# Manual for wxWindows 1.40: A portable GUI toolkit for C++

Julian Smart
Knowledge Based Decision Support Group
Artificial Intelligence Applications Institute
80 South Bridge
University of Edinburgh
EH1 1HN

April 1993

# Contents


**1 Introduction** **8**

1.1 What is wxWindows? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

1.2 Why another cross-platform development tool? . . . . . . . . . . . . . . . . . . . . . 8

1.3 wxWindows requirements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

1.4 Availability and location of wxWindows . . . . . . . . . . . . . . . . . . . . . . . . 10

1.5 Acknowledgments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

**2 Overview and comparison with other GUI models** **12**

2.1 Windows . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.1.1 MDI vs. SDI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.2 Dialog boxes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.3 Menus . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.4 Events . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.5 Repainting of windows . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.6 Scrolling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

2.7 Printing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

2.8 Programmatic versus interactive GUI building . . . . . . . . . . . . . . . . . . . . 16

2.9 Dimensions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

2.10 Colour . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

2.11 Online help . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

2.12 Interprocess Communication . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

2.12.1 What wxWindows has . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

2.12.2 Principles of DDE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

2.12.3 Data transfer . . . . . . . . . . 19

2.12.4 Examples . . . . . . . . . . 19

2.12.5 Remote Procedure Call . . . . . . . . . . 19

**3 Multi-platform development with wxWindows 20**

3.1 Include files and libraries . . . . . . . . . . 20

3.2 Makefiles . . . . . . . . . . 21

3.2.1 DOS makefiles . . . . . . . . . . 21

3.2.2 UNIX makefiles . . . . . . . . . . 21

3.3 Windows-specific files . . . . . . . . . . 21

3.3.1 Resource file . . . . . . . . . . 22

3.3.2 Module definition file . . . . . . . . . . 22

3.4 Memory models and memory allocation . . . . . . . . . . 22

3.5 Dynamic Link Libraries . . . . . . . . . . 23

3.6 Conditional compilation . . . . . . . . . . 23

3.7 Building on-line help . . . . . . . . . . 24

3.8 C++ issues . . . . . . . . . . 24

3.8.1 Templates . . . . . . . . . . 24

3.8.2 Definition of constructors . . . . . . . . . . 24

3.8.3 Pointers to functions . . . . . . . . . . 24

3.8.4 Precompiled headers . . . . . . . . . . 25

3.9 File handling . . . . . . . . . . 25

3.10 Large amounts of global data . . . . . . . . . . 26

**4 Bugs and future developments for wxWindows 27**

4.1 Bugs . . . . . . . . . . 27

4.2 Future developments . . . . . . . . . . 27

4.3 Troubleshooting . . . . . . . . . . 28

**5 Classes and data types 29**

5.1 Class hierarchy . . . . . . . . . . 29

5.2 wxApp: wxObject . . . 30
5.3 wxBrush: wxObject . . . 32
5.4 wxBrushList: wxList . . . 33
5.5 wxButton: wxItem . . . 34
5.6 wxCanvas: wxWindow . . . 34
5.7 wxCheckBox: wxItem . . . 39
5.8 wxChoice: wxItem . . . 40
5.9 wxClient: wxIPCObject . . . 41
5.10 wxColour: wxObject . . . 42
5.11 wxColourDatabase: wxObject . . . 42
5.12 wxConnection: wxObject . . . 43
5.13 wxCursor: wxObject . . . 46
5.14 wxDC: wxObject . . . 48
5.15 wxDialogBox: wxPanel . . . 54
5.16 wxEvent: wxObject . . . 56
5.17 wxFont: wxObject . . . 58
5.18 wxForm: wxObject . . . 59
5.18.1 The purpose of the form class . . . 59
5.18.2 Constraints on form items . . . 60
5.18.3 Form appearance . . . 60
5.18.4 Example . . . 61
5.18.5 Functions for making form items and constraints . . . 63
5.19 wxFrame: wxWindow . . . 65
5.20 wxFunction . . . 67
5.21 wxIcon: wxObject . . . 67
5.22 wxHashTable: wxObject . . . 68
5.23 wxHelpInstance: wxClient . . . 70
5.24 wxItem: wxWindow . . . 72
5.25 wxList: wxObject . . . 73

5.26 wxListBox: wxItem . . . . . . . . . . 76

5.27 wxMenu: wxItem . . . . . . . . . . 78

5.28 wxMenuBar: wxItem . . . . . . . . . . 79

5.29 wxMessage: wxItem . . . . . . . . . . 80

5.30 wxMetaFile: wxObject . . . . . . . . . . 81

5.31 wxMetaFileDC: wxDC . . . . . . . . . . 81

5.32 wxMultiText: wxText . . . . . . . . . . 82

5.33 wxNode: wxObject . . . . . . . . . . 82

5.34 wxObject . . . . . . . . . . 83

5.35 wxPanel: wxWindow . . . . . . . . . . 83

5.36 wxPathList: wxList . . . . . . . . . . 84

5.37 wxPen: wxObject . . . . . . . . . . 85

5.38 wxPenList: wxList . . . . . . . . . . 87

5.39 wxPoint: wxObject . . . . . . . . . . 88

5.40 wxServer: wxIPCObject . . . . . . . . . . 88

5.41 wxSlider: wxItem . . . . . . . . . . 89

5.42 wxStringList: wxList . . . . . . . . . . 89

5.43 wxText: wxWindow . . . . . . . . . . 90

5.44 wxTextWindow: wxWindow . . . . . . . . . . 91

5.45 wxTimer: wxObject . . . . . . . . . . 93

5.46 wxWindow: wxObject . . . . . . . . . . 94

**6 Miscellaneous functions** **99**

6.1 File functions . . . . . . . . . . 99

6.2 String functions . . . . . . . . . . 101

6.3 Dialog functions . . . . . . . . . . 101

6.4 GDI functions . . . . . . . . . . 103

6.5 Miscellaneous . . . . . . . . . . 103

**Glossary** **105**

**Index** **106**

## Copyright notice

# Chapter 1

# Introduction

## 1.1 What is wxWindows?

wxWindows is a class library for C++ providing GUI (Graphical User Interface) and other facilities on more than one platform. It currently supports subsets of Open Look (XView), Motif and Windows 3.1. It was originally developed at the Artificial Intelligence Applications Institute, University of Edinburgh, for internal use on a medium-sized project[1]. wxWindows has been released into the public domain in the hope that others will also find it useful.

This manual describes in detail the wxWindows version 1.40 API (Application Programmer's Interface). There is also a smaller, tutorial document which discusses various aspects of wxWindows use.

