PromoteList, 1312
- wxRichTextBuffer::RemoveEventHandler, 1312
- wxRichTextBuffer::RemoveHandler, 1313
- wxRichTextBuffer::ResetAndClearCommands, 1313
- wxRichTextBuffer::SaveFile, 1313
- wxRichTextBuffer::SetBasicStyle, 1313
- wxRichTextBuffer::SetDefaultStyle, 1313
- wxRichTextBuffer::SetListStyle, 1313
- wxRichTextBuffer::SetRenderer, 1314
- wxRichTextBuffer::SetStyle, 1314
- wxRichTextBuffer::SetStyleSheet, 1315
- wxRichTextBuffer::SubmitAction, 1315
- wxRichTextBuffer::SuppressingUndo, 1315
- wxRichTextBuffer::wxRichTextBuffer, 1298
- wxRichTextCharacterStyleDefinition, 1316
- wxRichTextCharacterStyleDefinition::~wxRichTextCharacterStyleDefinition, 1316
- wxRichTextCharacterStyleDefinition::wxRichTextCharacterStyleDefinition, 1315
- wxRichTextCtrl, 1316
- wxRichTextCtrl and styles, 2193
- wxRichTextCtrl dialogs, 2195
- wxRichTextCtrl roadmap, 2196
- wxRichTextCtrl::~wxRichTextCtrl, 1316
- wxRichTextCtrl::AddImage, 1317
- wxRichTextCtrl::AddParagraph, 1317
- wxRichTextCtrl::AppendText, 1317
- wxRichTextCtrl::ApplyAlignmentToSelection, 1317
- wxRichTextCtrl::ApplyBoldToSelection, 1317
- wxRichTextCtrl::ApplyItalicToSelection, 1317
- wxRichTextCtrl::ApplyStyle, 1317
- wxRichTextCtrl::ApplyStyleSheet, 1317
- wxRichTextCtrl::ApplyUnderlineToSelection, 1318
- wxRichTextCtrl::BatchingUndo, 1318
- wxRichTextCtrl::BeginAlignment, 1318
- wxRichTextCtrl::BeginBatchUndo, 1318
- wxRichTextCtrl::BeginBold, 1318
- wxRichTextCtrl::BeginCharacterStyle, 1318
- wxRichTextCtrl::BeginFont, 1318
- wxRichTextCtrl::BeginFontSize, 1318
- wxRichTextCtrl::BeginItalic, 1319
- wxRichTextCtrl::BeginLeftIndent, 1319
- wxRichTextCtrl::BeginLineSpacing, 1319
- wxRichTextCtrl::BeginListStyle, 1319
- wxRichTextCtrl::BeginNumberedBullet, 1319
- wxRichTextCtrl::BeginParagraphSpacing, 1320
- wxRichTextCtrl::BeginParagraphStyle, 1320
- wxRichTextCtrl::BeginRightIndent, 1320
- wxRichTextCtrl::BeginStyle, 1320
- wxRichTextCtrl::BeginSuppressUndo, 1321
- wxRichTextCtrl::BeginSymbolBullet, 1321
- wxRichTextCtrl::BeginTextColour, 1321
- wxRichTextCtrl::BeginUnderline, 1321
- wxRichTextCtrl::BeginURL, 1321
- wxRichTextCtrl::CanCopy, 1321
- wxRichTextCtrl::CanCut, 1321
- wxRichTextCtrl::CanDeleteSelection, 1322
- wxRichTextCtrl::CanPaste, 1322

- wxRichTextCtrl::CanRedo, 1322
- wxRichTextCtrl::CanUndo, 1322
- wxRichTextCtrl::Clear, 1322
- wxRichTextCtrl::ClearListStyle, 1322
- wxRichTextCtrl::Command, 1323
- wxRichTextCtrl::Copy, 1323
- wxRichTextCtrl::Create, 1323
- wxRichTextCtrl::Cut, 1323
- wxRichTextCtrl::Delete, 1323
- wxRichTextCtrl::DeleteSelectedContent, 1323
- wxRichTextCtrl::DeleteSelection, 1323
- wxRichTextCtrl::DiscardEdits, 1323
- wxRichTextCtrl::DoGetBestSize, 1324
- wxRichTextCtrl::EndAlignment, 1324
- wxRichTextCtrl::EndAllStyles, 1324
- wxRichTextCtrl::EndBatchUndo, 1324
- wxRichTextCtrl::EndBold, 1324
- wxRichTextCtrl::EndCharacterStyle, 1324
- wxRichTextCtrl::EndFont, 1324
- wxRichTextCtrl::EndFontSize, 1324
- wxRichTextCtrl::EndItalic, 1325
- wxRichTextCtrl::EndLeftIndent, 1325
- wxRichTextCtrl::EndLineSpacing, 1325
- wxRichTextCtrl::EndListStyle, 1325
- wxRichTextCtrl::EndNumberedBullet, 1325
- wxRichTextCtrl::EndParagraphSpacing, 1325
- wxRichTextCtrl::EndParagraphStyle, 1325
- wxRichTextCtrl::EndRightIndent, 1325
- wxRichTextCtrl::EndStyle, 1325
- wxRichTextCtrl::EndSuppressUndo, 1326
- wxRichTextCtrl::EndSymbolBullet, 1326
- wxRichTextCtrl::EndTextColour, 1326
- wxRichTextCtrl::EndUnderline, 1326
- wxRichTextCtrl::EndURL, 1326
- wxRichTextCtrl::ExtendSelection, 1326
- wxRichTextCtrl::FindNextWordPosition, 1326
- wxRichTextCtrl::Freeze, 1326
- wxRichTextCtrl::GetBasicStyle, 1327
- wxRichTextCtrl::GetBuffer, 1327
- wxRichTextCtrl::GetCaretPosition, 1327
- wxRichTextCtrl::GetCaretPositionForIndex, 1327
- wxRichTextCtrl::GetCommandProcessor, 1327
- wxRichTextCtrl::GetDefaultStyleEx, 1327
- wxRichTextCtrl::GetDelayedLayoutThreshold, 1327
- wxRichTextCtrl::GetFilename, 1328
- wxRichTextCtrl::GetFirstVisiblePosition, 1328
- wxRichTextCtrl::GetHandlerFlags, 1328
- wxRichTextCtrl::GetInsertionPoint, 1328
- wxRichTextCtrl::GetLastPosition, 1328
- wxRichTextCtrl::GetLineLength, 1328
- wxRichTextCtrl::GetLineText, 1328
- wxRichTextCtrl::GetLogicalPoint, 1328
- wxRichTextCtrl::GetNumberOfLines, 1329
- wxRichTextCtrl::GetPhysicalPoint, 1329
- wxRichTextCtrl::GetRange, 1329
- wxRichTextCtrl::GetSelection, 1329
- wxRichTextCtrl::GetSelectionRange, 1329
- wxRichTextCtrl::GetStringSelection, 1329
- wxRichTextCtrl::GetStyle, 1329
- wxRichTextCtrl::GetStyleForRange, 1330
- wxRichTextCtrl::GetStyleSheet, 1330
- wxRichTextCtrl::GetUncombinedStyle, 1330
- wxRichTextCtrl::GetValue, 1330
- wxRichTextCtrl::GetVisibleLineForCaretPosition, 1330
- wxRichTextCtrl::HasCharacterAttributes, 1331
- wxRichTextCtrl::HasParagraphAttributes, 1331
- wxRichTextCtrl::HasSelection, 1331
- wxRichTextCtrl::HitTest, 1331
- wxRichTextCtrl::Init, 1331
- wxRichTextCtrl::InitCommandEvent, 1332
- wxRichTextCtrl::IsDefaultStyleShowing, 1332
- wxRichTextCtrl::IsEditable, 1332
- wxRichTextCtrl::IsFrozen, 1333
- wxRichTextCtrl::IsModified, 1333
- wxRichTextCtrl::IsMultiLine, 1333
- wxRichTextCtrl::IsPositionVisible, 1333
- wxRichTextCtrl::IsSelectionAligned, 1333
- wxRichTextCtrl::IsSelectionBold, 1333
- wxRichTextCtrl::IsSelectionItalics, 1333
- wxRichTextCtrl::IsSelectionUnderlined, 1333
- wxRichTextCtrl::IsSingleLine, 1334
- wxRichTextCtrl::KeyboardNavigate, 1334
- wxRichTextCtrl::LayoutContent, 1334
- wxRichTextCtrl::LineBreak, 1334
- wxRichTextCtrl::LoadFile, 1334
- wxRichTextCtrl::MarkDirty, 1334
- wxRichTextCtrl::MoveCaret, 1334
- wxRichTextCtrl::MoveCaretBack, 1334
- wxRichTextCtrl::MoveCaretForward, 1335
- wxRichTextCtrl::MoveDown, 1335
- wxRichTextCtrl::MoveEnd, 1335
- wxRichTextCtrl::MoveHome, 1335
- wxRichTextCtrl::MoveLeft, 1335
- wxRichTextCtrl::MoveRight, 1335
- wxRichTextCtrl::MoveToLineEnd, 1335
- wxRichTextCtrl::MoveToLineStart, 1336
- wxRichTextCtrl::MoveToParagraphEnd, 1336
- wxRichTextCtrl::MoveToParagraphStart, 1336
- wxRichTextCtrl::MoveUp, 1336
- wxRichTextCtrl::Newline, 1336
- wxRichTextCtrl::NumberList, 1336
- wxRichTextCtrl::OnClear, 1337
- wxRichTextCtrl::OnContextMenu, 1337
- wxRichTextCtrl::OnCopy, 1337
- wxRichTextCtrl::OnCut, 1337
- wxRichTextCtrl::OnDropFiles, 1337
- wxRichTextCtrl::OnPaste, 1337
- wxRichTextCtrl::OnRedo, 1337
- wxRichTextCtrl::OnSelectAll, 1338
- wxRichTextCtrl::OnUndo, 1338
- wxRichTextCtrl::OnUpdateClear, 1338
- wxRichTextCtrl::OnUpdateCopy, 1338
- wxRichTextCtrl::OnUpdateCut, 1338
- wxRichTextCtrl::OnUpdatePaste, 1338
- wxRichTextCtrl::OnUpdateRedo, 1338
- wxRichTextCtrl::OnUpdateSelectAll, 1338
- wxRichTextCtrl::OnUpdateUndo, 1338
- wxRichTextCtrl::PageDown, 1339
- wxRichTextCtrl::PageUp, 1339
- wxRichTextCtrl::PaintBackground, 1339
- wxRichTextCtrl::Paste, 1339
- wxRichTextCtrl::PositionCaret, 1339

- wxRichTextCtrl::PositionToXY, 1339
- wxRichTextCtrl::PromoteList, 1339
- wxRichTextCtrl::Redo, 1340
- wxRichTextCtrl::Remove, 1340
- wxRichTextCtrl::Replace, 1340
- wxRichTextCtrl::SaveFile, 1340
- wxRichTextCtrl::ScrollIntoView, 1340
- wxRichTextCtrl::SelectAll, 1341
- wxRichTextCtrl::SelectNone, 1341
- wxRichTextCtrl::SetAndShowDefaultStyle, 1341
- wxRichTextCtrl::SetBasicStyle, 1341
- wxRichTextCtrl::SetCaretPosition, 1341
- wxRichTextCtrl::SetDefaultStyle, 1341
- wxRichTextCtrl::SetDefaultStyleToCursorStyle, 1341
- wxRichTextCtrl::SetDelayedLayoutThreshold, 1342
- wxRichTextCtrl::SetEditable, 1342
- wxRichTextCtrl::SetFilename, 1342
- wxRichTextCtrl::SetFont, 1342
- wxRichTextCtrl::SetHandlerFlags, 1342
- wxRichTextCtrl::SetInsertionPoint, 1342
- wxRichTextCtrl::SetInsertionPointEnd, 1342
- wxRichTextCtrl::SetListStyle, 1342
- wxRichTextCtrl::SetSelection, 1343
- wxRichTextCtrl::SetSelectionRange, 1343
- wxRichTextCtrl::SetStyle, 1343
- wxRichTextCtrl::SetStyleEx, 1344
- wxRichTextCtrl::SetStyleSheet, 1344
- wxRichTextCtrl::SetupScrollbars, 1345
- wxRichTextCtrl::SetValue, 1344
- wxRichTextCtrl::ShowPosition, 1345
- wxRichTextCtrl::SuppressingUndo, 1345
- wxRichTextCtrl::Thaw, 1345
- wxRichTextCtrl::Undo, 1345
- wxRichTextCtrl::WordLeft, 1345
- wxRichTextCtrl::WordRight, 1345
- wxRichTextCtrl::WriteImage, 1345
- wxRichTextCtrl::WriteText, 1346
- wxRichTextCtrl::wxRichTextCtrl, 1316
- wxRichTextCtrl::XYToPosition, 1346
- wxRichTextEvent, 1348
- wxRichTextEvent::Clone, 1348
- wxRichTextEvent::GetCharacter, 1348
- wxRichTextEvent::GetFlags, 1349
- wxRichTextEvent::GetNewStyleSheet, 1349
- wxRichTextEvent::GetOldStyleSheet, 1349
- wxRichTextEvent::GetPosition, 1349
- wxRichTextEvent::GetRange, 1349
- wxRichTextEvent::SetCharacter, 1349
- wxRichTextEvent::SetFlags, 1349
- wxRichTextEvent::SetNewStyleSheet, 1350
- wxRichTextEvent::SetOldStyleSheet, 1350
- wxRichTextEvent::SetPosition, 1350
- wxRichTextEvent::SetRange, 1350
- wxRichTextEvent::wxRichTextEvent, 1348
- wxRichTextFileHandler, 1350
- wxRichTextFileHandler::CanHandle, 1351
- wxRichTextFileHandler::CanLoad, 1351
- wxRichTextFileHandler::CanSave, 1351
- wxRichTextFileHandler::DoLoadFile, 1351
- wxRichTextFileHandler::DoSaveFile, 1351
- wxRichTextFileHandler::GetEncoding, 1351
- wxRichTextFileHandler::GetExtension, 1351
- wxRichTextFileHandler::GetFlags, 1351
- wxRichTextFileHandler::GetName, 1352
- wxRichTextFileHandler::GetType, 1352
- wxRichTextFileHandler::IsVisible, 1352
- wxRichTextFileHandler::LoadFile, 1352
- wxRichTextFileHandler::SaveFile, 1352
- wxRichTextFileHandler::SetEncoding, 1352
- wxRichTextFileHandler::SetExtension, 1352
- wxRichTextFileHandler::SetFlags, 1352
- wxRichTextFileHandler::SetName, 1353
- wxRichTextFileHandler::SetType, 1353
- wxRichTextFileHandler::SetVisible, 1353
- wxRichTextFileHandler::wxRichTextFileHandler, 1350
- wxRichTextFormattingDialog, 1354
- wxRichTextFormattingDialog::~wxRichTextFormattingDialog, 1355
- wxRichTextFormattingDialog::ApplyStyle, 1355
- wxRichTextFormattingDialog::Create, 1355
- wxRichTextFormattingDialog::GetAttributes, 1355
- wxRichTextFormattingDialog::GetDialog, 1355
- wxRichTextFormattingDialog::GetDialogAttributes, 1355
- wxRichTextFormattingDialog::GetDialogStyleDefinition, 1355
- wxRichTextFormattingDialog::GetFormattingDialogFactory, 1356
- wxRichTextFormattingDialog::GetImageList, 1356
- wxRichTextFormattingDialog::GetStyle, 1356
- wxRichTextFormattingDialog::GetStyleDefinition, 1356
- wxRichTextFormattingDialog::GetStyleSheet, 1356
- wxRichTextFormattingDialog::SetAttributes, 1356
- wxRichTextFormattingDialog::SetFormattingDialogFactory, 1356
- wxRichTextFormattingDialog::SetImageList, 1356
- wxRichTextFormattingDialog::SetStyle, 1357
- wxRichTextFormattingDialog::SetStyleDefinition, 1357
- wxRichTextFormattingDialog::UpdateDisplay, 1357
- wxRichTextFormattingDialog::wxRichTextFormattingDialog, 1354
- wxRichTextFormattingDialogFactory, 1357
- wxRichTextFormattingDialogFactory::~wxRichTextFormattingDialogFactory, 1357
- wxRichTextFormattingDialogFactory::CreateButtons, 1358
- wxRichTextFormattingDialogFactory::CreatePage, 1358
- wxRichTextFormattingDialogFactory::CreatePages, 1358
- wxRichTextFormattingDialogFactory::GetPageId, 1358
- wxRichTextFormattingDialogFactory::GetPageIdCount, 1358
- wxRichTextFormattingDialogFactory::GetPageImage, 1358