## 1.2 Why another cross-platform development tool?

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

1. low price
2. source availability
3. ability to use Open Look on the Sun instead of or as well as Motif
4. simplicity of programming
5. support for GCC (GNU C++)
6. support for interprocess communication

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

[1] A hypertext-based knowledge-acquisition and diagramming tool called HARDY.

wxWindows currently maps to three native APIs: XView or Motif on the Sun (and other platforms supporting XView or Motif) and Windows 3.1 and above on the PC. This covers a very large proportion of machines in use today: the Sun being an extremely popular workstation, and the PC being the most popular personal and business computer. However, there is nothing to prevent versions of the class library being written for the Macintosh and other environments. A Windows NT version is currently under development.

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

Choice of C++ compiler is currently important for wxWindows (see **Requirements**, below). On the PC, the tested compilers for wxWindows are Microsoft C/C++ Version 7 and (to a lesser extent) Borland C++ 3.1, and currently the makefiles are for MS C7 only.

On the Sun, GNU C++ (GCC) and AT&T C++ are known to work with wxWindows.

The importance of using a platform-independent class library cannot be overstated, since GUI application development is very time-consuming, and sustained popularity of particular GUIs cannot be guaranteed. Code can very quickly become obsolete if it addresses the wrong platform or audience. wxWindows helps to insulate the programmer from these winds of change. Although wxWindows may not be suitable for every application, it provides access to most of the functionality a GUI program normally requires, plus some extras such as form construction, interprocess communication and PostScript output, and can of course be extended as needs dictate. As a bonus, it provides an arguably cleaner interface to XView, Motif and Windows 3 than the native APIs. Programmers may find it worthwhile to use wxWindows even if they are developing on only one platform.

When Windows NT becomes readily available, it should require very little work to port wxWindows to it, and then wxWindows applications should work on a much broader range of hardware (albeit using a subset of NT's facilities).

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

- Low cost (free, in fact!)
- You get the source
- Several example programs, reasonable documentation
- Simple-to-use, object-oriented API
- No more messing with arcane X window calls under XView or Motif
- Graphics calls include splines, polylines, rounded rectangles, etc.
- XView-style panel item layout
- Status line facility
- Easy, object-oriented interprocess comms (DDE subset) under UNIX and Windows 3
- Encapsulated PostScript generation under UNIX, normal Windows 3 printing on the PC
- Virtually automatic MDI support under Windows

- Can be used to create DLLs under Windows, dynamic libraries on the Sun
- Support for Windows 3 printer and file common dialogs, with equivalents for UNIX
- Under Windows 3, support for creating metafiles and copying them to the clipboard.
- Programmatic form facility for building form-like screens fast, with constraints on values
- Hypertext help facility, with an API for invocation from applications

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

- No commercial support (I or the net might help, though)
- Restricted to large or huge models under Windows 3
- Limited bitmap support
- XView-style separation of frames, panels, canvases and text
- Not many panel items (widgets/controls) yet

## 1.3 wxWindows requirements

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

(a) PC:

1. a 386SX or higher PC running Windows 3.1
2. Microsoft C/C++ version 7 (including the SDK) or Borland C++ version 3.1 (other compilers may work)

(b) UNIX:

1. GNU C++ version 2.1 or later, or compatible compiler (such as AT&T C++)
2. a Sun or other workstation supporting GNU C++ and either XView 3.1 or Motif 1.1

Note that these are the only tested tools; others may work. wxWindows takes up between about two and five megabytes on both platforms, depending on whether wxWindows is being recompiled or whether the library file and header files are sufficient.

## 1.4 Availability and location of wxWindows

wxWindows is currently available from the Artificial Intelligence Applications Institute by anonymous FTP. FTP to skye.aiai.ed.ac.uk, log on as 'anonymous', and give your user ID as password. Change directory into pub/wxwin, change transmission method to binary, and get wxwin130.tar.Z

(UNIX) and/or wxwin130.zip (PC). Both archives contain the source code for both platforms, but only the executable demos for the specific platform (UNIX and Windows respectively).

wxWindows 1.40 is expected to be uploaded to the CICA Windows 3 archive at Indiana University (ftp.cica.indiana.edu). Version 1.20 is available from the British HENSA public domain software archive at pdsoft.lancs.ac.uk in the directory x/a/a103.

It is also available by post from:

Dr Julian Smart, Artificial Intelligence Applications Institute, University of Edinburgh, 80 South Bridge, Edinburgh, Scotland, EH1 1HN. Email jacs@aiai.ed.ac.uk, tel. 031-650-2746.

It is distributed in tar and/or zip format on a high-density 3.5" MS-DOS disk, with online PostScript, LaTeX and DVI format documentation. We regret we cannot supply printed documentation. Please supply your own disk and return postage, or enclose something to cover distribution costs.

## 1.5 Acknowledgments

Thanks are due to the AIAI for being willing to release wxWindows into the public domain, and to my wife Harriet Smart for her patience while I finished off wxWindows at weekends.

The UseNet has been an essential prop when coming up against tricky XView and Windows problems, so thanks to those who answered my queries, in particular Kari, Jamshid Afshar, Josep Fortiana, Ian Perrigo, Neil Smith and Robin Corbet.

I also acknowledge the author of xfig, the excellent UNIX drawing tool, from the source of which I have pinched some spline drawing code and a few cursor definitions. His copyright is included below.

# Chapter 2

# Overview and comparison with other GUI models

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

## 2.1 Windows

An application presents the user primarily with a series of windows. A window can be made up of a *frame* with one or more subwindows. This is like the XView model, rather than Motif or Windows 3 where child windows may be nested to any depth. The frame method was chosen since XView required it, and because it is a useful simplifying assumption, imposing few restrictions in practise. Note that the splitting of views beloved of the Open Look standard is not allowed in wxWindows.

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

In Xlib and Windows 3.1, canvases need to be painted using a handle to a 'device context'. The purpose of this is to enable more than one device context to be used for painting a given window, and in Windows 3 to enable devices to include printers and other forms of display. wxWindows also has the notion of a device context, but all drawing commands may also be directly issued to a canvas for convenience.

wxWindows defines several objects required for drawing on canvases: colours, fonts, pens and brushes. Neither XView nor Motif provides pens and brushes, which in Windows 3 allow the selection of different predefined 'drawing tools' with certain characteristics (line thickness, colour, fill style etc.). In X there are a limited range of fonts, while in Windows 3 an infinite selection of sizes is available thanks to the scaleable TrueType font system. Under XView, wxWindows chooses the closest matching font.

Windows 3 or Motif programmers may miss the ability to place panel items in canvases (to use XView terminology), or to draw graphics in panels. A future version of wxWindows may support drawing on panels, particularly in the Motif version, lifting these restrictions. Support for displaying panel items and drawing graphics will probably be moved up to the base **wxWindow** class, maintaining compatibility with existing wxWindows programs but allowing new programs more flexibility.