- 
- wxRichTextFormattingDialogFactory::SetSheetStyle, 1358
  - wxRichTextFormattingDialogFactory::ShowHelp, 1358
  - wxRichTextFormattingDialogFactory::wxRichTextFormattingDialogFactory, 1357
  - wxRichTextHeaderFooterData, 1359
  - wxRichTextHeaderFooterData::Clear, 1360
  - wxRichTextHeaderFooterData::Copy, 1360
  - wxRichTextHeaderFooterData::GetFont, 1360
  - wxRichTextHeaderFooterData::GetFooterMargin, 1360
  - wxRichTextHeaderFooterData::GetFooterText, 1360
  - wxRichTextHeaderFooterData::GetHeaderMargin, 1360
  - wxRichTextHeaderFooterData::GetHeaderText, 1360
  - wxRichTextHeaderFooterData::GetShowOnFirstPage, 1361
  - wxRichTextHeaderFooterData::GetText, 1361
  - wxRichTextHeaderFooterData::GetTextColour, 1361
  - wxRichTextHeaderFooterData::Init, 1361
  - wxRichTextHeaderFooterData::operator=, 1362
  - wxRichTextHeaderFooterData::SetFont, 1361
  - wxRichTextHeaderFooterData::SetFooterText, 1361
  - wxRichTextHeaderFooterData::SetHeaderText, 1361
  - wxRichTextHeaderFooterData::SetMargins, 1362
  - wxRichTextHeaderFooterData::SetShowOnFirstPage, 1362
  - wxRichTextHeaderFooterData::SetText, 1362
  - wxRichTextHeaderFooterData::SetTextColour, 1362
  - wxRichTextHeaderFooterData::wxRichTextHeaderFooterData, 1359
  - wxRichTextHTMLHandler, 1363
  - wxRichTextHTMLHandler::ClearTemporaryImageLocations, 1363
  - wxRichTextHTMLHandler::DeleteTemporaryImages, 1363
  - wxRichTextHTMLHandler::DoSaveFile, 1364
  - wxRichTextHTMLHandler::GetFontSizeMapping, 1364
  - wxRichTextHTMLHandler::GetTempDir, 1364
  - wxRichTextHTMLHandler::GetTemporaryImageLocations, 1364
  - wxRichTextHTMLHandler::SetFileCounter, 1364
  - wxRichTextHTMLHandler::SetFontSizeMapping, 1364
  - wxRichTextHTMLHandler::SetTempDir, 1365
  - wxRichTextHTMLHandler::SetTemporaryImageLocations, 1365
  - wxRichTextHTMLHandler::wxRichTextHTMLHandler, 1363
  - wxRichTextListStyleDefinition, 1366
  - wxRichTextListStyleDefinition::~wxRichTextListStyleDefinition, 1366
  - wxRichTextListStyleDefinition::CombineWithParagraphStyle, 1366
  - wxRichTextListStyleDefinition::FindLevelForIndent, 1366
  - wxRichTextListStyleDefinition::GetCombinedStyle, 1366
  - wxRichTextListStyleDefinition::GetCombinedStyleForLevel, 1366
  - wxRichTextListStyleDefinition::GetLevelAttributes, 1366
  - wxRichTextListStyleDefinition::GetLevelCount, 1367
  - wxRichTextListStyleDefinition::IsNumbered, 1367
  - wxRichTextListStyleDefinition::SetLevelAttributes, 1367
  - wxRichTextListStyleDefinition::wxRichTextListStyleDefinition, 1365
  - wxRichTextParagraphStyleDefinition, 1367
  - wxRichTextParagraphStyleDefinition::~wxRichTextParagraphStyleDefinition, 1367
  - wxRichTextParagraphStyleDefinition::GetNextStyle, 1368
  - wxRichTextParagraphStyleDefinition::SetNextStyle, 1368
  - wxRichTextParagraphStyleDefinition::wxRichTextParagraphStyleDefinition, 1367
  - wxRichTextPrinting, 1368
  - wxRichTextPrinting::GetFooterText, 1368
  - wxRichTextPrinting::GetHeaderFooterData, 1368
  - wxRichTextPrinting::GetHeaderText, 1369
  - wxRichTextPrinting::GetPageSetupData, 1369
  - wxRichTextPrinting::GetParentWindow, 1369
  - wxRichTextPrinting::GetPreviewRect, 1369
  - wxRichTextPrinting::GetPrintData, 1369
  - wxRichTextPrinting::GetTitle, 1369
  - wxRichTextPrinting::PageSetup, 1369
  - wxRichTextPrinting::PreviewBuffer, 1369
  - wxRichTextPrinting::PreviewFile, 1370
  - wxRichTextPrinting::PrintBuffer, 1370
  - wxRichTextPrinting::PrintFile, 1370
  - wxRichTextPrinting::SetFooterText, 1370
  - wxRichTextPrinting::SetHeaderFooterData, 1370
  - wxRichTextPrinting::SetHeaderFooterFont, 1370
  - wxRichTextPrinting::SetHeaderFooterTextColour, 1370
  - wxRichTextPrinting::SetHeaderText, 1371
  - wxRichTextPrinting::SetPageSetupData, 1371
  - wxRichTextPrinting::SetParentWindow, 1371
  - wxRichTextPrinting::SetPreviewRect, 1371
  - wxRichTextPrinting::SetPrintData, 1371
  - wxRichTextPrinting::SetShowOnFirstPage, 1371
  - wxRichTextPrinting::SetTitle, 1371
  - wxRichTextPrinting::wxRichTextPrinting, 1368
  - wxRichTextPrintout, 1372
  - wxRichTextPrintout::CalculateScaling, 1372
  - wxRichTextPrintout::GetHeaderFooterData, 1372
  - wxRichTextPrintout::GetPageInfo, 1372
  - wxRichTextPrintout::GetRichTextBuffer, 1372
  - wxRichTextPrintout::HasPage, 1372
  - wxRichTextPrintout::OnPreparePrinting, 1372
  - wxRichTextPrintout::OnPrintPage, 1373
  - wxRichTextPrintout::SetHeaderFooterData, 1373
  - wxRichTextPrintout::SetMargins, 1373
  - wxRichTextPrintout::SetRichTextBuffer, 1373
-

- 
- wxRichTextPrintout::wxRichTextPrintout, 1372
  - wxRichTextRange, 1373, 1374
  - wxRichTextRange::~Contains, 1374
  - wxRichTextRange::FromInternal, 1374
  - wxRichTextRange::GetEnd, 1374
  - wxRichTextRange::GetLength, 1374
  - wxRichTextRange::GetStart, 1374
  - wxRichTextRange::IsOutside, 1374
  - wxRichTextRange::IsWithin, 1374
  - wxRichTextRange::LimitTo, 1375
  - wxRichTextRange::operator-, 1375
  - wxRichTextRange::operator+, 1375
  - wxRichTextRange::operator=, 1376
  - wxRichTextRange::operator==, 1376
  - wxRichTextRange::SetEnd, 1375
  - wxRichTextRange::SetRange, 1375
  - wxRichTextRange::SetStart, 1375
  - wxRichTextRange::Swap, 1375
  - wxRichTextRange::ToInternal, 1375
  - wxRichTextRange::wxRichTextRange, 1373
  - wxRichTextStyleComboBoxCtrl, 1378
  - wxRichTextStyleComboBoxCtrl::~~wxRichTextStyleCo  
mboCtrl, 1378
  - wxRichTextStyleComboBoxCtrl::GetRichTextCtrl,  
1378
  - wxRichTextStyleComboBoxCtrl::GetStyleSheet, 1378
  - wxRichTextStyleComboBoxCtrl::SetRichTextCtrl,  
1378
  - wxRichTextStyleComboBoxCtrl::SetStyleSheet, 1379
  - wxRichTextStyleComboBoxCtrl::UpdateStyles, 1379
  - wxRichTextStyleComboBoxCtrl::wxRichTextStyleCo  
mboCtrl, 1378
  - wxRichTextStyleDefinition, 1376
  - wxRichTextStyleDefinition::~~wxRichTextStyleDefi  
nition, 1376
  - wxRichTextStyleDefinition::GetBaseStyle, 1376
  - wxRichTextStyleDefinition::GetDescription, 1377
  - wxRichTextStyleDefinition::GetName, 1377
  - wxRichTextStyleDefinition::GetStyle, 1377
  - wxRichTextStyleDefinition::GetStyleMergedWith  
Base, 1377
  - wxRichTextStyleDefinition::SetBaseStyle, 1377
  - wxRichTextStyleDefinition::SetDescription, 1377
  - wxRichTextStyleDefinition::SetName, 1377
  - wxRichTextStyleDefinition::SetStyle, 1377
  - wxRichTextStyleDefinition::wxRichTextStyleDefin  
ition, 1376
  - wxRICHTEXTSTYLELIST\_HIDE\_TYPE\_SELEC  
TOR, 1382
  - wxRichTextStyleListBox, 1379
  - wxRichTextStyleListBox::~~wxRichTextStyleListBo  
x, 1379
  - wxRichTextStyleListBox::ApplyStyle, 1379
  - wxRichTextStyleListBox::ConvertTenthsMMToPi  
xels, 1380
  - wxRichTextStyleListBox::CreateHTML, 1380
  - wxRichTextStyleListBox::GetApplyOnSelection,  
1380
  - wxRichTextStyleListBox::GetRichTextCtrl, 1380
  - wxRichTextStyleListBox::GetStyle, 1380
  - wxRichTextStyleListBox::GetStyleSheet, 1380
  - wxRichTextStyleListBox::GetStyleType, 1380
  - wxRichTextStyleListBox::OnGetItem, 1380
  - wxRichTextStyleListBox::OnLeftDown, 1380
  - wxRichTextStyleListBox::OnSelect, 1381
  - wxRichTextStyleListBox::SetApplyOnSelection,  
1381
  - wxRichTextStyleListBox::SetRichTextCtrl, 1381
  - wxRichTextStyleListBox::SetStyleSheet, 1381
  - wxRichTextStyleListBox::SetStyleType, 1381
  - wxRichTextStyleListBox::UpdateStyles, 1381
  - wxRichTextStyleListBox::wxRichTextStyleListBox,  
1379
  - wxRichTextStyleListCtrl, 1382
  - wxRichTextStyleListCtrl::Create, 1382
  - wxRichTextStyleListCtrl::GetRichTextCtrl, 1382
  - wxRichTextStyleListCtrl::GetStyleChoice, 1382
  - wxRichTextStyleListCtrl::GetStyleListBox, 1383
  - wxRichTextStyleListCtrl::GetStyleSheet, 1383
  - wxRichTextStyleListCtrl::GetStyleType, 1383
  - wxRichTextStyleListCtrl::SetRichTextCtrl, 1383
  - wxRichTextStyleListCtrl::SetStyleSheet, 1383
  - wxRichTextStyleListCtrl::SetStyleType, 1383
  - wxRichTextStyleListCtrl::UpdateStyles, 1383
  - wxRichTextStyleListCtrl::wxRichTextStyleListCtrl,  
1382
  - wxRichTextStyleOrganiserDialog, 1384
  - wxRichTextStyleOrganiserDialog::ApplyStyle,  
1385
  - wxRichTextStyleOrganiserDialog::Create, 1385
  - wxRichTextStyleOrganiserDialog::GetFlags, 1385
  - wxRichTextStyleOrganiserDialog::GetRestartNu  
mbering, 1385
  - wxRichTextStyleOrganiserDialog::GetRichTextCt  
rl, 1386
  - wxRichTextStyleOrganiserDialog::GetSelectedSt  
yle, 1386
  - wxRichTextStyleOrganiserDialog::GetSelectedSt  
yleDefinition, 1386
  - wxRichTextStyleOrganiserDialog::GetStyleSheet,  
1386
  - wxRichTextStyleOrganiserDialog::SetFlags, 1386
  - wxRichTextStyleOrganiserDialog::SetRestartNu  
mbering, 1386
  - wxRichTextStyleOrganiserDialog::SetRichTextCtr  
l, 1386
  - wxRichTextStyleOrganiserDialog::SetShowToolTi  
ps, 1386
  - wxRichTextStyleOrganiserDialog::SetStyleSheet,  
1387
  - wxRichTextStyleOrganiserDialog::wxRichTextSty  
leOrganiserDialog, 1384, 1385
  - wxRichTextStyleSheet, 1387
  - wxRichTextStyleSheet::~~wxRichTextStyleSheet,  
1387
  - wxRichTextStyleSheet::AddCharacterStyle, 1387
  - wxRichTextStyleSheet::AddListStyle, 1387
  - wxRichTextStyleSheet::AddParagraphStyle, 1388
  - wxRichTextStyleSheet::AddStyle, 1388
  - wxRichTextStyleSheet::DeleteStyles, 1388
  - wxRichTextStyleSheet::FindCharacterStyle, 1388
  - wxRichTextStyleSheet::FindListStyle, 1388
  - wxRichTextStyleSheet::FindParagraphStyle,
-