### 2.1.1 MDI vs. SDI

In Windows 3, a popular technique is to use the MDI (Multiple Document Interface) style, where the application window has a number of iconizable document windows which fit within it. This can save clutter on the desktop since iconizing or moving the parent window iconizes or moves all the child windows. MDI contrasts with SDI (Single Document Interface) in which windows are not constrained within one parent window, and normal practice is to run one instance of the application per document. Since wxWindows uses the large model, and large model programs may be limited to one instance at a time (unless only one data segment is used), it makes sense to offer MDI support.

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

- Since it is likely that wxWindows programmers are writing for multiple platforms, and only Windows 3 supports MDI, they will not wish to spend much or any time implementing MDI features.
- The difference between writing MDI and SDI applications has been reduced to almost zero so even the beginner can easily produce MDI programs.
- What wxWindows's auto-MDI cannot handle is hardly worth the extra effort anyway!

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

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

## 2.2 Dialog boxes

A *dialog box* is like a panel, with an implicit frame surrounding it. A dialog box may be *modal* (no other window in this application is active and the calling program flow is suspended) or *modeless* (any window may be interacted with and control returns immediately to the program). With dialog boxes, creation of separate frames and panels is not necessary, and under Windows 3, additional functionality is added 'for free', such as tabbing between items. Any panel item may be attached to a dialog box since wxDialogBox is derived from the wxPanel.

Note that under 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 receiving a dialog initialization message. This is too restrictive for wxWindows, where any window may be created and possibly displayed before its contents are created.

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

## 2.3 Menus

Menus are mainly used in menu bars, but could also be used for popup menus on panels (not yet implemented). A menu bar is a sequence of pull-down command menus near the top of the window, usually with a **File** menu as the first menu. In Windows 3 and Motif, the menu bar is a standard user interface component. Under XView, wxWindows must simulate a menu bar with a series of menu buttons. Under wxWindows for Motif and Windows 3, but not XView, placing and ampersand before a letter in a menu name causes it to be underscored and interpreted as a kind of hotkey. Under XView such underscores are ignored. The Motif version of wxWindows automatically right-justifies the help menu, if there is one.

Menu items are identified by integer identifiers, and when a menu item is selected, the parent frame is notified using the **OnMenuCommand** member.

## 2.4 Events

In XView and Motif, events (such as resizing, painting, mouse clicks, button presses) are handled by a rather arbitrary collection of optional XView and Xlib callbacks. In Windows 3, events are *messages* which are handled by a window procedure, or ignored.

In wxWindows, GUI objects mostly receive events by wxWindows calling user-overridable handlers, such as **OnEvent**, **OnChar** and **OnSize**. Frames can receive **OnClose** events from the window manager (X) or system control menu (Windows 3). These events are handled by the application by deriving new window classes and overriding the event-handling member functions.

Panel item notification has to be handled rather differently. In Windows 3, all panel item events (such as a button press) are sent to the parent window (requiring large case statements to differentiate the events), and all window events are sent to the relevant window. In XView and Motif, different types of callback function have to be defined, depending on the event.

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

## 2.5 Repainting of windows

All windows except for the canvas are repainted by the system. The canvas requires a paint message handler to be defined (and therefore canvas derivation is obligatory). Under XView and

Motif, wxWindows allows a canvas to be retained if desired, which means that fewer paint messages are received and scrolling is fast. The Motif implementation has a slight overhead in that drawing must be done both to the canvas and to the backing pixmap, but this is usually made up for by the speed of repainting.

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

See below for more information on repainting and scrolling.

## 2.6 Scrolling

Scrolling can be a tricky subject for a novice GUI programmer to tackle. The Windows 3 API ensures that the suffering is as acute as possible by requiring the application to program the scrollbar behaviour and to check the scrollbar positions on repainting. XView has the decency to provide a canvas event with coordinates that reflect what the scrollbars are doing. In wxWindows, the XView approach is taken, so that all the programmer has to do is to create scrollbars with a given scroll length and increment, and the repaint procedure automatically reflects scrollbar positions. Obviously this simple approach can be inefficient if the canvas doesn't know what is actually in view, so the application can get the current view in order to limit the amount of repainting required.

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

Under Windows 3.1, wxWindows scrolls the bits of the canvas around when the user generates a scroll event, so the canvas does not need to be cleared and only the damaged areas need be repainted—so all the application need do is redraw the whole image, and scrolling will appear to be relatively smooth. This is the case in the `hello.exe` demo.

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

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

Scrolling panels have yet to be implemented in wxWindows.

## 2.7 Printing

Printing in Windows 3 is relatively easy, since all drawing is done to a 'device context' which could equally well be associated with a printer as with a window. Windows 3 handles the plethora of printer types that abound in the PC environment. In X under UNIX, the standard is PostScript; X, XView and Motif provide no help at all. The solution adopted in wxWindow is to use a device context for canvases and printers, with Windows 3 printing supported on the PC and an Encapsulated PostScript driver provided under X. Thus graphic code may be extremely generic - the same piece of code can draw to Windows 3 screens of all types, to X windows, and to hundreds of different printers.

## 2.8 Programmatic versus interactive GUI building

Interactive tools for rapidly building GUIs are all the rage, and wxWindows does not at this point support this. Loading Windows 3 dialog boxes from resource files will be possible in future in wxWindows, so that dialogs may be constructed interactively. However, this will not carry through to other platforms, so a more general solution is required. One possibility is to translate Windows 3 resource scripts or DevGuide descriptions into wxWindows code, or into a form that wxWindows could load. Using YACC and LEX such a project could be quite straightforward, although there will not always be a mapping between the dialog builder and wxWindows constructs.

Another consideration is that it is not always possible to build GUI components interactively: the 'what you see is *all* you get' syndrome. When complex repositioning of items depending on window size is required, then GUI builders such as the Microsoft dialog editor are useless.

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

## 2.9 Dimensions

The graphics origin is always the top left hand corner of a window. Dimensions are a problem in a multi-platform application, since display and character widths will change from machine to machine, even more so than for different PC display boards. At the moment wxWindows uses pixels; Windows 3 tackles the problem by using 'dialog units' based on the size of the standard system font. To avoid this problem when creating panel items, wxWindows provides automatic left to right, top to bottom item layout (similar to XView), in addition to absolute positioning, in which case, portability is up to the discretion of the programmer.

A canvas has a *mapping mode* associated with it, which determines the meaning of dimensions in subsequent graphics operations. Drawing may be done using various units including mm, 1/10 mm, pixels and points. Future versions of wxWindows will support a change of graphics origin.