- 1388
- wxRichTextStyleSheet::FindStyle, 1388
- wxRichTextStyleSheet::GetCharacterStyle, 1388
- wxRichTextStyleSheet::GetCharacterStyleCount, 1389
- wxRichTextStyleSheet::GetDescription, 1389
- wxRichTextStyleSheet::GetListStyle, 1389
- wxRichTextStyleSheet::GetListStyleCount, 1389
- wxRichTextStyleSheet::GetName, 1389
- wxRichTextStyleSheet::GetParagraphStyle, 1389
- wxRichTextStyleSheet::GetParagraphStyleCount, 1389
- wxRichTextStyleSheet::RemoveCharacterStyle, 1389
- wxRichTextStyleSheet::RemoveListStyle, 1389
- wxRichTextStyleSheet::RemoveParagraphStyle, 1390
- wxRichTextStyleSheet::RemoveStyle, 1390
- wxRichTextStyleSheet::SetDescription, 1390
- wxRichTextStyleSheet::SetName, 1390
- wxRichTextStyleSheet::wxRichTextStyleSheet, 1387
- wxRichTextXMLHandler, 1391
- wxRichTextXMLHandler::CanLoad, 1391
- wxRichTextXMLHandler::CanSave, 1391
- wxRichTextXMLHandler::CreateStyle, 1391
- wxRichTextXMLHandler::DoLoadFile, 1391
- wxRichTextXMLHandler::DoSaveFile, 1391
- wxRichTextXMLHandler::ExportXML, 1391
- wxRichTextXMLHandler::GetNodeContent, 1391
- wxRichTextXMLHandler::GetParamNode, 1392
- wxRichTextXMLHandler::GetParamValue, 1392
- wxRichTextXMLHandler::GetStyle, 1392
- wxRichTextXMLHandler::GetText, 1392
- wxRichTextXMLHandler::HasParam, 1392
- wxRichTextXMLHandler::ImportXML, 1392
- wxRichTextXMLHandler::wxRichTextXMLHandler, 1391
- wxRIGHT, 1447
- wxRmdir, 1924
- wxSafeShowMessage, 1975
- wxSafeYield, 1912
- wxSashEvent, 1393
- wxSashEvent::GetDragRect, 1394
- wxSashEvent::GetDragStatus, 1394
- wxSashEvent::GetEdge, 1394
- wxSashEvent::wxSashEvent, 1393
- wxSashLayoutWindow, 1395
- wxSashLayoutWindow::Create, 1396
- wxSashLayoutWindow::GetAlignment, 1396
- wxSashLayoutWindow::GetOrientation, 1396
- wxSashLayoutWindow::OnCalculateLayout, 1396
- wxSashLayoutWindow::OnQueryLayoutInfo, 1397
- wxSashLayoutWindow::SetAlignment, 1397
- wxSashLayoutWindow::SetDefaultSize, 1397
- wxSashLayoutWindow::SetOrientation, 1397
- wxSashLayoutWindow::wxSashLayoutWindow, 1395
- wxSashWindow, 1398, 1399
- wxSashWindow::~wxSashWindow, 1399
- wxSashWindow::GetMaximumSizeX, 1400
- wxSashWindow::GetMaximumSizeY, 1400
- wxSashWindow::GetMinimumSizeX, 1400
- wxSashWindow::GetMinimumSizeY, 1400
- wxSashWindow::GetSashVisible, 1399
- wxSashWindow::HasBorder, 1400
- wxSashWindow::SetMaximumSizeX, 1400
- wxSashWindow::SetMaximumSizeY, 1401
- wxSashWindow::SetMinimumSizeX, 1401
- wxSashWindow::SetMinimumSizeY, 1401
- wxSashWindow::SetSashBorder, 1401
- wxSashWindow::SetSashVisible, 1401
- wxSashWindow::wxSashWindow, 1398
- wxSB\_FLAT, 1541
- wxSB\_HORIZONTAL, 1409
- wxSB\_NORMAL, 1541
- wxSB\_RAISED, 1541
- wxSB\_VERTICAL, 1409
- wxScopedArray, 1403
- wxScopedArray::get, 1403
- wxScopedArray::operator [], 1403
- wxScopedArray::reset, 1403
- wxScopedArray::swap, 1403
- wxScopedArray::wxScopedArray, 1403
- wxScopedPtr::~wxScopedPtr, 1405
- wxScopedPtr::get, 1406
- wxScopedPtr::operator \*, 1406
- wxScopedPtr::operator -, 1406
- wxScopedPtr::release, 1405
- wxScopedPtr::reset, 1405
- wxScopedPtr::swap, 1406
- wxScopedPtr::wxScopedPtr, 1405
- wxScopedTiedPtr, 1406
- wxScopedTiedPtr::~wxScopedTiedPtr, 1407
- wxScopedTiedPtr::wxScopedTiedPtr, 1406
- wxScreenDC, 1407
- wxScreenDC::EndDrawingOnTop, 1408
- wxScreenDC::StartDrawingOnTop, 1407
- wxScreenDC::wxScreenDC, 1407
- wxScrollBar, 1411
- wxScrollBar::~wxScrollBar, 1412
- wxScrollBar::Create, 1412
- wxScrollBar::GetPageSize, 1412
- wxScrollBar::GetRange, 1412
- wxScrollBar::GetThumbPosition, 1412
- wxScrollBar::GetThumbSize, 1413
- wxScrollBar::SetScrollbar, 1413
- wxScrollBar::SetThumbPosition, 1413
- wxScrollBar::wxScrollBar, 1411
- wxScrolledWindow, 1416
- wxScrolledWindow::~wxScrolledWindow, 1417
- wxScrolledWindow::CalcScrolledPosition, 1417
- wxScrolledWindow::CalcUnscrolledPosition, 1417
- wxScrolledWindow::Create, 1418
- wxScrolledWindow::DoPrepareDC, 1420
- wxScrolledWindow::EnableScrolling, 1418
- wxScrolledWindow::GetScrollPixelsPerUnit, 1418
- wxScrolledWindow::GetViewStart, 1419
- wxScrolledWindow::GetVirtualSize, 1419
- wxScrolledWindow::IsRetained, 1420
- wxScrolledWindow::OnDraw, 1421
- wxScrolledWindow::PrepareDC, 1421
- wxScrolledWindow::Scroll, 1421

- wxScrolledWindow::SetScrollbars, 1421
- wxScrolledWindow::SetScrollRate, 1423
- wxScrolledWindow::SetTargetWindow, 1423
- wxScrolledWindow::wxScrolledWindow, 1416
- wxScrollEvent, 1425
- wxScrollEvent::GetOrientation, 1425
- wxScrollEvent::GetPosition, 1426
- wxScrollEvent::wxScrollEvent, 1425
- wxScrollWinEvent, 1427
- wxScrollWinEvent::GetOrientation, 1427
- wxScrollWinEvent::GetPosition, 1427
- wxScrollWinEvent::wxScrollWinEvent, 1427
- wxSearchCtrl, 1433, 1434
- wxSearchCtrl::~~wxSearchCtrl, 1434
- wxSearchCtrl::GetMenu, 1435
- wxSearchCtrl::IsCancelButtonVisible, 1435
- wxSearchCtrl::IsSearchButtonVisible, 1435
- wxSearchCtrl::SetMenu, 1434
- wxSearchCtrl::ShowCancelButton, 1435
- wxSearchCtrl::ShowSearchButton, 1435
- wxSearchCtrl::wxSearchCtrl, 1433
- wxSemaphore, 1428
- wxSemaphore::~~wxSemaphore, 1428
- wxSemaphore::Post, 1428
- wxSemaphore::TryWait, 1428
- wxSemaphore::Wait, 1429
- wxSemaphore::WaitTimeout, 1429
- wxSemaphore::wxSemaphore, 1428
- wxServer, 1431
- wxServer::Create, 1431
- wxServer::OnAcceptConnection, 1431
- wxServer::wxServer, 1431
- wxSetClipboardData, 1951
- wxSetCursor, 1946
- wxSetCursorEvent, 1430
- wxSetCursorEvent::GetCursor, 1430
- wxSetCursorEvent::GetX, 1430
- wxSetCursorEvent::GetY, 1430
- wxSetCursorEvent::HasCursor, 1430
- wxSetCursorEvent::SetCursor, 1430
- wxSetCursorEvent::wxSetCursorEvent, 1430
- wxSetDisplayName, 1960
- wxSetEnv, 1984
- wxSetPrinterCommand, 1948
- wxSetPrinterFile, 1948
- wxSetPrinterMode, 1948
- wxSetPrinterOptions, 1948
- wxSetPrinterOrientation, 1948
- wxSetPrinterPreviewCommand, 1948
- wxSetPrinterScaling, 1949
- wxSetPrinterTranslation, 1949
- wxSetWorkingDirectory, 1924
- wxSHAPED, 1447
- wxShell, 1916
- wxShowTip, 1943
- wxShutdown, 1916
- wxSimpleHtmlListBox, 858
- wxSimpleHtmlListBox::~~wxSimpleHtmlListBox, 858
- wxSimpleHtmlListBox::Create, 858
- wxSimpleHtmlListBox::wxSimpleHtmlListBox, 857
- wxSingleChoiceDialog, 1436
- wxSingleChoiceDialog overview, 2131
- wxSingleChoiceDialog::GetSelection, 1437
- wxSingleChoiceDialog::GetSelectionClientData, 1437
- wxSingleChoiceDialog::GetStringSelection, 1437
- wxSingleChoiceDialog::SetSelection, 1437
- wxSingleChoiceDialog::ShowModal, 1437
- wxSingleChoiceDialog::wxSingleChoiceDialog, 1436
- wxSingleInstanceChecker, 1439
- wxSingleInstanceChecker::~~wxSingleInstanceChecker, 1440
- wxSingleInstanceChecker::Create, 1439
- wxSingleInstanceChecker::IsAnotherRunning, 1439
- wxSingleInstanceChecker::wxSingleInstanceChecker, 1439
- wxSize, 1440
- wxSize::DecBy, 1440
- wxSize::DecTo, 1441
- wxSize::GetHeight, 1441
- wxSize::GetWidth, 1441
- wxSize::IncBy, 1441
- wxSize::IncTo, 1442
- wxSize::IsFullySpecified, 1441
- wxSize::Scale, 1442
- wxSize::Set, 1442
- wxSize::SetDefaults, 1442
- wxSize::SetHeight, 1442
- wxSize::SetWidth, 1443
- wxSize::wxSize, 1440
- wxSizeEvent, 1444
- wxSizeEvent::GetSize, 1444
- wxSizeEvent::wxSizeEvent, 1444
- wxSizer, 1445
- wxSizer::~~wxSizer, 1445
- wxSizer::Add, 1445
- wxSizer::AddSpacer, 1448
- wxSizer::AddStretchSpacer, 1448
- wxSizer::CalcMin, 1448
- wxSizer::Clear, 1448
- wxSizer::ComputeFittingClientSize, 1448
- wxSizer::ComputeFittingWindowSize, 1449
- wxSizer::Detach, 1449
- wxSizer::Fit, 1449
- wxSizer::FitInside, 1449
- wxSizer::GetChildren, 1450
- wxSizer::GetContainingWindow, 1450
- wxSizer::GetItem, 1450
- wxSizer::GetMinSize, 1450
- wxSizer::GetPosition, 1450
- wxSizer::GetSize, 1450
- wxSizer::Hide, 1451
- wxSizer::Insert, 1451
- wxSizer::InsertSpacer, 1451
- wxSizer::InsertStretchSpacer, 1452
- wxSizer::IsShown, 1452
- wxSizer::Layout, 1452
- wxSizer::Prepend, 1452
- wxSizer::PrependSpacer, 1452
- wxSizer::PrependStretchSpacer, 1453
- wxSizer::RecalcSizes, 1453



---

wxSizer::Remove, 1453  
wxSizer::Replace, 1453  
wxSizer::SetDimension, 1454  
wxSizer::SetItemMinSize, 1454  
wxSizer::SetMinSize, 1454  
wxSizer::SetSizeHints, 1454  
wxSizer::SetVirtualSizeHints, 1454  
wxSizer::Show, 1455  
wxSizer::wxSizer, 1445  
wxSizerFlags, 1456  
wxSizerFlags::Align, 1456  
wxSizerFlags::Border, 1456  
wxSizerFlags::Center, 1456  
wxSizerFlags::Centre, 1456  
wxSizerFlags::DoubleBorder, 1456  
wxSizerFlags::DoubleHorzBorder, 1456  
wxSizerFlags::Expand, 1457  
wxSizerFlags::FixedMinSize, 1457  
wxSizerFlags::GetDefaultBorder, 1457  
wxSizerFlags::Left, 1457  
wxSizerFlags::Proportion, 1457  
wxSizerFlags::ReserveSpaceEvenIfHidden, 1457  
wxSizerFlags::Right, 1457  
wxSizerFlags::Shaped, 1458  
wxSizerFlags::TripleBorder, 1458  
wxSizerFlags::wxSizerFlags, 1455  
wxSizerItem, 1458, 1459  
wxSizerItem::~~wxSizerItem, 1459  
wxSizerItem::CalcMin, 1459  
wxSizerItem::DeleteWindows, 1459  
wxSizerItem::DetachSizer, 1459  
wxSizerItem::GetBorder, 1459  
wxSizerItem::GetFlag, 1459  
wxSizerItem::GetMinSize, 1459  
wxSizerItem::GetPosition, 1459  
wxSizerItem::GetProportion, 1460  
wxSizerItem::GetRatio, 1460  
wxSizerItem::GetRect, 1460  
wxSizerItem::GetSize, 1460  
wxSizerItem::GetSizer, 1460  
wxSizerItem::GetSpacer, 1460  
wxSizerItem::GetUserData, 1460  
wxSizerItem::GetWindow, 1460  
wxSizerItem::IsShown, 1461  
wxSizerItem::IsSizer, 1461  
wxSizerItem::IsSpacer, 1461  
wxSizerItem::IsWindow, 1461  
wxSizerItem::SetBorder, 1461  
wxSizerItem::SetDimension, 1461  
wxSizerItem::SetFlag, 1461  
wxSizerItem::SetInitSize, 1461  
wxSizerItem::SetProportion, 1461  
wxSizerItem::SetRatio, 1462  
wxSizerItem::SetSizer, 1462  
wxSizerItem::SetSpacer, 1462  
wxSizerItem::SetWindow, 1462  
wxSizerItem::Show, 1462  
wxSizerItem::wxSizerItem, 1458  
wxSL\_AUTOTICKS, 1463  
wxSL\_BOTTOM, 1463  
wxSL\_HORIZONTAL, 1463  
wxSL\_INVERSE, 1463  
wxSL\_LABELS, 1463  
wxSL\_LEFT, 1463  
wxSL\_RIGHT, 1463  
wxSL\_SELRange, 1463  
wxSL\_TOP, 1463  
wxSL\_VERTICAL, 1463  
wxSleep, 1979  
wxSlider, 1465  
wxSlider::~~wxSlider, 1466  
wxSlider::ClearSel, 1466  
wxSlider::ClearTicks, 1466  
wxSlider::Create, 1467  
wxSlider::GetLineSize, 1467  
wxSlider::GetMax, 1467  
wxSlider::GetMin, 1467  
wxSlider::GetPageSize, 1467  
wxSlider::GetSelEnd, 1468  
wxSlider::GetSelStart, 1468  
wxSlider::GetThumbLength, 1468  
wxSlider::GetTickFreq, 1468  
wxSlider::GetValue, 1469  
wxSlider::SetLineSize, 1469  
wxSlider::SetPageSize, 1469  
wxSlider::SetRange, 1469  
wxSlider::SetSelection, 1470  
wxSlider::SetThumbLength, 1470  
wxSlider::SetTick, 1470  
wxSlider::SetTickFreq, 1471  
wxSlider::SetValue, 1471  
wxSlider::wxSlider, 1465  
wxSnprintf, 1932  
wxSocketAddress, 1472  
wxSocketAddress::~~wxSocketAddress, 1472  
wxSocketAddress::Clear, 1472  
wxSocketAddress::wxSocketAddress, 1472  
wxSocketBase, 1475  
wxSocketBase::~~wxSocketBase, 1475  
wxSocketBase::Close, 1475  
wxSocketBase::Destroy, 1476  
wxSocketBase::Discard, 1476  
wxSocketBase::Error, 1476  
wxSocketBase::GetClientData, 1476  
wxSocketBase::GetFlags, 1477  
wxSocketBase::GetLocal, 1477  
wxSocketBase::GetPeer, 1477  
wxSocketBase::InterruptWait, 1477  
wxSocketBase::IsConnected, 1477  
wxSocketBase::IsData, 1478  
wxSocketBase::IsDisconnected, 1478  
wxSocketBase::IsOk, 1478  
wxSocketBase::LastCount, 1478  
wxSocketBase::LastError, 1478  
wxSocketBase::Notify, 1478  
wxSocketBase::Peek, 1482  
wxSocketBase::Read, 1482  
wxSocketBase::ReadMsg, 1483  
wxSocketBase::RestoreState, 1479  
wxSocketBase::SaveState, 1479  
wxSocketBase::SetClientData, 1479  
wxSocketBase::SetEventHandler, 1479  
wxSocketBase::SetFlags, 1480  
wxSocketBase::SetLocal, 1481

---

---

wxSocketBase::SetNotify, 1481  
wxSocketBase::SetTimeout, 1482  
wxSocketBase::Unread, 1484  
wxSocketBase::Wait, 1484  
wxSocketBase::WaitForLost, 1485  
wxSocketBase::WaitForRead, 1486  
wxSocketBase::WaitForWrite, 1486  
wxSocketBase::Write, 1487  
wxSocketBase::WriteMsg, 1487  
wxSocketBase::wxSocketBase, 1475  
wxSocketClient, 1488  
wxSocketClient::~wxSocketClient, 1488  
wxSocketClient::Connect, 1489  
wxSocketClient::WaitOnConnect, 1489  
wxSocketClient::wxSocketClient, 1488  
wxSocketEvent, 1491  
wxSocketEvent::GetClientData, 1491  
wxSocketEvent::GetSocket, 1491  
wxSocketEvent::GetSocketEvent, 1491  
wxSocketEvent::wxSocketEvent, 1491  
wxSocketInputStream, 1492  
wxSocketInputStream::wxSocketInputStream, 1492  
wxSocketOutputStream, 1492  
wxSocketOutputStream::wxSocketOutputStream, 1492  
wxSocketServer, 1492  
wxSocketServer::~wxSocketServer, 1493  
wxSocketServer::Accept, 1493  
wxSocketServer::AcceptWith, 1493  
wxSocketServer::WaitForAccept, 1494  
wxSocketServer::wxSocketServer, 1492  
wxSortedArray, 78  
wxSound, 1495  
wxSound::~wxSound, 1495  
wxSound::Create, 1495  
wxSound::IsOk, 1495  
wxSound::IsPlaying, 1496  
wxSound::Play, 1496  
wxSound::Stop, 1496  
wxSound::wxSound, 1495  
wxSP\_3D, 1508  
wxSP\_3DBORDER, 1508  
wxSP\_3DSASH, 1508  
wxSP\_ARROW\_KEYS, 1497, 1500  
wxSP\_BORDER, 1508  
wxSP\_HORIZONTAL, 1497  
wxSP\_LIVE\_UPDATE, 1509  
wxSP\_NO\_XP\_THEME, 1508  
wxSP\_NOBORDER, 1508  
wxSP\_PERMIT\_UNSPPLIT, 1508  
wxSP\_VERTICAL, 1497  
wxSP\_WRAP, 1497, 1500  
wxSpinButton, 1498  
wxSpinButton::~wxSpinButton, 1498  
wxSpinButton::Create, 1498  
wxSpinButton::GetMax, 1499  
wxSpinButton::GetMin, 1499  
wxSpinButton::GetValue, 1499  
wxSpinButton::SetRange, 1499  
wxSpinButton::SetValue, 1500  
wxSpinButton::wxSpinButton, 1498  
wxSpinCtrl, 1501  
wxSpinCtrl::Create, 1502  
wxSpinCtrl::GetMax, 1503  
wxSpinCtrl::GetMin, 1502  
wxSpinCtrl::GetValue, 1502  
wxSpinCtrl::SetRange, 1502  
wxSpinCtrl::SetSelection, 1502  
wxSpinCtrl::SetValue, 1502  
wxSpinCtrl::wxSpinCtrl, 1501  
wxSpinEvent, 1503  
wxSpinEvent::GetPosition, 1503  
wxSpinEvent::SetPosition, 1504  
wxSpinEvent::wxSpinEvent, 1503  
wxSplashScreen, 1504  
wxSplashScreen::~wxSplashScreen, 1505  
wxSplashScreen::GetSplashStyle, 1505  
wxSplashScreen::GetSplashWindow, 1505  
wxSplashScreen::GetTimeout, 1505  
wxSplashScreen::OnCloseWindow, 1505  
wxSplashScreen::wxSplashScreen, 1504  
wxSplitPath, 1925  
wxSplitterEvent, 1507  
wxSplitterEvent::GetSashPosition, 1507  
wxSplitterEvent::GetWindowBeingRemoved, 1507  
wxSplitterEvent::GetX, 1507  
wxSplitterEvent::GetY, 1507  
wxSplitterEvent::SetSashPosition, 1508  
wxSplitterEvent::wxSplitterEvent, 1507  
wxSplitterRenderParams::border, 1519  
wxSplitterRenderParams::isHotSensitive, 1519  
wxSplitterWindow, 1510  
wxSplitterWindow::~wxSplitterWindow, 1511  
wxSplitterWindow::Create, 1511  
wxSplitterWindow::GetMinimumPaneSize, 1511  
wxSplitterWindow::GetSashGravity, 1511  
wxSplitterWindow::GetSashPosition, 1511  
wxSplitterWindow::GetSplitMode, 1512  
wxSplitterWindow::GetWindow1, 1512  
wxSplitterWindow::GetWindow2, 1512  
wxSplitterWindow::Initialize, 1512  
wxSplitterWindow::IsSplit, 1512  
wxSplitterWindow::OnDoubleClickSash, 1513  
wxSplitterWindow::OnSashPositionChange, 1513  
wxSplitterWindow::OnUnsplit, 1513  
wxSplitterWindow::ReplaceWindow, 1514  
wxSplitterWindow::SetMinimumPaneSize, 1515  
wxSplitterWindow::SetSashGravity, 1514  
wxSplitterWindow::SetSashPosition, 1515  
wxSplitterWindow::SetSashSize, 1515  
wxSplitterWindow::SetSplitMode, 1516  
wxSplitterWindow::SplitHorizontally, 1516  
wxSplitterWindow::SplitVertically, 1517  
wxSplitterWindow::Unsplit, 1518  
wxSplitterWindow::UpdateSize, 1518  
wxSplitterWindow::wxSplitterWindow, 1510  
wxST\_NO\_AUTORESIZE, 1535  
wxST\_SIZEGRIP, 1537  
wxStackFrame::GetAddress, 1519  
wxStackFrame::GetFileName, 1519  
wxStackFrame::GetLevel, 1520  
wxStackFrame::GetLine, 1520

- wxStackFrame::GetModule, 1520
- wxStackFrame::GetName, 1520
- wxStackFrame::GetOffset, 1520
- wxStackFrame::GetParam, 1520
- wxStackFrame::GetParamCount, 1521
- wxStackFrame::HasSourceLocation, 1521
- wxStackWalker, 1522
- wxStackWalker::~wxStackWalker, 1522
- wxStackWalker::OnStackFrame, 1522
- wxStackWalker::Walk, 1522
- wxStackWalker::WalkFromException, 1522
- wxStackWalker::wxStackWalker, 1522
- wxStandardPaths::Get, 1523
- wxStandardPaths::GetConfigDir, 1523
- wxStandardPaths::GetDataDir, 1523
- wxStandardPaths::GetDocumentsDir, 1524
- wxStandardPaths::GetExecutablePath, 1524
- wxStandardPaths::GetInstallPrefix, 1524
- wxStandardPaths::GetLocalDataDir, 1525
- wxStandardPaths::GetLocalizedResourcesDir, 1525
- wxStandardPaths::GetPluginsDir, 1525
- wxStandardPaths::GetResourcesDir, 1525
- wxStandardPaths::GetTempDir, 1526
- wxStandardPaths::GetUserConfigDir, 1526
- wxStandardPaths::GetUserDataDir, 1526
- wxStandardPaths::GetUserLocalDataDir, 1527
- wxStandardPaths::SetInstallPrefix, 1527
- wxStartTimer, 1979
- wxStaticBitmap, 1528
- wxStaticBitmap::Create, 1529
- wxStaticBitmap::GetBitmap, 1529
- wxStaticBitmap::GetIcon, 1529
- wxStaticBitmap::SetBitmap, 1529
- wxStaticBitmap::SetIcon, 1529
- wxStaticBitmap::wxStaticBitmap, 1528
- wxStaticBox, 1530, 1531
- wxStaticBox::~wxStaticBox, 1531
- wxStaticBox::Create, 1531
- wxStaticBox::wxStaticBox, 1530
- wxStaticBoxSizer, 1532
- wxStaticBoxSizer::GetStaticBox, 1532
- wxStaticBoxSizer::wxStaticBoxSizer, 1532
- wxStaticCast, 1970
- wxStaticLine, 1533
- wxStaticLine::Create, 1534
- wxStaticLine::GetDefaultSize, 1534
- wxStaticLine::IsVertical, 1534
- wxStaticLine::wxStaticLine, 1533
- wxStaticText, 1535
- wxStaticText::Create, 1536
- wxStaticText::GetLabel, 1536
- wxStaticText::SetLabel, 1536
- wxStaticText::Wrap, 1536
- wxStaticText::wxStaticText, 1535
- wxStatusBar, 1537
- wxStatusBar::~wxStatusBar, 1538
- wxStatusBar::Create, 1538
- wxStatusBar::GetFieldRect, 1538
- wxStatusBar::GetFieldsCount, 1538
- wxStatusBar::GetStatusText, 1539
- wxStatusBar::PopStatusText, 1539
- wxStatusBar::PushStatusText, 1539
- wxStatusBar::SetFieldsCount, 1539
- wxStatusBar::SetMinHeight, 1540
- wxStatusBar::SetStatusStyles, 1541
- wxStatusBar::SetStatusText, 1540
- wxStatusBar::SetStatusWidths, 1540
- wxStatusBar::wxStatusBar, 1537
- wxSTAY\_ON\_TOP, 497, 682, 1047, 1052, 1114
- wxStdDialogButtonSizer, 1542
- wxStdDialogButtonSizer::AddButton, 1542
- wxStdDialogButtonSizer::Realize, 1543
- wxStdDialogButtonSizer::SetAffirmativeButton, 1543
- wxStdDialogButtonSizer::SetCancelButton, 1543
- wxStdDialogButtonSizer::SetNegativeButton, 1543
- wxStdDialogButtonSizer::wxStdDialogButtonSizer, 1542
- wxStopWatch, 1544
- wxStopWatch::Pause, 1544
- wxStopWatch::Resume, 1544
- wxStopWatch::Start, 1544
- wxStopWatch::Time, 1544
- wxStopWatch::wxStopWatch, 1544
- wxStrcmp, 1930
- wxStreamBase, 1545
- wxStreamBase::~wxStreamBase, 1545
- wxStreamBase::GetLastError, 1545
- wxStreamBase::GetLength, 1545
- wxStreamBase::GetSize, 1545
- wxStreamBase::IsOk, 1546
- wxStreamBase::IsSeekable, 1546
- wxStreamBase::OnSysRead, 1546
- wxStreamBase::OnSysSeek, 1546
- wxStreamBase::OnSysTell, 1546
- wxStreamBase::OnSysWrite, 1546
- wxStreamBase::wxStreamBase, 1545
- wxStreamBuffer, 1547, 1548
- wxStreamBuffer::~wxStreamBuffer, 1548
- wxStreamBuffer::FillBuffer, 1552
- wxStreamBuffer::Fixed, 1551
- wxStreamBuffer::Flushable, 1551
- wxStreamBuffer::FlushBuffer, 1551
- wxStreamBuffer::GetBufferEnd, 1550
- wxStreamBuffer::GetBufferPos, 1551
- wxStreamBuffer::GetBufferStart, 1550
- wxStreamBuffer::GetChar, 1548
- wxStreamBuffer::GetDataLeft, 1552
- wxStreamBuffer::GetIntPosition, 1551
- wxStreamBuffer::GetLastAccess, 1551
- wxStreamBuffer::PutChar, 1549
- wxStreamBuffer::Read, 1548
- wxStreamBuffer::ResetBuffer, 1549
- wxStreamBuffer::Seek, 1549
- wxStreamBuffer::SetBufferIO, 1550
- wxStreamBuffer::SetIntPosition, 1551
- wxStreamBuffer::Stream, 1552
- wxStreamBuffer::Tell, 1549
- wxStreamBuffer::Write, 1548
- wxStreamBuffer::wxStreamBuffer, 1547
- wxStreamToTextRedirector, 1553
- wxStreamToTextRedirector::~wxStreamToTextR

- edirector, 1553
- wxStreamToTextRedirector::wxStreamToTextRe  
director, 1553
- wxStricmp, 1931
- wxString, 1560, 1561
- wxString::~~wxString, 1561
- wxString::AfterFirst, 1562
- wxString::AfterLast, 1562
- wxString::Alloc, 1561
- wxString::Append, 1562
- wxString::BeforeFirst, 1562
- wxString::BeforeLast, 1563
- wxString::c\_str, 1563
- wxString::char\_str, 1563
- wxString::Clear, 1563
- wxString::Cmp, 1564
- wxString::CmpNoCase, 1564
- wxString::CompareTo, 1564
- wxString::Contains, 1564
- wxString::Empty, 1564
- wxString::EndsWith, 1572
- wxString::Find, 1565
- wxString::First, 1565
- wxString::fn\_str, 1565
- wxString::Format, 1565
- wxString::FormatV, 1566
- wxString::Freq, 1566
- wxString::From8BitData, 1566
- wxString::FromAscii, 1566
- wxString::FromUTF8, 1566
- wxString::GetChar, 1567
- wxString::GetData, 1567
- wxString::GetWritableChar, 1567
- wxString::GetWriteBuf, 1567
- wxString::Index, 1567
- wxString::IsAscii, 1567
- wxString::IsEmpty, 1568
- wxString::IsNull, 1568
- wxString::IsNumber, 1568
- wxString::IsSameAs, 1568
- wxString::IsWord, 1568
- wxString::Last, 1569
- wxString::Left, 1569
- wxString::Len, 1569
- wxString::Length, 1569
- wxString::Lower, 1569
- wxString::LowerCase, 1569
- wxString::MakeLower, 1569
- wxString::MakeUpper, 1570
- wxString::Matches, 1570
- wxString::mb\_str, 1570
- wxString::Mid, 1570
- wxString::operator (), 1578
- wxString::operator [], 1577
- wxString::operator +, 1577
- wxString::operator +=, 1577
- wxString::operator <<, 1578
- wxString::operator =, 1577
- wxString::operator >>, 1578
- wxString::operator const wxChar\*, 1578
- wxString::operator!, 1576
- wxString::Pad, 1570
- wxString::Prepend, 1570
- wxString::Printf, 1570
- wxString::PrintfV, 1571
- wxString::Remove, 1571
- wxString::RemoveLast, 1571
- wxString::Replace, 1571
- wxString::Right, 1572
- wxString::SetChar, 1572
- wxString::Shrink, 1572
- wxString::StartsWith, 1572
- wxString::Strip, 1572
- wxString::SubString, 1573
- wxString::To8BitData, 1573
- wxString::ToAscii, 1573
- wxString::ToDouble, 1573
- wxString::ToLong, 1574
- wxString::ToLongLong, 1574
- wxString::ToULong, 1574
- wxString::ToULongLong, 1575
- wxString::ToUTF8, 1575
- wxString::Trim, 1575
- wxString::Truncate, 1575
- wxString::UngetWriteBuf, 1575
- wxString::Upper, 1575
- wxString::UpperCase, 1576
- wxString::utf8\_str, 1576
- wxString::wc\_str, 1576
- wxString::wchar\_str, 1576
- wxString::wxString, 1560
- wxStringBuffer, 1579
- wxStringBuffer::~~wxStringBuffer, 1580
- wxStringBuffer::operator wxChar \*, 1580
- wxStringBuffer::wxStringBuffer, 1579
- wxStringBufferLength, 1581
- wxStringBufferLength::~~wxStringBufferLength,  
1581
- wxStringBufferLength::operator wxChar \*, 1581
- wxStringBufferLength::SetLength, 1581
- wxStringBufferLength::wxStringBufferLength,  
1581
- wxStringClientData, 1582
- wxStringClientData::GetData, 1582
- wxStringClientData::SetData, 1582
- wxStringClientData::wxStringClientData, 1581
- wxStringEq, 1931
- wxStringInputStream, 1582
- wxStringInputStream::wxStringInputStream, 1582
- wxSTRINGIZE, 1961
- wxSTRINGIZE\_T, 1961
- wxStringMatch, 1931
- wxStringOutputStream, 1583
- wxStringOutputStream::GetString, 1583
- wxStringOutputStream::wxStringOutputStream,  
1583
- wxStringTokenize, 1931
- wxStringTokenizer, 1584, 1585
- wxStringTokenizer::CountTokens, 1585
- wxStringTokenizer::GetLastDelimiter, 1585
- wxStringTokenizer::GetNextToken, 1585
- wxStringTokenizer::GetPosition, 1585
- wxStringTokenizer::GetString, 1585
- wxStringTokenizer::HasMoreTokens, 1585