## 2.10 Colour

Under XView and Motif, the default palette is used for canvases, giving a limited range of colours. More colours show up under Windows 3. Future versions of wxWindows should improve upon colour and palette handling.

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

## 2.11 Online help

Most modern GUI applications have on-line hypertext help. In Windows 3, help is normally supplied in binary files which are read by an external program, which is itself accessed from within the application using Microsoft-supplied function calls. XView has provisions for simple context sensitive help, but no equivalent of the Windows 3 browseable help.

From version 1.30, wxWindows comes with wxHelp, a hypertext help system which may be invoked from wxWindows applications. wxHelp works in two modes—edit and end-user. In edit mode, an ASCII file may be marked up with different fonts and colours, and divided into sections. In end-user mode, no editing is possible, and the user browses principally by clicking on highlighted blocks.

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

## 2.12 Interprocess Communication

### 2.12.1 What wxWindows has

Interprocess communication (IPC) has always been a tricky area, and the plethora of techniques on different platforms has not helped. Microsoft has laid down several standards for IPC under Windows 3, the most fundamental being Dynamic Data Exchange (DDE). DDE is the basis for wxWindows's IPC capability: the same, simple, object-oriented interface is provided for a subset of DDE under both Windows on the PC, and under XView and Motif on UNIX. The UNIX version is implemented using sockets, and allows processes on the same or different machines to talk to each other.

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

### 2.12.2 Principles of DDE

The following describes how wxWindows implements DDE. The following three classes are crucial.

1. wxClient. This represents the client application, and is used only within a client program.
2. wxServer. This represents the server application, and is used only within a server program.

3. wxConnection. This represents the connection from the current client or server to the other application (server or client), and can be used in both server and client programs. Most DDE transactions operate on this object.

Messages between applications are usually identified by three variables: connection object, topic name and item name. A data string is a fourth element of some messages. To create a connection (a conversation in Windows parlance), the client application sends the message **MakeConnection** to the client object, with a string service name to identify the server and a topic name to identify the topic for the duration of the connection. Under UNIX, the service name must contain an integer port identifier.

The server then responds and either vetos the connection or allows it. If allowed, a connection object is created which persists until the connection is closed. The connection object is then used for subsequent messages between client and server.

To create a working server, the programmer must:

1. Derive a class from wxServer.
2. Override the handler **OnAcceptConnection** for accepting or rejecting a connection, on the basis of the topic argument. This member must create and return a connection object if the connection is accepted.
3. Create an instance of your server object, and call **Create** to activate it, giving it a service name.
4. Derive a class from wxConnection.
5. Provide handlers for various messages that are sent to the server side of a wxConnection.

To create a working client, the programmer must:

1. Derive a class from wxClient.
2. Override the handler **OnMakeConnection** to create and return an appropriate connection object.
3. Create an instance of your client object.
4. Derive a class from wxConnection.
5. Provide handlers for various messages that are sent to the client side of a wxConnection.
6. When appropriate, create a new connection by sending a **MakeConnection** message to the client object, with arguments host name (processed in UNIX only), service name, and topic name for this connection. The client object will call **OnMakeConnection** to create a connection object of the desired type.
7. Use the wxConnection member functions to send messages to the server.

### 2.12.3 Data transfer

These are the ways that data can be transferred from one application to another.

- **Execute:** the client calls the server with a data string representing a command to be executed. This succeeds or fails, depending on the server's willingness to answer. If the client wants to find the result of the Execute command other than success or failure, it has to explicitly call Request.
- **Request:** the client asks the server for a particular data string associated with a given item string. If the server is unwilling to reply, the return value is NULL. Otherwise, the return value is a string (actually a pointer to the connection buffer, so it should not be deallocated by the application).
- **Poke:** The client sends a data string associated with an item string directly to the server. This succeeds or fails.
- **Advise:** The client asks to be advised of any change in data associated with a particular item. If the server agrees, the server will send an OnAdvise message to the client along with the item and data.

The default data type is wxCF_TEXT (ASCII text), and the default data size is the length of the null-terminated string. Windows-specific data types could also be used on the PC.

### 2.12.4 Examples

See the sample programs *server* and *client* in the IPC samples directory. Run the server, then the client. This demonstrates using the Execute, Request, and Poke commands from the client, together with an Advise loop: selecting an item in the server list box causes that item to be highlighted in the client list box.

See also the source for wxHelp, which is a DDE server, and the files wx_help.h and wx_help.cc which implement the client interface to wxHelp.

### 2.12.5 Remote Procedure Call

DDE is quite a low level protocol, and all encoding and decoding of messages must be done by the client and server applications. The wxWindows extension *PROLOGIO* implements a remote procedure call protocol (RPC) so that a server can implement a library of functions for a client to call. PROLOGIO makes it easy for applications to pack and unpack the arguments and return value(s) of procedure calls, and provides a mechanism for the server to register its available calls and automatically handle the routing of calls to appropriate server callbacks, one to a procedure definition. All this makes calling or implementing server facilities childishly simple. Since PROLOGIO sits on top of the DDE wrapper, it is also platform independent. See the PROLOGIO manual.

PROLOGIO is expected to be available in the first quarter of 1993.

# Chapter 3

# Multi-platform development with wxWindows

## 3.1 Include files and libraries

Under UNIX, use the library libwx.a. Under DOS, use the library wx.lib.

Under both operating systems, include the file <wx.h>, or to save on compilation time, include only those header files relevant to the source file. The header files are as follows.

- common.h - important definitions and includes
- wx.h - includes all wxWindows headers
- wx_canvs.h - drawing on canvases
- wx_dc.h - drawing on canvases and printers
- wx_dialg.h - user-defined and standard dialogs
- wx_event.h - used in conjunction with windows and panel items
- wx_form.h - forms allow succinct expression of panels with constrained items
- wx_frame.h - frame window handling
- wx_gdi.h - graphics objects such as colours, pens, brushes and icons
- wx_help.h - API for invoking wxHelp
- wx_hash.h - hash table implementation
- wx_ipc.h - interprocess communication (using a subset of the DDE standard)
- wx_item.h - panel items
- wx_list.h - linked list implementation, heavily used by wxWindows
- wx_main.h - application initialization
- wx_obj.h - base class for wxWindows classes

- wx_panel.h - panels
- wx_privt.h - some private wxWindows declarations
- wx_text.h - text subwindows
- wx_timer.h - timer object
- wx_utils.h - various utilities
- wx_win.h - abstract base class for frames, subwindows and items

Unfortunately, including specific header files for purposes of compiling speed can be a problem on the PC if **wx.h** is precompiled, since then **wx.h** must be mentioned in each source (see below). A tradeoff is probably necessary between speed of compiling under UNIX versus on the PC.