- wxStringTokenizer::SetString, 1585
- wxStringTokenizer::wxStringTokenizer, 1584
- wxStripMenuCodes, 1961
- wxStrlen, 1931
- wxSUPPRESS\_GCC\_PRIVATE\_DTOR\_WARNING, 1961
- wxSW\_3D, 1398
- wxSW\_3DBORDER, 1398
- wxSW\_3DSASH, 1398
- wxSW\_BORDER, 1398
- wxSymbolPickerDialog, 1587
- wxSymbolPickerDialog::Create, 1588
- wxSymbolPickerDialog::GetFontName, 1588
- wxSymbolPickerDialog::GetFromUnicode, 1588
- wxSymbolPickerDialog::GetNormalTextFontName, 1588
- wxSymbolPickerDialog::GetSymbol, 1588
- wxSymbolPickerDialog::GetSymbolChar, 1589
- wxSymbolPickerDialog::HasSelection, 1589
- wxSymbolPickerDialog::SetFontName, 1589
- wxSymbolPickerDialog::SetFromUnicode, 1589
- wxSymbolPickerDialog::SetNormalTextFontName, 1589
- wxSymbolPickerDialog::SetSymbol, 1589
- wxSymbolPickerDialog::SetUnicodeMode, 1589
- wxSymbolPickerDialog::UseNormalFont, 1589
- wxSymbolPickerDialog::wxSymbolPickerDialog, 1587
- wxSysColourChangedEvent, 1590
- wxSysColourChangedEvent::wxSysColourChangedEvent, 1590
- wxSysErrorCode, 1975
- wxSysErrorMsg, 1976
- wxSYSTEM\_MENU, 497, 683, 1047, 1052, 1114
- wxSystemOptions, 1593
- wxSystemOptions::GetOption, 1593
- wxSystemOptions::GetOptionInt, 1593
- wxSystemOptions::HasOption, 1593
- wxSystemOptions::IsFalse, 1593
- wxSystemOptions::SetOption, 1594
- wxSystemOptions::wxSystemOptions, 1593
- wxSystemSettings, 1594
- wxSystemSettings::GetColour, 1594
- wxSystemSettings::GetFont, 1596
- wxSystemSettings::GetMetric, 1596
- wxSystemSettings::GetScreenType, 1598
- wxSystemSettings::wxSystemSettings, 1594
- wxT, 1932
- wxTAB\_TRAVERSAL, 1796
- wxTarEntry, 1600
- wxTarEntry::Get/SetAccessTime, 1600
- wxTarEntry::Get/SetCreateTime, 1600
- wxTarEntry::Get/SetDevMajor and Get/SetDevMinor, 1600
- wxTarEntry::Get/SetGroupId and Get/SetUserId, 1601
- wxTarEntry::Get/SetGroupName and Get/SetUserName, 1601
- wxTarEntry::Get/SetLinkName, 1601
- wxTarEntry::Get/SetMode, 1602
- wxTarEntry::Get/SetSize, 1602
- wxTarEntry::Get/SetTypeFlag, 1602
- wxTarEntry::GetInternalName, 1601
- wxTarEntry::operator=, 1603
- wxTarEntry::wxTarEntry, 1600
- wxTarInputStream, 1603, 1604
- wxTarInputStream::CloseEntry, 1604
- wxTarInputStream::GetNextEntry, 1604
- wxTarInputStream::OpenEntry, 1604
- wxTarInputStream::wxTarInputStream, 1603
- wxTarOutputStream, 1605
- wxTarOutputStream::~wxTarOutputStream, 1605
- wxTarOutputStream::Close, 1605
- wxTarOutputStream::CloseEntry, 1606
- wxTarOutputStream::CopyArchiveMetaData, 1606
- wxTarOutputStream::CopyEntry, 1606
- wxTarOutputStream::Get/SetBlockingFactor, 1606
- wxTarOutputStream::PutNextDirEntry, 1606
- wxTarOutputStream::PutNextEntry, 1607
- wxTarOutputStream::wxTarOutputStream, 1605
- wxTaskBarIcon, 1608
- wxTaskBarIcon::~wxTaskBarIcon, 1608
- wxTaskBarIcon::CreatePopupMenu, 1608
- wxTaskBarIcon::IsIconInstalled, 1608
- wxTaskBarIcon::IsOk, 1609
- wxTaskBarIcon::PopupMenu, 1609
- wxTaskBarIcon::RemoveIcon, 1609
- wxTaskBarIcon::SetIcon, 1609
- wxTaskBarIcon::wxTaskBarIcon, 1608
- wxBOTTOM, 1696
- wxBOTTOM\_DOCKABLE, 1695
- wxBOTTOM\_FLAT, 1695
- wxBOTTOM\_HORIZONTAL, 1696
- wxBOTTOM\_HORZ\_LAYOUT, 1696
- wxBOTTOM\_HORZ\_TEXT, 1696
- wxBOTTOM\_NO\_TOOLTIPS, 1696
- wxBOTTOM\_NOALIGN, 1696
- wxBOTTOM\_NODIVIDER, 1696
- wxBOTTOM\_NOICONS, 1696
- wxBOTTOM\_RIGHT, 1696
- wxBOTTOM\_TEXT, 1696
- wxBOTTOM\_VERTICAL, 1696
- wxTCPClient, 1610
- wxTCPClient::MakeConnection, 1610
- wxTCPClient::OnMakeConnection, 1610
- wxTCPClient::ValidHost, 1610
- wxTCPClient::wxTCPClient, 1610
- wxTCPConnection, 1612
- wxTCPConnection::Advise, 1612
- wxTCPConnection::Disconnect, 1612
- wxTCPConnection::Execute, 1612
- wxTCPConnection::OnAdvise, 1612
- wxTCPConnection::OnDisconnect, 1613
- wxTCPConnection::OnExecute, 1613
- wxTCPConnection::OnPoke, 1613
- wxTCPConnection::OnRequest, 1613
- wxTCPConnection::OnStartAdvise, 1613
- wxTCPConnection::OnStopAdvise, 1613
- wxTCPConnection::Poke, 1613
- wxTCPConnection::Request, 1614
- wxTCPConnection::StartAdvise, 1614
- wxTCPConnection::StopAdvise, 1614

wxTCPConnection::wxTCPConnection, 1611  
wxTCPServer, 1615  
wxTCPServer::Create, 1615  
wxTCPServer::OnAcceptConnection, 1615  
wxTCPServer::wxTCPServer, 1615  
wxTE\_AUTO\_URL, 1634  
wxTE\_BESTWRAP, 1635  
wxTE\_CAPITALIZE, 1433, 1635  
wxTE\_CENTRE, 1433, 1634, 1635  
wxTE\_CHARWRAP, 1635  
wxTE\_DONTWRAP, 1635  
wxTE\_LEFT, 1433, 1634, 1635  
wxTE\_MULTILINE, 1634  
wxTE\_NOHIDESEL, 1433, 1634  
wxTE\_PASSWORD, 1173, 1634, 1635, 1656  
wxTE\_PROCESS\_ENTER, 136, 226, 234, 1432, 1634  
wxTE\_PROCESS\_TAB, 1433, 1634  
wxTE\_READONLY, 1634, 1635  
wxTE\_RICH, 1634  
wxTE\_RICH2, 1634  
wxTE\_RIGHT, 1433, 1635  
wxTE\_WORDWRAP, 1635  
wxTempFile, 1616  
wxTempFile::~wxTempFile, 1617  
wxTempFile::Commit, 1617  
wxTempFile::Discard, 1617  
wxTempFile::IsOpened, 1616  
wxTempFile::Length, 1616  
wxTempFile::Open, 1616  
wxTempFile::Seek, 1617  
wxTempFile::Tell, 1617  
wxTempFile::Write, 1617  
wxTempFile::wxTempFile, 1616  
wxTempFileOutputStream, 1618  
wxTempFileOutputStream::Commit, 1618  
wxTempFileOutputStream::Discard, 1618  
wxTempFileOutputStream::wxTempFileOutputSteam, 1618  
wxTextAttr, 1620  
wxTextAttr::GetAlignment, 1620  
wxTextAttr::GetBackgroundColour, 1620  
wxTextAttr::GetFlags, 1622  
wxTextAttr::GetFont, 1620  
wxTextAttr::GetLeftIndent, 1620  
wxTextAttr::GetLeftSubIndent, 1620  
wxTextAttr::GetRightIndent, 1620  
wxTextAttr::GetTabs, 1621  
wxTextAttr::GetTextColour, 1621  
wxTextAttr::HasAlignment, 1621  
wxTextAttr::HasBackgroundColour, 1621  
wxTextAttr::HasFont, 1621  
wxTextAttr::HasLeftIndent, 1621  
wxTextAttr::HasRightIndent, 1621  
wxTextAttr::HasTabs, 1621  
wxTextAttr::HasTextColour, 1622  
wxTextAttr::IsDefault, 1622  
wxTextAttr::Merge, 1622  
wxTextAttr::SetAlignment, 1622  
wxTextAttr::SetBackgroundColour, 1622  
wxTextAttr::SetFlags, 1622  
wxTextAttr::SetFont, 1622  
wxTextAttr::SetLeftIndent, 1623  
wxTextAttr::SetRightIndent, 1623  
wxTextAttr::SetTabs, 1623  
wxTextAttr::SetTextColour, 1623  
wxTextAttr::wxTextAttr, 1620  
wxTextAttrEx, 1298, 1626  
wxTextAttrEx::GetBulletFont, 1626  
wxTextAttrEx::GetBulletName, 1626  
wxTextAttrEx::GetBulletNumber, 1626  
wxTextAttrEx::GetBulletStyle, 1626  
wxTextAttrEx::GetBulletText, 1627  
wxTextAttrEx::GetCharacterStyleName, 1627  
wxTextAttrEx::GetLineSpacing, 1627  
wxTextAttrEx::GetListStyleName, 1627  
wxTextAttrEx::GetOutlineLevel, 1627  
wxTextAttrEx::GetParagraphSpacingAfter, 1627  
wxTextAttrEx::GetParagraphSpacingBefore, 1627  
wxTextAttrEx::GetParagraphStyleName, 1627  
wxTextAttrEx::GetTextEffectFlags, 1628  
wxTextAttrEx::GetTextEffects, 1628  
wxTextAttrEx::GetURL, 1628  
wxTextAttrEx::HasBulletName, 1628  
wxTextAttrEx::HasBulletNumber, 1628  
wxTextAttrEx::HasBulletStyle, 1628  
wxTextAttrEx::HasBulletText, 1628  
wxTextAttrEx::HasCharacterStyleName, 1628  
wxTextAttrEx::HasLineSpacing, 1629  
wxTextAttrEx::HasListStyleName, 1629  
wxTextAttrEx::HasOutlineLevel, 1629  
wxTextAttrEx::HasPageBreak, 1629  
wxTextAttrEx::HasParagraphSpacingAfter, 1629  
wxTextAttrEx::HasParagraphSpacingBefore, 1629  
wxTextAttrEx::HasParagraphStyleName, 1629  
wxTextAttrEx::HasTextEffects, 1629  
wxTextAttrEx::HasURL, 1630  
wxTextAttrEx::Init, 1630  
wxTextAttrEx::IsCharacterStyle, 1630  
wxTextAttrEx::IsDefault, 1630  
wxTextAttrEx::IsParagraphStyle, 1630  
wxTextAttrEx::operator=, 1633  
wxTextAttrEx::SetBulletFont, 1630  
wxTextAttrEx::SetBulletName, 1630  
wxTextAttrEx::SetBulletNumber, 1630  
wxTextAttrEx::SetBulletStyle, 1631  
wxTextAttrEx::SetBulletText, 1631  
wxTextAttrEx::SetCharacterStyleName, 1631  
wxTextAttrEx::SetLineSpacing, 1631  
wxTextAttrEx::SetListStyleName, 1632  
wxTextAttrEx::SetOutlineLevel, 1632  
wxTextAttrEx::SetPageBreak, 1632  
wxTextAttrEx::SetParagraphSpacingAfter, 1632  
wxTextAttrEx::SetParagraphSpacingBefore, 1632  
wxTextAttrEx::SetParagraphStyleName, 1632  
wxTextAttrEx::SetTextEffectFlags, 1632  
wxTextAttrEx::SetTextEffects, 1632  
wxTextAttrEx::SetURL, 1633  
wxTextAttrEx::wxTextAttrEx, 1626  
wxTextCtrl, 1638  
wxTextCtrl::~wxTextCtrl, 1639  
wxTextCtrl::AppendText, 1639

wxTextCtrl::CanCopy, 1639  
wxTextCtrl::CanCut, 1639  
wxTextCtrl::CanPaste, 1640  
wxTextCtrl::CanRedo, 1640  
wxTextCtrl::CanUndo, 1640  
wxTextCtrl::ChangeValue, 1651  
wxTextCtrl::Clear, 1640  
wxTextCtrl::Copy, 1640  
wxTextCtrl::Create, 1640  
wxTextCtrl::Cut, 1640  
wxTextCtrl::DiscardEdits, 1641  
wxTextCtrl::EmulateKeyPress, 1641  
wxTextCtrl::GetDefaultStyle, 1641  
wxTextCtrl::GetInsertionPoint, 1641  
wxTextCtrl::GetLastPosition, 1641  
wxTextCtrl::GetLineLength, 1642  
wxTextCtrl::GetLineText, 1642  
wxTextCtrl::GetNumberOfLines, 1642  
wxTextCtrl::GetRange, 1643  
wxTextCtrl::GetSelection, 1643  
wxTextCtrl::GetStringSelection, 1643  
wxTextCtrl::GetStyle, 1643  
wxTextCtrl::GetValue, 1644  
wxTextCtrl::HitTest, 1644  
wxTextCtrl::IsEditable, 1644  
wxTextCtrl::IsEmpty, 1644  
wxTextCtrl::IsModified, 1645  
wxTextCtrl::IsMultiLine, 1645  
wxTextCtrl::IsSingleLine, 1645  
wxTextCtrl::LoadFile, 1645  
wxTextCtrl::MarkDirty, 1646  
wxTextCtrl::OnDropFiles, 1646  
wxTextCtrl::operator <<, 1653  
wxTextCtrl::Paste, 1646  
wxTextCtrl::PositionToXY, 1646  
wxTextCtrl::Redo, 1647  
wxTextCtrl::Remove, 1647  
wxTextCtrl::Replace, 1647  
wxTextCtrl::SaveFile, 1648  
wxTextCtrl::SetDefaultStyle, 1648  
wxTextCtrl::SetEditable, 1649  
wxTextCtrl::SetInsertionPoint, 1649  
wxTextCtrl::SetInsertionPointEnd, 1649  
wxTextCtrl::SetMaxLength, 1649  
wxTextCtrl::SetModified, 1650  
wxTextCtrl::SetSelection, 1650  
wxTextCtrl::SetStyle, 1650  
wxTextCtrl::SetValue, 1651  
wxTextCtrl::ShowPosition, 1652  
wxTextCtrl::Undo, 1652  
wxTextCtrl::WriteText, 1652  
wxTextCtrl::wxTextCtrl, 1638  
wxTextCtrl::XYToPosition, 1652  
wxTextDataObject, 1654  
wxTextDataObject::GetText, 1654  
wxTextDataObject::GetTextLength, 1654  
wxTextDataObject::SetText, 1654  
wxTextDataObject::wxTextDataObject, 1654  
wxTextDropTarget, 1655  
wxTextDropTarget::OnDrop, 1655  
wxTextDropTarget::OnDropText, 1655  
wxTextDropTarget::wxTextDropTarget, 1655  
wxTextEntryDialog, 1656  
wxTextEntryDialog overview, 2131  
wxTextEntryDialog::~wxTextEntryDialog, 1656  
wxTextEntryDialog::GetValue, 1657  
wxTextEntryDialog::SetValue, 1657  
wxTextEntryDialog::ShowModal, 1657  
wxTextEntryDialog::wxTextEntryDialog, 1656  
wxTextFile, 1658  
wxTextFile::~wxTextFile, 1659  
wxTextFile::AddLine, 1659  
wxTextFile::Clear, 1662  
wxTextFile::Close, 1659  
wxTextFile::Create, 1659  
wxTextFile::Eof, 1660  
wxTextFile::Exists, 1659  
wxTextFile::GetCurrentLine, 1660  
wxTextFile::GetEOL, 1660  
wxTextFile::GetFirstLine, 1660  
wxTextFile::GetLastLine, 1661  
wxTextFile::GetLine, 1659  
wxTextFile::GetLineCount, 1659  
wxTextFile::GetLineType, 1661  
wxTextFile::GetName, 1662  
wxTextFile::GetNextLine, 1661  
wxTextFile::GetPrevLine, 1661  
wxTextFile::GoToLine, 1660  
wxTextFile::GuessType, 1661  
wxTextFile::InsertLine, 1662  
wxTextFile::IsOpened, 1659  
wxTextFile::Open, 1662  
wxTextFile::operator[], 1660  
wxTextFile::RemoveLine, 1662  
wxTextFile::Write, 1662  
wxTextFile::wxTextFile, 1658  
wxTextInputStream, 1663  
wxTextInputStream::~wxTextInputStream, 1664  
wxTextInputStream::GetChar, 1665  
wxTextInputStream::Read16, 1664  
wxTextInputStream::Read16S, 1664  
wxTextInputStream::Read32, 1664  
wxTextInputStream::Read32S, 1665  
wxTextInputStream::Read8, 1664  
wxTextInputStream::Read8S, 1664  
wxTextInputStream::ReadDouble, 1665  
wxTextInputStream::ReadLine, 1665  
wxTextInputStream::ReadString, 1665  
wxTextInputStream::ReadWord, 1665  
wxTextInputStream::SetStringSeparators, 1665  
wxTextInputStream::wxTextInputStream, 1663  
wxTextOutputStream, 1666  
wxTextOutputStream::~wxTextOutputStream, 1666  
wxTextOutputStream::GetMode, 1667  
wxTextOutputStream::PutChar, 1667  
wxTextOutputStream::SetMode, 1667  
wxTextOutputStream::Write16, 1667  
wxTextOutputStream::Write32, 1667  
wxTextOutputStream::Write8, 1667  
wxTextOutputStream::WriteDouble, 1667  
wxTextOutputStream::WriteString, 1667  
wxTextOutputStream::wxTextOutputStream, 1666

wxTextValidator, 1668  
wxTextValidator::Clone, 1669  
wxTextValidator::GetExcludes, 1669  
wxTextValidator::GetIncludes, 1669  
wxTextValidator::GetStyle, 1669  
wxTextValidator::OnChar, 1669  
wxTextValidator::SetExcludes, 1670  
wxTextValidator::SetIncludes, 1670  
wxTextValidator::SetStyle, 1670  
wxTextValidator::TransferFromWindow, 1670  
wxTextValidator::TransferToWindow, 1670  
wxTextValidator::Validate, 1670  
wxTextValidator::wxTextValidator, 1668  
wxTGAHandler, 907  
wxTHICK\_FRAME, 497, 1047, 1052  
wxThread, 1672  
wxThread deletion, 1671  
wxThread::~~wxThread, 1673  
wxThread::Create, 1673  
wxThread::Delete, 1674  
wxThread::Entry, 1674  
wxThread::Exit, 1674  
wxThread::GetCPUCount, 1674  
wxThread::GetCurrentId, 1674  
wxThread::GetId, 1675  
wxThread::GetPriority, 1675  
wxThread::IsAlive, 1675  
wxThread::IsDetached, 1675  
wxThread::IsMain, 1675  
wxThread::IsPaused, 1675  
wxThread::IsRunning, 1676  
wxThread::Kill, 1676  
wxThread::OnExit, 1676  
wxThread::Pause, 1676  
wxThread::Resume, 1677  
wxThread::Run, 1676  
wxThread::SetConcurrency, 1677  
wxThread::SetPriority, 1677  
wxThread::Sleep, 1677  
wxThread::TestDestroy, 1677  
wxThread::This, 1678  
wxThread::Wait, 1678  
wxThread::wxThread, 1672  
wxThread::Yield, 1678  
wxThreadHelper, 1679  
wxThreadHelper::~~wxThreadHelper, 1679  
wxThreadHelper::Create, 1679  
wxThreadHelper::Entry, 1680  
wxThreadHelper::GetThread, 1680  
wxThreadHelper::m\_thread, 1679  
wxThreadHelper::wxThreadHelper, 1679  
wxTIFFHandler, 906  
wxTimer, 1681  
wxTimer::~~wxTimer, 1681  
wxTimer::GetInterval, 1681  
wxTimer::IsOneShot, 1681  
wxTimer::IsRunning, 1681  
wxTimer::Notify, 1682  
wxTimer::SetOwner, 1682  
wxTimer::Start, 1682  
wxTimer::Stop, 1682  
wxTimer::wxTimer, 1681  
wxTimerEvent::GetInterval, 1683  
wxTimeSpan, 1689  
wxTimeSpan::Abs, 1685  
wxTimeSpan::Add, 1685  
wxTimeSpan::Day, 1685  
wxTimeSpan::Days, 1685  
wxTimeSpan::Format, 1685  
wxTimeSpan::GetDays, 1686  
wxTimeSpan::GetHours, 1686  
wxTimeSpan::GetMilliseconds, 1686  
wxTimeSpan::GetMinutes, 1686  
wxTimeSpan::GetSeconds, 1686  
wxTimeSpan::GetValue, 1686  
wxTimeSpan::GetWeeks, 1687  
wxTimeSpan::Hour, 1687  
wxTimeSpan::Hours, 1687  
wxTimeSpan::IsEqualTo, 1687  
wxTimeSpan::IsLongerThan, 1687  
wxTimeSpan::IsNegative, 1687  
wxTimeSpan::IsNull, 1687  
wxTimeSpan::IsPositive, 1687  
wxTimeSpan::IsShorterThan, 1688  
wxTimeSpan::Millisecond, 1688  
wxTimeSpan::Milliseconds, 1688  
wxTimeSpan::Minute, 1688  
wxTimeSpan::Minutes, 1688  
wxTimeSpan::Multiply, 1688  
wxTimeSpan::Neg, 1688  
wxTimeSpan::Negate, 1688  
wxTimeSpan::Second, 1689  
wxTimeSpan::Seconds, 1689  
wxTimeSpan::Subtract, 1689  
wxTimeSpan::Week, 1689  
wxTimeSpan::Weeks, 1689  
wxTimeSpan::wxTimeSpan, 1689  
wxTINY\_CAPTION\_HORIZ, 1114  
wxTINY\_CAPTION\_VERT, 1114  
wxTipProvider, 1690  
wxTipProvider::GetTip, 1690  
wxTipProvider::PreprocessTip, 1690  
wxTipProvider::wxTipProvider, 1690  
wxTipWindow, 1691  
wxTipWindow::SetBoundingRect, 1692  
wxTipWindow::SetTipWindowPtr, 1692  
wxTipWindow::wxTipWindow, 1691  
wxToggleButton, 1693  
wxToggleButton::~~wxToggleButton, 1694  
wxToggleButton::Create, 1694  
wxToggleButton::GetValue, 1694  
wxToggleButton::SetValue, 1694  
wxToggleButton::wxToggleButton, 1693  
wxToolBar, 1697  
wxToolBar::~~wxToolBar, 1698  
wxToolBar::AddCheckTool, 1700  
wxToolBar::AddControl, 1698  
wxToolBar::AddRadioTool, 1700  
wxToolBar::AddSeparator, 1698  
wxToolBar::AddTool, 1698  
wxToolBar::ClearTools, 1700  
wxToolBar::DeleteTool, 1700  
wxToolBar::DeleteToolByPos, 1701  
wxToolBar::EnableTool, 1701



- wxToolBar::FindById, 1701
- wxToolBar::FindControl, 1701
- wxToolBar::FindToolForPosition, 1702
- wxToolBar::GetMargins, 1703
- wxToolBar::GetToolBitmapSize, 1702
- wxToolBar::GetToolClientData, 1703
- wxToolBar::GetToolEnabled, 1703
- wxToolBar::GetToolLongHelp, 1703
- wxToolBar::GetToolPacking, 1704
- wxToolBar::GetToolPos, 1704
- wxToolBar::GetToolsCount, 1702
- wxToolBar::GetToolSeparation, 1704
- wxToolBar::GetToolShortHelp, 1704
- wxToolBar::GetToolSize, 1702
- wxToolBar::GetToolState, 1705
- wxToolBar::InsertControl, 1705
- wxToolBar::InsertSeparator, 1705
- wxToolBar::InsertTool, 1705
- wxToolBar::OnLeftClick, 1706
- wxToolBar::OnMouseEnter, 1706
- wxToolBar::OnRightClick, 1707
- wxToolBar::Realize, 1707
- wxToolBar::RemoveTool, 1707
- wxToolBar::SetBitmapResource, 1708
- wxToolBar::SetMargins, 1708
- wxToolBar::SetToolBitmapSize, 1708
- wxToolBar::SetToolClientData, 1709
- wxToolBar::SetToolDisabledBitmap, 1709
- wxToolBar::SetToolLongHelp, 1709
- wxToolBar::SetToolNormalBitmap, 1710
- wxToolBar::SetToolPacking, 1710
- wxToolBar::SetToolSeparation, 1710
- wxToolBar::SetToolShortHelp, 1710
- wxToolBar::ToggleTool, 1711
- wxToolBar::wxToolBar, 1697
- wxToolTip, 1713
- wxToolTip::Enable, 1712
- wxToolTip::GetTip, 1713
- wxToolTip::GetWindow, 1713
- wxToolTip::SetDelay, 1712
- wxToolTip::SetTip, 1713
- wxToolTip::wxToolTip, 1712
- wxTOP, 1447
- wxTopLevelWindow::CanSetTransparent, 1713
- wxTopLevelWindow::EnableCloseButton, 1713
- wxTopLevelWindow::GetDefaultItem, 1714
- wxTopLevelWindow::GetIcon, 1714
- wxTopLevelWindow::GetIcons, 1714
- wxTopLevelWindow::GetTitle, 1714
- wxTopLevelWindow::HandleSettingChange, 1714
- wxTopLevelWindow::Iconize, 1715
- wxTopLevelWindow::IsActive, 1715
- wxTopLevelWindow::IsAlwaysMaximized, 1715
- wxTopLevelWindow::IsFullScreen, 1715
- wxTopLevelWindow::IsIconized, 1715
- wxTopLevelWindow::IsMaximized, 1715
- wxTopLevelWindow::IsUsingNativeDecorations, 1716
- wxTopLevelWindow::Maximize, 1716
- wxTopLevelWindow::RequestUserAttention, 1716
- wxTopLevelWindow::SetDefaultItem, 1716
- wxTopLevelWindow::SetIcon, 1717
- wxTopLevelWindow::SetIcons, 1717
- wxTopLevelWindow::SetLeftMenu, 1717
- wxTopLevelWindow::SetMaxSize, 1718
- wxTopLevelWindow::SetMinSize, 1718
- wxTopLevelWindow::SetRightMenu, 1719
- wxTopLevelWindow::SetShape, 1719
- wxTopLevelWindow::SetSizeHints, 1718
- wxTopLevelWindow::SetTitle, 1719
- wxTopLevelWindow::SetTransparent, 1720
- wxTopLevelWindow::ShouldPreventAppExit, 1720
- wxTopLevelWindow::ShowFullScreen, 1720
- wxTopLevelWindow::UseNativeDecorations, 1721
- wxTopLevelWindow::UseNativeDecorationsByDefault, 1721
- wxTR\_DEFAULT\_STYLE, 1729
- wxTR\_EDIT\_LABELS, 1728
- wxTR\_EXTENDED, 1729
- wxTR\_FULL\_ROW\_HIGHLIGHT, 1729
- wxTR\_HAS\_BUTTONS, 1729
- wxTR\_HAS\_VARIABLE\_ROW\_HEIGHT, 1729
- wxTR\_HIDE\_ROOT, 1729
- wxTR\_LINES\_AT\_ROOT, 1729
- wxTR\_MULTIPLE, 1729
- wxTR\_NO\_BUTTONS, 1728
- wxTR\_NO\_LINES, 1729
- wxTR\_ROW\_LINES, 1729
- wxTR\_SINGLE, 1729
- wxTrace, 1976
- WXTRACE, 1976
- wxTraceLevel, 1977
- WXTRACELEVEL, 1976
- wxTransferFileToStream, 1925
- wxTransferStreamToFile, 1925
- wxTRANSLATE, 1932
- wxTRANSPARENT\_WINDOW, 1796
- wxTrap, 1983
- wxTreebook, 1722, 1723
- wxTreebook::~wxTreebook, 1723
- wxTreebook::AddPage, 1723
- wxTreebook::AddSubPage, 1723
- wxTreebook::AssignImageList, 1724
- wxTreebook::ChangeSelection, 1726
- wxTreebook::CollapseNode, 1724
- wxTreebook::Create, 1724
- wxTreebook::DeleteAllPages, 1724
- wxTreebook::DeletePage, 1724
- wxTreebook::ExpandNode, 1724
- wxTreebook::GetPageImage, 1724
- wxTreebook::GetPageParent, 1725
- wxTreebook::GetPageText, 1725
- wxTreebook::GetSelection, 1725
- wxTreebook::InsertPage, 1725
- wxTreebook::InsertSubPage, 1725
- wxTreebook::IsNodeExpanded, 1725
- wxTreebook::SetImageList, 1726
- wxTreebook::SetPageImage, 1726
- wxTreebook::SetPageText, 1726
- wxTreebook::SetSelection, 1726
- wxTreebook::wxTreebook, 1722
- wxTreebookEvent, 1728