## 3.2 Makefiles

At the moment there is no attempt to make UNIX makefiles and nmake makefiles compatible, i.e. one makefile is required for each environment. If anyone knows how to avoid this duplication, I would be interested!

Sample makefiles are included with the library and demos, but not for Borland C++, so if anyone can send me some Borland makefiles I will make them available with a future release.

### 3.2.1 DOS makefiles

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

### 3.2.2 UNIX makefiles

All makefiles have the targets **xview** and **motif**, which build versions of a library or binary appropriate to desired GUI toolkit. To change from one toolkit to another you must remember to recompile anything using wxWindows, for example by using the **wxclean** target. Debugging information is usually included; you may add DEBUG= on the command line to compile without it, or use the UNIX **strip** command to remove debugging information (faster and more thorough).

## 3.3 Windows-specific files

wxWindows application compilation under Windows 3 requires at least two extra files, resource and module definition files.

### 3.3.1 Resource file

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

```
rcinclude wx.rc
```

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

```
wxicon icon wx.ico
```

The icon can then be referenced by name when creating a frame icon. See the Windows 3 SDK documentation.

### 3.3.2 Module definition file

A module definition file (extension DEF) looks like the following:

```
; hello.def
;

NAME         Hello
DESCRIPTION  'Hello'

EXETYPE      WINDOWS
STUB         'WINSTUB.EXE'

CODE         PRELOAD MOVEABLE DISCARDABLE
DATA         PRELOAD MOVEABLE MULTIPLE

HEAPSIZE     1024
STACKSIZE    8192
```

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

## 3.4 Memory models and memory allocation

Under UNIX, this isn't a problem. Under Windows, the only really viable way to go is to use the large model, which uses the global heap instead of the local heap for memory allocation. Unless more than one read-write data segment is used (see section 3.10 below), large model programs may still have multiple instances under MS C/C++ 7. Microsoft give the following guidelines for producing multiple-instance large model programs:

- Do not use `/ND` to name extra data segments unless the segment is READONLY.
- Use the .DEF file to mark extra data segments READONLY.
- Do not use __far or FAR to mark data items.

- Use `/PACKDATA` to combine data segments.
- Use `/Gt65500 /Gx` to force all data into the default data segment.

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

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

## 3.5 Dynamic Link Libraries

wxWindows may be used to produce DLLs which run under Windows 3.1. Note that this is not the same thing as having wxWindows as a DLL, which is not currently possible. For MSC 7, use the makefile with the argument DLL=1 to produce a version of the wxWindows library which may be used in a DLL application. There is a bug in MSC 7 which makes the compiler complain about returned floats, which goes away when the `/Os` option is used, which is why that flag is set in the makefile.

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

## 3.6 Conditional compilation

One of the purposes of wxWindows is to reduce the need for conditional compilation in source code, which can be messy and confusing to follow. However, sometimes it is necessary to incorporate platform-specific features (such as metafile use under Windows 3.1). The following identifers may be used for this purpose, along with any user-supplied ones:

- `wx_x` - for code which should work under any X toolkit
- `wx_xview` - for code which should work under XView only
- `wx_motif` - for code which should work under Motif only
- `wx_msw` - for code which should work under Microsoft Windows only

For example:

```
  ...
#ifdef wx_x
  (void)wxMessageBox("Sorry, metafiles not available under X.");
#endif
#ifdef wx_msw
  wxMetaFileDC dc;
  DrawIt(dc);
```

```
  wxMetaFile *mf = dc.Close();
  mf->SetClipboard();
  delete mf;
#endif
  ...
```

## 3.7 Building on-line help

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

## 3.8 C++ issues

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

### 3.8.1 Templates

I have not investigated using the cut-down template facility in MSC 7; probably it is best to avoid using templates for maximum portability.

### 3.8.2 Definition of constructors

GCC allows the user to omit constructor definitions where a parent class provides a constructor with parameters. In MSC 7, all constructors with parameters must be defined in the derived class, or the compiler cannot find the required constructor. This may mean defining dummy constructors which call parent constructors, for example:

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

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

### 3.8.3 Pointers to functions

GCC is clever in its matching of function pointer arguments to the declaration of the function, and will not complain in the following case:

```
typedef void (*wxFunction) (wxObject&, wxEvent&);
```

```
...
void mycallback(wxButton& button, wxEvent& event);
...
wxButton button(parent, &mycallback, label, 100, 200);
```

Since wxButton is derived from wxObject, the function *mycallback* is a subtype of **wxFunction**. In MSC 7, however, the function is not an exact match, and the compiler complains. The solution is to place a cast in front of the function address, thus:

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

### 3.8.4 Precompiled headers

MSC 7 supports precompiled headers, which can save a great deal of compiling time. However, if the header file to be compiled only appears in the DOS environment (for example `windows.h`), there is a problem. The filename must appear in any source file including this header file, not just in another header file; therefore any application using `wx.h` and requiring precompilation of `windows.h` must include a reference to `windows.h` in all the source files. It is not possible to conditionally include this reference due to another MSC 7 restriction, so the only solution seems to be to have a dummy `windows.h` in the UNIX environment.

If `wx.h` is itself precompiled, this problem should go away assuming all source files include `wx.h`. Precompilation is useful only for stable code and so precompiling the wxWindows headers is not practical if wxWindows itself is being altered.

## 3.9 File handling

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

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

See also section 6.1 for descriptions of miscellaneous file handling functions.

## 3.10 Large amounts of global data

Under Windows, it is possible that the default data segment becomes too large (for example, a large number of small, global data items have been declared). This may be cured by using more than one data segment. In MSC7, specify the `/Gt` compiler option with a number representing the data size threshold for putting data items in a separate segment. For example, `/Gt8`.

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

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

# Chapter 4

# Bugs and future developments for wxWindows

## 4.1 Bugs

These are the known bugs:

1. Using **SetClientSize** before setting the menu bar on a frame does not cause a menu bar to appear under X. Use **SetClientSize** after changes to the window which might affect its size.
2. Modal dialogs under XView do not treat some panel items correctly. This XView problem is partially corrected with the list item scrollbar fix supplied with wxWindows, but choice items (and probably other items) still don't work properly.
3. MDI support is hampered by the inability to paint icons dynamically onto the iconized window client area, thus limiting icons to one image for all child windows.
4. Mapping mode support is patchy.
5. PostScript output control could be improved. Mapping mode ignored by PostScript driver.
6. Some text window behaviour is not implemented in Windows (e.g. character input).
7. Support for panel items such as radio buttons is absent.
8. On creation of a new frame, XView often appears to send too many repaint events to subwindows. I can't find a way around this, any suggestions appreciated.

## 4.2 Future developments

The following enhancements are under consideration, some of which have been discussed earlier.

1. Ports to other windowing systems, such as Motif and NT. These are both under way.

2. Better support for bitmap operations and image file loading.
3. a wxToolBar class.
4. interception of all wxWindows system events, for 'meta-applications' such as a special-needs interface.
5. More panel item types.
6. Changeable panel item fonts.
7. An enhanced text window for the Windows environment
8. Use of a tool such as DevGuide to generate wxWindows code.

## 4.3 Troubleshooting

*Under Windows, dialog boxes refuse to appear.*

You probably forgot to include the file wx.rc in your resource script (see Section 3.3.1).

*Under X, the menu bar does not appear, and a blank area of window appears instead.*

You may be using **SetClientSize** before the menu bar has been created. Call it after the whole window and its subwindows have been created.

*Under XView, the following message appears:*

`XView warning: SERVER_IMAGE_BITMAP_FILE: Server image creation failed (Server Image package)`

The application may be looking for an icon file which does not exist or has been referenced by a relative icon pathname. Use an absolute path, or include the icon image in the source file using conditional compilation (see the **wxIcon** class entry).

*The program exits abnormally without initializing.*

You may be declaring a pen, brush, icon, cursor or colour globally. These objects automatically add themselves to global lists which may not be initialized before the object constructors are called, and so only global pointers to these objects may be declared. After or during **OnInit** is called, these objects may be created with impunity.

*Under Windows, I get a link error.*

See sections 3.4 and 3.10. Also, try invoking the MSC7 help compiler with the given error message.

# Chapter 5

# Classes and data types

## 5.1 Class hierarchy

The GUI-specific wxWindows class hierarchy is shown in Figure 5.1. Many other, non-GUI classes have been omitted.

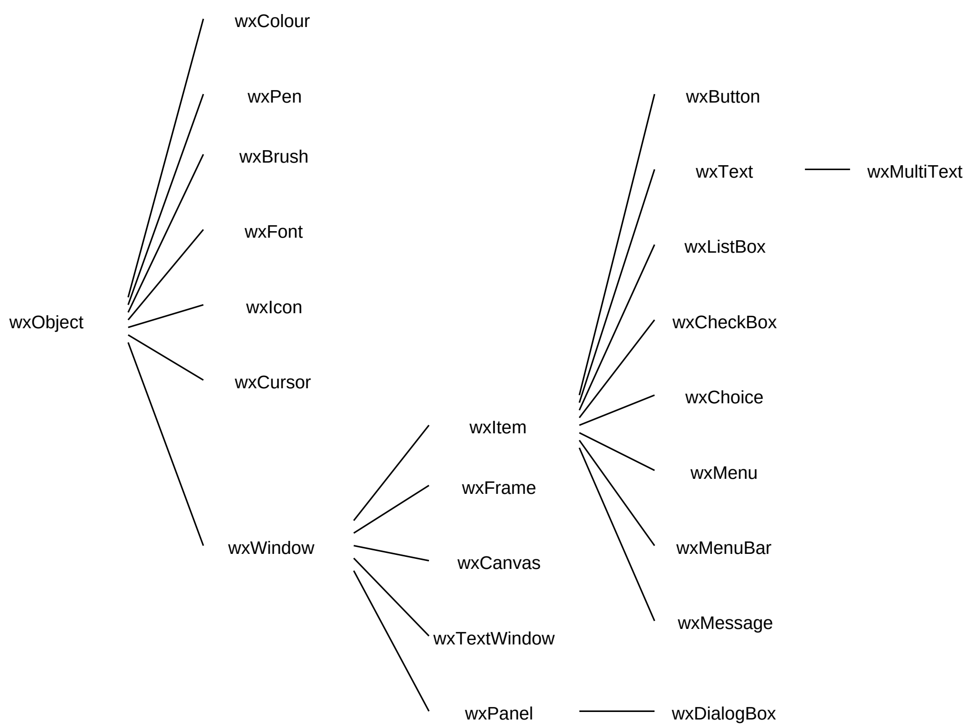


Figure 5.1: wxWindows class hierarchy

In the following descriptions of the wxWindows classes and their member functions, note that descriptions of inherited member functions are not duplicated in derived classes unless their behaviour is different. So in using a class such as wxCanvas, be aware that wxWindow functions may be relevant.

Note also that arguments with default values may be omitted from a function call, for brevity. Size and position arguments may usually be given a value of -1 (the default), in which case wxWindows will choose a suitable value.

The member functions are given in alphabetical order except for the constructors and destructors which appear first.

## 5.2 wxApp: wxObject

The **wxApp** class represents the application itself. A wxWindows application does not have a *main* procedure; the equivalent is the *OnInit* member defined for a class derived from **wxApp**. *OnInit* must create and return a main window frame as a bare minimum. If NULL is returned from *OnInit*, the application will exit. Note that the program's command line arguments, represented by *argc* and *argv*, are available from within **wxApp** member functions.

An application closes by destroying all windows. Because all frames must be destroyed for the application to exit, it is advisable to use parent frames wherever possible when creating new frames, so that deleting the top level frame will automatically delete child frames. In emergencies the *wxExit* function can be called to kill the application.

An example of defining an application follows:

```
class DerivedApp: public wxApp
{
 public:
  wxFrame *OnInit(void);
};

wxFrame *DerivedApp::OnInit(void)
{
  wxFrame *the_frame = new wxFrame(argv[0]);
  ...
  return the_frame;
}

MyApp DerivedApp;
```

## wxApp::wxApp

**void wxApp(void)**

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

## wxApp::~wxApp

**void ~wxApp(void)**

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

## wxApp::argc

**int** *argc*

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

## wxApp::argv

**char **** *argv*

Command line arguments (after environment-specific processing).

## wxApp::Initialized

**Bool Initialized**(**void**)

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

## wxApp::MainLoop

**void MainLoop**(**void**)

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

## wxApp::OnExit

**void OnExit**(**void**)

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

## wxApp::OnInit

**wxFrame * OnInit**(**void**)

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

## 5.3 wxBrush: wxObject

A brush is a drawing tool for filling in areas. It is used for painting the background of rectangles, ellipses, etc. It has a colour and a style - the style may be wxSOLID (normal) or wxTRANSPARENT (the brush isn't used). On a monochrome display, the default behaviour is to show all brushes as white. If you wish the policy to be 'all non-white colours are black', as with pens, uncomment the piece of code documented in **SetBrush** in wx_dc.cc. Alternatively, set the **Colour** member of the device context to TRUE, and select appropriate colours.

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

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

### wxBrush::wxBrush

**void wxBrush**(**void**)

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

**void wxBrush**(**char ***colour_name*, **int** *style*)

Constructs a brush: uninitialized, initialized with an RGB colour and a style, or initialized using a colour name and a style. If the named colour form is used, an appropriate **wxColour** structure is found in the colour database.