---

wxTreebookEvent::GetOldSelection, 1728  
wxTreebookEvent::GetSelection, 1728  
wxTreebookEvent::wxTreebookEvent, 1728  
wxTreeCtrl, 1731, 1734  
wxTreeCtrl::~wxTreeCtrl, 1732  
wxTreeCtrl::AddRoot, 1732  
wxTreeCtrl::AppendItem, 1732  
wxTreeCtrl::AssignButtonsImageList, 1732  
wxTreeCtrl::AssignImageList, 1732  
wxTreeCtrl::AssignStateImageList, 1733  
wxTreeCtrl::Collapse, 1733  
wxTreeCtrl::CollapseAll, 1733  
wxTreeCtrl::CollapseAllChildren, 1733  
wxTreeCtrl::CollapseAndReset, 1733  
wxTreeCtrl::Create, 1733  
wxTreeCtrl::Delete, 1734  
wxTreeCtrl::DeleteAllItems, 1734  
wxTreeCtrl::DeleteChildren, 1734  
wxTreeCtrl::EditLabel, 1734  
wxTreeCtrl::EndEditLabel, 1734  
wxTreeCtrl::EnsureVisible, 1735  
wxTreeCtrl::Expand, 1735  
wxTreeCtrl::ExpandAll, 1735  
wxTreeCtrl::ExpandAllChildren, 1735  
wxTreeCtrl::GetBoundingRect, 1735  
wxTreeCtrl::GetButtonsImageList, 1736  
wxTreeCtrl::GetChildrenCount, 1736  
wxTreeCtrl::GetCount, 1736  
wxTreeCtrl::GetEditControl, 1736  
wxTreeCtrl::GetFirstChild, 1736  
wxTreeCtrl::GetFirstVisibleItem, 1737  
wxTreeCtrl::GetImageList, 1737  
wxTreeCtrl::GetIndent, 1737  
wxTreeCtrl::GetItemBackgroundColour, 1737  
wxTreeCtrl::GetItemData, 1737  
wxTreeCtrl::GetItemFont, 1738  
wxTreeCtrl::GetItemImage, 1738  
wxTreeCtrl::GetItemParent, 1739  
wxTreeCtrl::GetItemSelectedImage, 1740  
wxTreeCtrl::GetItemText, 1738  
wxTreeCtrl::GetItemTextColour, 1738  
wxTreeCtrl::GetLastChild, 1738  
wxTreeCtrl::GetNextChild, 1739  
wxTreeCtrl::GetNextSibling, 1739  
wxTreeCtrl::GetNextVisible, 1739  
wxTreeCtrl::GetPrevSibling, 1739  
wxTreeCtrl::GetPrevVisible, 1740  
wxTreeCtrl::GetQuickBestSize, 1740  
wxTreeCtrl::GetRootItem, 1740  
wxTreeCtrl::GetSelection, 1740  
wxTreeCtrl::GetSelections, 1740  
wxTreeCtrl::GetStateImageList, 1741  
wxTreeCtrl::HitTest, 1741  
wxTreeCtrl::InsertItem, 1741  
wxTreeCtrl::IsBold, 1742  
wxTreeCtrl::IsEmpty, 1742  
wxTreeCtrl::IsExpanded, 1742  
wxTreeCtrl::IsSelected, 1742  
wxTreeCtrl::IsVisible, 1742  
wxTreeCtrl::ItemHasChildren, 1742  
wxTreeCtrl::OnCompareItems, 1743  
wxTreeCtrl::PrependItem, 1743  
wxTreeCtrl::ScrollTo, 1743  
wxTreeCtrl::SelectItem, 1743  
wxTreeCtrl::SetButtonsImageList, 1743  
wxTreeCtrl::SetImageList, 1744  
wxTreeCtrl::SetIndent, 1744  
wxTreeCtrl::SetItemBackgroundColour, 1744  
wxTreeCtrl::SetItemBold, 1744  
wxTreeCtrl::SetItemData, 1744  
wxTreeCtrl::SetItemDropHighlight, 1745  
wxTreeCtrl::SetItemFont, 1745  
wxTreeCtrl::SetItemHasChildren, 1745  
wxTreeCtrl::SetItemImage, 1745  
wxTreeCtrl::SetItemSelectedImage, 1745  
wxTreeCtrl::SetItemText, 1745  
wxTreeCtrl::SetItemTextColour, 1746  
wxTreeCtrl::SetQuickBestSize, 1746  
wxTreeCtrl::SetStateImageList, 1746  
wxTreeCtrl::SetWindowStyle, 1746  
wxTreeCtrl::SortChildren, 1746  
wxTreeCtrl::Toggle, 1746  
wxTreeCtrl::ToggleItemSelection, 1747  
wxTreeCtrl::Unselect, 1747  
wxTreeCtrl::UnselectAll, 1747  
wxTreeCtrl::UnselectItem, 1747  
wxTreeCtrl::wxTreeCtrl, 1731  
wxTreeEvent, 1750  
wxTreeEvent::GetItem, 1750  
wxTreeEvent::GetKeyCode, 1750  
wxTreeEvent::GetKeyEvent, 1750  
wxTreeEvent::GetLabel, 1750  
wxTreeEvent::GetOldItem, 1750  
wxTreeEvent::GetPoint, 1751  
wxTreeEvent::IsEditCancelled, 1751  
wxTreeEvent::SetToolTip, 1751  
wxTreeEvent::wxTreeEvent, 1750  
wxTreeItemData, 1752  
wxTreeItemData::~wxTreeItemData, 1752  
wxTreeItemData::GetId, 1752  
wxTreeItemData::SetId, 1752  
wxTreeItemData::wxTreeItemData, 1751  
wxTreeItemId, 1748  
wxTreeItemId::IsOk, 1748  
wxTreeItemId::wxTreeItemId, 1747  
wxUINT16\_SWAP\_ALWAYS, 1963  
wxUINT16\_SWAP\_ON\_BE, 1964  
wxUINT16\_SWAP\_ON\_LE, 1964  
wxUINT32\_SWAP\_ALWAYS, 1963  
wxUINT32\_SWAP\_ON\_BE, 1964  
wxUINT32\_SWAP\_ON\_LE, 1964  
wxULL, 1962  
wxUninitialize, 1912  
wxUnix2DosFilename, 1922  
wxUnsetEnv, 1984  
wxUpdateUIEvent, 1754  
wxUpdateUIEvent::CanUpdate, 1754  
wxUpdateUIEvent::Check, 1754  
wxUpdateUIEvent::Enable, 1754  
wxUpdateUIEvent::GetChecked, 1755  
wxUpdateUIEvent::GetEnabled, 1755  
wxUpdateUIEvent::GetMode, 1756  
wxUpdateUIEvent::GetSetChecked, 1755  
wxUpdateUIEvent::GetSetEnabled, 1755

- wxUpdateUIEvent::GetSetShown, 1755
- wxUpdateUIEvent::GetSetText, 1755
- wxUpdateUIEvent::GetShown, 1755
- wxUpdateUIEvent::GetText, 1756
- wxUpdateUIEvent::GetUpdateInterval, 1756
- wxUpdateUIEvent::ResetUpdateTime, 1756
- wxUpdateUIEvent::SetMode, 1756
- wxUpdateUIEvent::SetText, 1757
- wxUpdateUIEvent::SetUpdateInterval, 1757
- wxUpdateUIEvent::Show, 1754
- wxUpdateUIEvent::wxUpdateUIEvent, 1754
- wxURI, 1758
- wxURI::BuildUnescapedURI, 1759
- wxURI::BuildURI, 1759
- wxURI::Create, 1759
- wxURI::GetFragment, 1759
- wxURI::GetHostType, 1759
- wxURI::GetPassword, 1760
- wxURI::GetPath, 1760
- wxURI::GetPort, 1760
- wxURI::GetQuery, 1760
- wxURI::GetScheme, 1761
- wxURI::GetServer, 1761
- wxURI::GetUser, 1761
- wxURI::GetUserInfo, 1761
- wxURI::HasFragment, 1761
- wxURI::HasPath, 1761
- wxURI::HasPort, 1762
- wxURI::HasQuery, 1762
- wxURI::HasScheme, 1762
- wxURI::HasServer, 1762
- wxURI::HasUser, 1762
- wxURI::IsReference, 1762
- wxURI::operator ==, 1762
- wxURI::Resolve, 1763
- wxURI::Unescape, 1763
- wxURI::wxURI, 1758
- wxURL, 1764
- wxURL::~~wxURL, 1764
- wxURL::GetError, 1764
- wxURL::GetInputStream, 1765
- wxURL::GetProtocol, 1764
- wxURL::IsOk, 1765
- wxURL::SetDefaultProxy, 1765
- wxURL::SetProxy, 1766
- wxURL::SetURL, 1766
- wxURL::wxURL, 1764
- wxURLDataObject, 1767
- wxURLDataObject::GetURL, 1767
- wxURLDataObject::SetURL, 1767
- wxURLDataObject::wxURLDataObject, 1767
- wxUsleep, 1979
- wxVaCopy, 1962
- wxValidator, 1768
- wxValidator::~~wxValidator, 1768
- wxValidator::Clone, 1768
- wxValidator::GetWindow, 1768
- wxValidator::SetBellOnError, 1768
- wxValidator::SetWindow, 1768
- wxValidator::TransferFromWindow, 1769
- wxValidator::TransferToWindow, 1769
- wxValidator::Validate, 1769
- wxValidator::wxValidator, 1768
- wxVariant, 1770, 1771
- wxVariant::~~wxVariant, 1772
- wxVariant::Append, 1772
- wxVariant::Clear, 1772
- wxVariant::ClearList, 1772
- wxVariant::Convert, 1772
- wxVariant::Delete, 1772
- wxVariant::GetArrayString, 1773
- wxVariant::GetBool, 1773
- wxVariant::GetChar, 1773
- wxVariant::GetCount, 1772
- wxVariant::GetData, 1773
- wxVariant::GetDateTime, 1773
- wxVariant::GetDouble, 1773
- wxVariant::GetLong, 1773
- wxVariant::GetName, 1773
- wxVariant::GetString, 1774
- wxVariant::GetType, 1774
- wxVariant::GetVoidPtr, 1774
- wxVariant::GetWxObjectPtr, 1774
- wxVariant::Insert, 1774
- wxVariant::IsNull, 1774
- wxVariant::IsType, 1774
- wxVariant::IsValueKindOf, 1774
- wxVariant::MakeNull, 1775
- wxVariant::MakeString, 1775
- wxVariant::Member, 1775
- wxVariant::NullList, 1775
- wxVariant::operator !=, 1776
- wxVariant::operator [], 1777
- wxVariant::operator =, 1775
- wxVariant::operator ==, 1776
- wxVariant::operator double, 1777
- wxVariant::operator void\*, 1777
- wxVariant::operator wxChar, 1777
- wxVariant::operator wxDateTime, 1778
- wxVariant::operator wxString, 1777
- wxVariant::SetData, 1775
- wxVariant::wxVariant, 1770
- wxVariantData, 1778
- wxVariantData::DecRef, 1778
- wxVariantData::Eq, 1779
- wxVariantData::GetType, 1779
- wxVariantData::GetValueClassInfo, 1779
- wxVariantData::IncRef, 1779
- wxVariantData::Read, 1779
- wxVariantData::Write, 1779
- wxVariantData::wxVariantData, 1778
- wxView, 1780
- wxView overview, 2134
- wxView::~~wxView, 1781
- wxView::Activate, 1781
- wxView::Close, 1781
- wxView::GetDocument, 1781
- wxView::GetDocumentManager, 1781
- wxView::GetFrame, 1781
- wxView::GetViewName, 1781
- wxView::m\_viewDocument, 1780
- wxView::m\_viewFrame, 1780
- wxView::m\_viewTypeName, 1780
- wxView::OnActivateView, 1782

- wxView::OnChangeFilename, 1782
- wxView::OnClose, 1782
- wxView::OnClosingDocument, 1782
- wxView::OnCreate, 1782
- wxView::OnCreatePrintout, 1782
- wxView::OnDraw, 1783
- wxView::OnUpdate, 1783
- wxView::SetDocument, 1783
- wxView::SetFrame, 1783
- wxView::SetViewName, 1783
- wxView::wxView, 1780
- wxVListBox, 1784
- wxVListBox::Clear, 1784
- wxVListBox::Create, 1784
- wxVListBox::DeselectAll, 1785
- wxVListBox::GetFirstSelected, 1785
- wxVListBox::GetItemCount, 1785
- wxVListBox::GetMargins, 1786
- wxVListBox::GetNextSelected, 1786
- wxVListBox::GetSelectedCount, 1786
- wxVListBox::GetSelection, 1786
- wxVListBox::GetSelectionBackground, 1786
- wxVListBox::HasMultipleSelection, 1787
- wxVListBox::IsCurrent, 1787
- wxVListBox::IsSelected, 1787
- wxVListBox::OnDrawBackground, 1787
- wxVListBox::OnDrawItem, 1787
- wxVListBox::OnDrawSeparator, 1788
- wxVListBox::OnMeasureItem, 1788
- wxVListBox::Select, 1788
- wxVListBox::SelectAll, 1789
- wxVListBox::SelectRange, 1789
- wxVListBox::SetItemCount, 1789
- wxVListBox::SetMargins, 1789
- wxVListBox::SetSelection, 1789
- wxVListBox::SetSelectionBackground, 1790
- wxVListBox::Toggle, 1790
- wxVListBox::wxVListBox, 1784
- wxVLogDebug, 1974
- wxVLogError, 1972
- wxVLogFatalError, 1972
- wxVLogMessage, 1973
- wxVLogStatus, 1973
- wxVLogSysError, 1973
- wxVLogTrace, 1974
- wxVLogVerbose, 1973
- wxVLogWarning, 1972
- wxVSCROLL, 1052, 1797
- wxVScrolledWindow, 1791
- wxVScrolledWindow::Create, 1791
- wxVScrolledWindow::EstimateTotalHeight, 1792
- wxVScrolledWindow::GetFirstVisibleLine, 1792
- wxVScrolledWindow::GetLastVisibleLine, 1792
- wxVScrolledWindow::GetLineCount, 1792
- wxVScrolledWindow::GetVisibleBegin, 1793
- wxVScrolledWindow::GetVisibleEnd, 1793
- wxVScrolledWindow::HitTest, 1793
- wxVScrolledWindow::IsVisible, 1793
- wxVScrolledWindow::OnGetLineHeight, 1793
- wxVScrolledWindow::OnGetLinesHint, 1794
- wxVScrolledWindow::RefreshAll, 1794
- wxVScrolledWindow::RefreshLine, 1794
- wxVScrolledWindow::RefreshLines, 1794
- wxVScrolledWindow::ScrollLines, 1794
- wxVScrolledWindow::ScrollPages, 1795
- wxVScrolledWindow::ScrollToLine, 1795
- wxVScrolledWindow::SetLineCount, 1795
- wxVScrolledWindow::wxVScrolledWindow, 1791
- wxVsprintf, 1933
- wxWakeUpIdle, 1912
- wxWANTS\_CHARS, 1796
- wxWidgets 1.xx compatibility functions, 1557
- wxWidgets calls in secondary threads, 1672
- wxWidgets classes implemented in wxPython, 2203
- wxWidgets predefined command identifiers, 2137
- wxWinCE, 2235
- wxWindow, 1798
- wxWindow::~wxWindow, 1799
- wxWindow::AddChild, 1799
- wxWindow::CacheBestSize, 1799
- wxWindow::CaptureMouse, 1799
- wxWindow::Center, 1800
- wxWindow::CenterOnParent, 1800
- wxWindow::CenterOnScreen, 1800
- wxWindow::Centre, 1800
- wxWindow::CentreOnParent, 1801
- wxWindow::CentreOnScreen, 1801
- wxWindow::ClearBackground, 1801
- wxWindow::ClientToScreen, 1801
- wxWindow::ClientToWindowSize, 1802
- wxWindow::Close, 1802
- wxWindow::ConvertDialogToPixels, 1803
- wxWindow::ConvertPixelsToDialog, 1804
- wxWindow::Destroy, 1804
- wxWindow::DestroyChildren, 1805
- wxWindow::Disable, 1805
- wxWindow::DoGetBestSize, 1805
- wxWindow::DoUpdateWindowUI, 1805
- wxWindow::DragAcceptFiles, 1806
- wxWindow::Enable, 1806
- wxWindow::FindFocus, 1806
- wxWindow::FindWindow, 1807
- wxWindow::FindWindowById, 1807
- wxWindow::FindWindowByLabel, 1807
- wxWindow::FindWindowByName, 1808
- wxWindow::Fit, 1808
- wxWindow::FitInside, 1808
- wxWindow::Freeze, 1808
- wxWindow::GetAcceleratorTable, 1809
- wxWindow::GetAccessible, 1809
- wxWindow::GetAdjustedBestSize, 1809
- wxWindow::GetBackgroundColour, 1809
- wxWindow::GetBackgroundStyle, 1809
- wxWindow::GetBestSize, 1810
- wxWindow::GetCapture, 1810
- wxWindow::GetCaret, 1810
- wxWindow::GetCharHeight, 1810
- wxWindow::GetCharWidth, 1810
- wxWindow::GetChildren, 1810
- wxWindow::GetClassDefaultAttributes, 1811
- wxWindow::GetClientSize, 1811
- wxWindow::GetConstraints, 1812
- wxWindow::GetContainingSizer, 1812