### wxBrush::~wxBrush

**void ~wxBrush**(**void**)

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

### wxBrush::GetColour

**wxColour& GetColour**(**void**)

Returns a reference to the brush colour.

### wxBrush::GetStyle

**int GetStyle**(**void**)

Returns the brush style (wxSOLID or wxTRANSPARENT).

## wxBrush::SetColour

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

**void SetColour(char ***colour_name***)**

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

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

## wxBrush::SetStyle

**void SetStyle(int** *style***)**

Sets the brush style (wxSOLID or wxTRANSPARENT).

# 5.4 wxBrushList: wxList

A brush list is a list containing all brushes which have been created. There is only one instance of this class: **wxTheBrushList**. Use this object to search for a previously created brush of the desired type and create it if not already found. In some windowing systems, the brush may be a scarce resource, so it is best to reuse old resources if possible. When an application finishes, all brushes will be deleted and their resources freed, eliminating the possibility of 'memory leaks'.

## wxBrushList::wxBrushList

**void wxBrushList(void)**

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

## wxBrushList::AddBrush

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

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

## wxBrushList::FindOrCreateBrush

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

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

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

## wxBrushList::RemoveBrush

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

Used by wxWindows to remove a brush from the list.

## 5.5 wxButton: wxItem

### wxButton::wxButton

**void wxButton(wxPanel** **parent*, **wxFunction** *func*, **char** **label*,
**int** *x = -1*, **int** *y = -1*, **int** *width = -1*, **int** *height = -1*)

Constructor, creating and showing a button. If *width* or *height* are omitted (or are less than zero), an appropriate size will be used for the button. *func* may be NULL; otherwise it is used as the callback for the button. Note that the cast (wxFunction) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration.

### wxButton::~wxButton

**void ~wxButton(void)**

Destructor, destroying the button.

## 5.6 wxCanvas: wxWindow

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

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

### wxCanvas::wxCanvas

**void wxCanvas(wxFrame** **parent*, **int** *x = -1*, **int** *y = -1*, **int** *width = -1*, **int** *height = -1*,
**int** *style = wxRETAINED*)

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

- wxBORDER, to give the canvas a thin border (Windows 3 only)
- wxRETAINED, to give the canvas a backing store, making repainting much faster but at a memory premium (XView and Motif only)

### wxCanvas::~wxCanvas

**void ~wxCanvas(void)**

Destructor.

### wxCanvas::Clear

**void Clear**(**void**)

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

# wxCanvas::DestroyClippingRegion

**void DestroyClippingRegion**(**void**)

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

# wxCanvas::DrawEllipse

**void DrawEllipse**(**float** *x*, **float** *y*, **float** *width*, **float** *height*)

Draws an ellipse contained in the rectangle with the given top left corner, and with the given size. The current pen is used for the outline and the current brush for filling the shape.

# wxCanvas::DrawLine

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

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

# wxCanvas::DrawLines

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

**void DrawLines**(**wxList** ***points*, **float** *xoffset = 0*, **float** *yoffset = 0*)

Draw lines using an array of *points* of size *n*, or list of pointers to points, adding the optional offset coordinate. The current pen is used for drawing the lines. The programmer is responsible for deleting the list of points.

# wxCanvas::DrawPolygon

**void DrawPolygon**(**int** *n*, **wxPoint** *points[]*, **float** *xoffset = 0*, **float** *yoffset = 0*)

**void DrawPolygon**(**wxList** ***points*, **float** *xoffset = 0*, **float** *yoffset = 0*)

Draw a filled polygon 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 outline, and the current brush for filling the shape. Using a transparent brush suppresses filling. The programmer is responsible for deleting the list of points.

Note that wxWindows does not close the first and last points automatically.

# wxCanvas::DrawPoint

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

Draws a point using the current pen.

## wxCanvas::DrawRectangle

**void DrawRectangle**(**float** *x*, **float** *y*, **float** *width*, **float** *height*)

Draws a rectangle with the given top left corner, and with the given size. The current pen is used for the outline and the current brush for filling the shape.

## wxCanvas::DrawRoundedRectangle

**void DrawRoundedRectangle**(**float** *x*, **float** *y*, **float** *width*, **float** *height*, **float** *radius = 20*)

Draws a rectangle with the given top left corner, and with the given size. The corners are quarter-circles using the given radius. The current pen is used for the outline and the current brush for filling the shape.

## wxCanvas::DrawSpline

**void DrawSpline**(**wxList** **points*)

Draws a spline between all given control points, using the current pen. Doesn't delete the wxList and contents. The spline is drawn using a series of lines, using an algorithm taken from the X drawing program 'XFIG'.

**void DrawSpline**(**float** *x1*, **float** *y1*, **float** *x2*, **float** *y2*, **float** *x3*, **float** *y3*)

Draws a three-point spline using the current pen.

## wxCanvas::DrawText

**void DrawText**(**char** **text*, **float** *x*, **float** *y*)

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

## wxCanvas::GetDC

**wxDC * GetDC**(**void**)

Get a pointer to the canvas's device context. Note that the canvas's apparent device context may be temporarily replaced by the application using *SetContext*, so for example a printer context can be substituted and programs using the canvas's device context will really write to the printer. The canvas's *real* device context is unaffected by *SetContext*.

## wxCanvas::ResetContext

**void ResetContext(void)**

Reset a canvas's context set by *SetContext*.

# wxCanvas::Scroll

**void Scroll(int** *x_pos*, **int** *y_pos*)

Scrolls a canvas so the view start is at the given point. The positions are in scroll units, not pixels, so to convert to pixels you will have to multiply by the number of pixels per scroll increment. If either parameter is -1, that position will be ignored (no change in that direction).

# wxCanvas::SetBackground

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

Sets the current background brush for the canvas. The brush should not be deleted while being used by a canvas. All brushes are deleted automatically when the application terminates.

# wxCanvas::SetClippingRegion

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

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

# wxCanvas::SetContext

**void SetContext(wxDC** **dc*)

Set a substitute context for the canvas, so applications which think they're drawing to canvases can be fooled into drawing to the printer. Reset with *ResetContext*.

# wxCanvas::SetBrush

**void SetBrush(wxBrush** **brush*)

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

# wxCanvas::SetFont

**void SetFont(wxFont** **font*)

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

## wxCanvas::SetLogicalFunction

**void SetLogicalFunction(int** *function***)**

Sets the current logical function for the canvas. The possible values are:

- wxXOR
- wxINVERT
- wxOR_REVERSE
- wxAND_REVERSE
- wxCOPY

The default is wxCOPY, which simply draws with the current colour. The others combine the current colour and the background using a logical operation. wxXOR is commonly used for drawing rubber bands or moving outlines, since drawing twice reverts to the original colour.