wxWindow::GetCursor, 1812  
wxWindow::GetDefaultAttributes, 1812  
wxWindow::GetDropTarget, 1813  
wxWindow::GetEffectiveMinSize, 1810  
wxWindow::GetEventHandler, 1813  
wxWindow::GetExtraStyle, 1813  
wxWindow::GetFont, 1813  
wxWindow::GetForegroundColour, 1813  
wxWindow::GetGrandParent, 1814  
wxWindow::GetHandle, 1814  
wxWindow::GetHelpText, 1814  
wxWindow::GetHelpTextAtPoint, 1814  
wxWindow::GetId, 1815  
wxWindow::GetLabel, 1815  
wxWindow::GetMaxSize, 1815  
wxWindow::GetMinSize, 1815  
wxWindow::GetName, 1815  
wxWindow::GetNextSibling, 1816  
wxWindow::GetParent, 1816  
wxWindow::GetPosition, 1816  
wxWindow::GetPrevSibling, 1817  
wxWindow::GetRect, 1817  
wxWindow::GetScreenPosition, 1817  
wxWindow::GetScreenRect, 1818  
wxWindow::GetScrollPos, 1818  
wxWindow::GetScrollRange, 1818  
wxWindow::GetScrollThumb, 1818  
wxWindow::GetSize, 1818  
wxWindow::GetSizer, 1819  
wxWindow::GetTextExtent, 1819  
wxWindow::GetToolTip, 1820  
wxWindow::GetUpdateRegion, 1820  
wxWindow::GetValidator, 1820  
wxWindow::GetVirtualSize, 1821  
wxWindow::GetWindowBorderSize, 1821  
wxWindow::GetWindowStyleFlag, 1821  
wxWindow::GetWindowVariant, 1821  
wxWindow::HasCapture, 1821  
wxWindow::HasFlag, 1822  
wxWindow::HasMultiplePages, 1822  
wxWindow::HasScrollbar, 1822  
wxWindow::HasTransparentBackground, 1822  
wxWindow::Hide, 1822  
wxWindow::InheritAttributes, 1822  
wxWindow::InitDialog, 1823  
wxWindow::InvalidateBestSize, 1823  
wxWindow::IsDoubleBuffered, 1823  
wxWindow::IsEnabled, 1823  
wxWindow::IsExposed, 1824  
wxWindow::IsFrozen, 1824  
wxWindow::IsRetained, 1824  
wxWindow::IsShown, 1824  
wxWindow::IsShownOnScreen, 1825  
wxWindow::IsTopLevel, 1825  
wxWindow::Layout, 1825  
wxWindow::LineDown, 1825  
wxWindow::LineUp, 1825  
wxWindow::Lower, 1825  
wxWindow::MakeModal, 1825  
wxWindow::Move, 1826  
wxWindow::MoveAfterInTabOrder, 1827  
wxWindow::MoveBeforeInTabOrder, 1827  
wxWindow::Navigate, 1827  
wxWindow::OnInternalIdle, 1827  
wxWindow::PageDown, 1828  
wxWindow::PageUp, 1828  
wxWindow::PopEventHandler, 1828  
wxWindow::PopupMenu, 1828  
wxWindow::PushEventHandler, 1829  
wxWindow::Raise, 1830  
wxWindow::Refresh, 1830  
wxWindow::RefreshRect, 1830  
wxWindow::RegisterHotKey, 1830  
wxWindow::ReleaseMouse, 1831  
wxWindow::RemoveChild, 1831  
wxWindow::RemoveEventHandler, 1832  
wxWindow::Reparent, 1832  
wxWindow::ScreenToClient, 1832  
wxWindow::ScrollLines, 1833  
wxWindow::ScrollPages, 1833  
wxWindow::ScrollWindow, 1834  
wxWindow::SetAcceleratorTable, 1834  
wxWindow::SetAccessible, 1834  
wxWindow::SetAutoLayout, 1834  
wxWindow::SetBackgroundColour, 1835  
wxWindow::SetBackgroundStyle, 1836  
wxWindow::SetCaret, 1836  
wxWindow::SetClientSize, 1836  
wxWindow::SetConstraints, 1837  
wxWindow::SetContainingSizer, 1837  
wxWindow::SetCursor, 1837  
wxWindow::SetDropTarget, 1838  
wxWindow::SetEventHandler, 1838  
wxWindow::SetExtraStyle, 1839  
wxWindow::SetFocus, 1839  
wxWindow::SetFocusFromKbd, 1840  
wxWindow::SetFont, 1840  
wxWindow::SetForegroundColour, 1840  
wxWindow::SetHelpText, 1841  
wxWindow::SetId, 1841  
wxWindow::SetInitialBestSize, 1838  
wxWindow::SetInitialSize, 1836  
wxWindow::SetLabel, 1841  
wxWindow::SetMaxSize, 1842  
wxWindow::SetMinSize, 1842  
wxWindow::SetName, 1842  
wxWindow::SetOwnBackgroundColour, 1842  
wxWindow::SetOwnFont, 1843  
wxWindow::SetOwnForegroundColour, 1843  
wxWindow::SetPalette, 1843  
wxWindow::SetScrollbar, 1843  
wxWindow::SetScrollPos, 1844  
wxWindow::SetSize, 1845  
wxWindow::SetSizeHints, 1846  
wxWindow::SetSizer, 1846  
wxWindow::SetSizerAndFit, 1847  
wxWindow::SetThemeEnabled, 1847  
wxWindow::SetToolTip, 1847  
wxWindow::SetValidator, 1848  
wxWindow::SetVirtualSize, 1848  
wxWindow::SetVirtualSizeHints, 1848  
wxWindow::SetWindowStyle, 1849  
wxWindow::SetWindowStyleFlag, 1849  
wxWindow::SetWindowVariant, 1849

---

wxWindow::ShouldInheritColours, 1849  
wxWindow::Show, 1850  
wxWindow::Thaw, 1850  
wxWindow::ToggleWindowStyle, 1850  
wxWindow::TransferDataFromWindow, 1851  
wxWindow::TransferDataToWindow, 1851  
wxWindow::UnregisterHotKey, 1851  
wxWindow::Update, 1852  
wxWindow::UpdateWindowUI, 1852  
wxWindow::Validate, 1853  
wxWindow::WarpPointer, 1853  
wxWindow::WindowToClientSize, 1853  
wxWindow::wxWindow, 1798  
wxWindowCreateEvent, 1855  
wxWindowCreateEvent::wxWindowCreateEvent, 1855  
wxWindowDC, 1856  
wxWindowDC::wxWindowDC, 1856  
wxWindowDestroyEvent, 1856  
wxWindowDestroyEvent::wxWindowDestroyEvent, 1856  
wxWindowDisabler, 1857  
wxWindowDisabler::~wxWindowDisabler, 1857  
wxWindowDisabler::wxWindowDisabler, 1857  
wxWindowUpdateLocker, 1854  
wxWindowUpdateLocker::~wxWindowUpdateLocker, 1854  
wxWindowUpdateLocker::wxWindowUpdateLocker, 1854  
wxWizard, 1858, 1859  
wxWizard::Create, 1859  
wxWizard::FitToPage, 1860  
wxWizard::GetBitmap, 1860  
wxWizard::GetCurrentPage, 1860  
wxWizard::GetPageAreaSizer, 1860  
wxWizard::GetPageSize, 1861  
wxWizard::HasNextPage, 1861  
wxWizard::HasPrevPage, 1861  
wxWizard::RunWizard, 1862  
wxWizard::SetBitmap, 1862  
wxWizard::SetBorder, 1862  
wxWizard::SetPageSize, 1862  
wxWizard::wxWizard, 1858  
wxWIZARD\_EX\_HELPBUTTON, 1858  
wxWizardEvent, 1863  
wxWizardEvent::GetDirection, 1863  
wxWizardEvent::GetPage, 1863  
wxWizardEvent::wxWizardEvent, 1863  
wxWizardPage, 1864  
wxWizardPage::GetBitmap, 1865  
wxWizardPage::GetNext, 1865  
wxWizardPage::GetPrev, 1865  
wxWizardPage::wxWizardPage, 1864  
wxWizardPageSimple, 1866  
wxWizardPageSimple::Chain, 1866  
wxWizardPageSimple::SetNext, 1866  
wxWizardPageSimple::SetPrev, 1866  
wxWizardPageSimple::wxWizardPageSimple, 1866  
wxWriteResource, 1962, 1963  
wxWS\_EX\_BLOCK\_EVENTS, 1797, 1839  
wxWS\_EX\_CONTEXTHELP, 1839  
wxWS\_EX\_PROCESS\_IDLE, 1798, 1839  
wxWS\_EX\_PROCESS\_UI\_UPDATES, 1798, 1839  
wxWS\_EX\_TRANSIENT, 1797, 1839  
wxWS\_EX\_VALIDATE\_RECURSIVELY, 1797, 1839  
wxXCharBuffer Overview, 2052  
wxXmlDocument, 1868  
wxXmlDocument::~wxXmlDocument, 1868  
wxXmlDocument::DetachRoot, 1868  
wxXmlDocument::GetEncoding, 1869  
wxXmlDocument::GetFileEncoding, 1869  
wxXmlDocument::GetRoot, 1869  
wxXmlDocument::GetVersion, 1869  
wxXmlDocument::IsOk, 1869  
wxXmlDocument::Load, 1869  
wxXmlDocument::operator=, 1871  
wxXmlDocument::Save, 1870  
wxXmlDocument::SetEncoding, 1870  
wxXmlDocument::SetFileEncoding, 1870  
wxXmlDocument::SetRoot, 1870  
wxXmlDocument::SetVersion, 1870  
wxXmlDocument::wxXmlDocument, 1868  
wxXmlNode, 1872  
wxXmlNode::~wxXmlNode, 1872  
wxXmlNode::AddChild, 1873  
wxXmlNode::AddProperty, 1873  
wxXmlNode::DeleteProperty, 1873  
wxXmlNode::GetChildren, 1873  
wxXmlNode::GetContent, 1873  
wxXmlNode::GetDepth, 1873  
wxXmlNode::GetName, 1874  
wxXmlNode::GetNext, 1874  
wxXmlNode::GetNodeContent, 1873  
wxXmlNode::GetParent, 1874  
wxXmlNode::GetProperties, 1874  
wxXmlNode::GetPropVal, 1874  
wxXmlNode::GetType, 1875  
wxXmlNode::HasProp, 1875  
wxXmlNode::InsertChild, 1875  
wxXmlNode::InsertChildAfter, 1875  
wxXmlNode::IsWhitespaceOnly, 1875  
wxXmlNode::operator=, 1877  
wxXmlNode::RemoveChild, 1876  
wxXmlNode::SetChildren, 1876  
wxXmlNode::SetContent, 1876  
wxXmlNode::SetName, 1876  
wxXmlNode::SetNext, 1876  
wxXmlNode::SetParent, 1876  
wxXmlNode::SetProperties, 1876  
wxXmlNode::SetType, 1877  
wxXmlNode::wxXmlNode, 1872  
wxXmlProperty, 1877  
wxXmlProperty::~wxXmlProperty, 1877  
wxXmlProperty::GetName, 1878  
wxXmlProperty::GetNext, 1878  
wxXmlProperty::GetValue, 1878  
wxXmlProperty::SetName, 1878  
wxXmlProperty::SetNext, 1878  
wxXmlProperty::SetValue, 1878  
wxXmlProperty::wxXmlProperty, 1877  
wxXmlResource, 1879

---

- 
- wxXmlResource::~wxXmlResource, 1880
  - wxXmlResource::AddHandler, 1880
  - wxXmlResource::AttachUnknownControl, 1880
  - wxXmlResource::ClearHandlers, 1880
  - wxXmlResource::CompareVersion, 1880
  - wxXmlResource::Get, 1880
  - wxXmlResource::GetDomain, 1883
  - wxXmlResource::GetFlags, 1880
  - wxXmlResource::GetVersion, 1881
  - wxXmlResource::GetXRCID, 1881
  - wxXmlResource::InitAllHandlers, 1881
  - wxXmlResource::Load, 1881
  - wxXmlResource::LoadBitmap, 1881
  - wxXmlResource::LoadDialog, 1881
  - wxXmlResource::LoadFrame, 1882
  - wxXmlResource::LoadIcon, 1882
  - wxXmlResource::LoadMenu, 1882
  - wxXmlResource::LoadMenuBar, 1882
  - wxXmlResource::LoadPanel, 1882
  - wxXmlResource::LoadToolBar, 1882
  - wxXmlResource::Set, 1883
  - wxXmlResource::SetDomain, 1883
  - wxXmlResource::SetFlags, 1883
  - wxXmlResource::Unload, 1883
  - wxXmlResource::wxXmlResource, 1879
  - wxXmlResourceHandler, 1884
  - wxXmlResourceHandler::~wxXmlResourceHandler, 1884
  - wxXmlResourceHandler::AddStyle, 1884
  - wxXmlResourceHandler::AddWindowStyles, 1884
  - wxXmlResourceHandler::CanHandle, 1884
  - wxXmlResourceHandler::CreateChildren, 1884
  - wxXmlResourceHandler::CreateChildrenPrivately, 1884
  - wxXmlResourceHandler::CreateResFromNode, 1884
  - wxXmlResourceHandler::CreateResource, 1885
  - wxXmlResourceHandler::DoCreateResource, 1885
  - wxXmlResourceHandler::GetBitmap, 1885
  - wxXmlResourceHandler::GetBool, 1885
  - wxXmlResourceHandler::GetColour, 1885
  - wxXmlResourceHandler::GetCurFilesystem, 1885
  - wxXmlResourceHandler::GetDimension, 1885
  - wxXmlResourceHandler::GetFont, 1886
  - wxXmlResourceHandler::GetIcon, 1886
  - wxXmlResourceHandler::GetID, 1886
  - wxXmlResourceHandler::GetLong, 1886
  - wxXmlResourceHandler::GetName, 1886
  - wxXmlResourceHandler::GetNodeContent, 1886
  - wxXmlResourceHandler::GetParamNode, 1886
  - wxXmlResourceHandler::GetParamValue, 1886
  - wxXmlResourceHandler::GetPosition, 1887
  - wxXmlResourceHandler::GetSize, 1887
  - wxXmlResourceHandler::GetStyle, 1887
  - wxXmlResourceHandler::GetText, 1887
  - wxXmlResourceHandler::HasParam, 1887
  - wxXmlResourceHandler::IsOfClass, 1887
  - wxXmlResourceHandler::SetParentResource, 1887
  - wxXmlResourceHandler::SetupWindow, 1888
  - wxXmlResourceHandler::wxXmlResourceHandler, 1884
  - wxXPMHandler, 907
  - wxYield, 1912
  - wxZipEntry, 1891
  - wxZipEntry::Clone, 1891
  - wxZipEntry::Get/SetComment, 1891
  - wxZipEntry::Get/SetExternalAttributes, 1892
  - wxZipEntry::Get/SetExtra, 1892
  - wxZipEntry::Get/SetLocalExtra, 1893
  - wxZipEntry::Get/SetMethod, 1893
  - wxZipEntry::Get/SetMode, 1893
  - wxZipEntry::Get/SetSystemMadeBy, 1894
  - wxZipEntry::GetCompressedSize, 1891
  - wxZipEntry::GetCrc, 1892
  - wxZipEntry::GetFlags, 1892
  - wxZipEntry::GetInternalName, 1892
  - wxZipEntry::IsMadeByUnix, 1894
  - wxZipEntry::IsText/SetIsText, 1894
  - wxZipEntry::operator=, 1894
  - wxZipEntry::SetNotifier, 1894
  - wxZipEntry::wxZipEntry, 1891
  - wxZipInputStream, 1895
  - wxZipInputStream::CloseEntry, 1895
  - wxZipInputStream::GetComment, 1896
  - wxZipInputStream::GetNextEntry, 1896
  - wxZipInputStream::GetTotalEntries, 1896
  - wxZipInputStream::OpenEntry, 1896
  - wxZipInputStream::wxZipInputStream, 1895
  - wxZipNotifier::OnEntryUpdated, 1897
  - wxZipOutputStream, 1898
  - wxZipOutputStream::~wxZipOutputStream, 1898
  - wxZipOutputStream::Close, 1898
  - wxZipOutputStream::CloseEntry, 1898
  - wxZipOutputStream::CopyArchiveMetaData, 1898
  - wxZipOutputStream::CopyEntry, 1898
  - wxZipOutputStream::Get/SetLevel, 1899
  - wxZipOutputStream::PutNextDirEntry, 1899
  - wxZipOutputStream::PutNextEntry, 1899
  - wxZipOutputStream::SetComment, 1899
  - wxZipOutputStream::wxZipOutputStream, 1897
  - wxZlibInputStream, 1900
  - wxZlibInputStream::CanHandleGZip, 1900
  - wxZlibInputStream::wxZlibInputStream, 1900
  - wxZlibOutputStream, 1901
  - wxZlibOutputStream::CanHandleGZip, 1902
  - wxZlibOutputStream::wxZlibOutputStream, 1901
- X—
- x, 1252
  - x, 1194
  - x, 1253
  - Xor, 1268, 1269
  - XRC C++ sample, 2108
  - XRC concepts, 2106
  - XRC file format, 2114
  - XRC resource file sample, 2111
  - XToCol, 768
  - XToEdgeOfCol, 768
-

XYToPosition, 1346, 1652

## **—Y—**

y, 1194, 1252

y, 1253

Year, 348

Years, 348

Yield, 56, 1678

YToEdgeOfRow, 768

YToRow, 769