## wxCanvas::SetPen

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

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

## wxCanvas::SetScrollbars

**void SetScrollbars(int** *horiz_pixels*, **int** *vert_pixels*, **int** *x_length*, **int** *y_length*, **int** *x_page*, **int** *y_page***)**

Sets up vertical and/or horizontal scrollbars. The first pair of parameters give the number of pixels per 'scroll step', i.e. amount moved when the up or down scroll arrows are pressed. These may be 0 or less for no scrollbar. The second pair gives the length of scrollbar in scroll steps, which effectively sets the size of the 'virtual canvas'. The third pair gives the number of scroll steps in a 'page', i.e. amount moved when pressing above or below the scrollbar control, or using page up/page down (not yet implemented).

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

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

## wxCanvas::SetTextBackground

**void SetTextBackground(wxColour** **colour***)**

Sets the current text background colour for the canvas. Do not delete *colour*.

## wxCanvas::SetTextForeground

**void SetTextForeground(wxColour ***_colour_**)**

Sets the current text foreground colour for the canvas. Do not delete _colour_.

## wxCanvas::ViewStart

**void ViewStart(int ***_x_**, int *** _y_**)**

Get the position at which the visible portion of the canvas starts. If either of the scrollbars is not at the home position, _x_ and/or _y_ will be greater than zero. Combined with **GetClientSize**, the application can use this function to efficiently redraw only the visible portion of the canvas. The positions are in logical scroll units, not pixels, so to convert to pixels you will have to multiply by the number of pixels per scroll increment.

# 5.7 wxCheckBox: wxItem

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

## wxCheckBox::wxCheckBox

**void wxCheckBox(wxPanel ***_parent_**, wxFunction** _func_**, char ***_label_**,**
**int** _x = -1_**, int** _y = -1_**, int** _width = -1_**, int** _height = -1_**)**

Constructor, creating and showing a checkbox. If _width_ or _height_ are omitted (or are less than zero), an appropriate size will be used for the check box. _func_ may be NULL; otherwise it is used as the callback for the check box. Note that the cast (wxFunction) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration.

## wxCheckBox::~wxCheckBox

**void ~wxCheckBox(void)**

Destructor, destroying the checkbox.

## wxCheckBox::GetValue

**Bool GetValue(void)**

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

## wxCheckBox::SetValue

**void SetValue(Bool** _state_**)**

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

## 5.8 wxChoice: wxItem

A choice item is used to select one of a list of strings. Unlike a listbox, only the selection is visible until the user pulls down the menu of choices. Under XView and Motif, all selections are visible when the menu is displayed. Under Windows 3, a scrolling list is displayed when the user wants to change the selection. Note that under XView, creating a choice item with a large number of strings takes a long time due to the inefficiency of Sun's implementation of the XView choice item.

### wxChoice::wxChoice

**void wxChoice**(**wxPanel** **parent*, **wxFunction** *func*, **char** **label*,
**int** *x = -1*, **int** *y = -1*, **int** *width = -1*, **int** *height = -1*, **int** *n*, **char** **choices[]*)

Constructor, creating and showing a choice. If *width* or *height* are omitted (or are less than zero), an appropriate size will be used for the choice. *func* may be NULL; otherwise it is used as the callback for the choice. Note that the cast (wxFunction) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration.

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

### wxChoice::~wxChoice

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

Destructor, destroying the choice item.

### wxChoice::Append

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

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

### wxChoice::Clear

**void Clear**(**void**)

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

### wxChoice::FindString

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

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

## wxChoice::GetSelection

**int GetSelection(void)**

Gets the id (position) of the selected string.

## wxChoice::GetStringSelection

**char * GetStringSelection(void)**

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

## wxChoice::SetSelection

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

Sets the choice by passing the desired string position.

## wxChoice::SetStringSelection

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

Sets the choice by passing the desired string.

## wxChoice::String

**char * String(int** *n***)**

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

# 5.9 wxClient: wxIPCObject

A wxClient object represents the client part of a client-server DDE (Dynamic Data Exchange) conversation (available in *both* Windows and UNIX). See section 2.12.

## wxClient::wxClient

**void wxClient(void)**

Constructs a client object.

## wxClient::MakeConnection

**wxConnection * MakeConnection**(**char** **host*, **char** **service*, **char** **topic*)

Tries to make a connection with a server specified by the host (machine name under UNIX, ignored under Windows), service name (must contain an integer port number under UNIX), and topic string. If the server allows a connection, a wxConnection object will be returned. The type of wxConnection returned can be altered by deriving the **OnMakeConnection** member to return your own derived connection object.

## wxClient::OnMakeConnection

**wxConnection * OnMakeConnection**(**void**)

The type of wxConnection returned from a **MakeConnection** call can be altered by deriving the **OnMakeConnection** member to return your own derived connection object. By default, an ordinary wxConnection object is returned.

# 5.10 wxColour: wxObject

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

## wxColour::wxColour

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

**void wxColour**(**char** * *colour_name*)

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

## wxColour::Get

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

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

## wxColour::Set

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

Sets the RGB value.

# 5.11 wxColourDatabase: wxObject

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

The colours in the standard database are as follows:

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

Note that wxWindows' colour handling in XView and Motif canvases is poor and so only some of these colours are likely to show up. This should be improved in a subsequent release.

### wxColourDatabase::wxColourDatabase

**void wxColourDatabase**(**void**)

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

### wxColourDatabase::FindColour

**wxColour * FindColour**(**char** ***colour_name*)

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

### wxColourDatabase::FindName

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

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

### wxColourDatabase::Initialize

**char * Initialize**(**void**)

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

## 5.12 wxConnection: wxObject

A wxConnection object represents the connection between a client and a server. It can be created by making a connection using a client object, or by the acceptance of a connection by a server object. It implements the bulk of a DDE (Dynamic Data Exchange) conversation (available in *both* Windows and UNIX). See section 2.12.

## wxConnection::wxConnection

**void wxConnection(void)**

**void wxConnection(char** **buffer*, **int** *size*)

Constructs a connection object. If no user-defined connection object is to be derived from wxConnection, then the constructor should not be called directly, since the default connection object will be provided on requesting (or accepting) a connection. However, if the user defines his or her own derived connection object, the **Server::OnAcceptConnection** and/or **Client::OnMakeConnection** members should be replaced by functions which construct the new connection object. If the arguments of the wxConnection constructor are void, then a default buffer is associated with the connection. Otherwise, the programmer must provide a a buffer and size of the buffer for the connection object to use in transactions.

## wxConnection::Advise

**Bool Advise(char** **item*, **char** **data*, **int** *size = -1*, **int** *format = wxCF_TEXT*)

Called by the server application to advise the client of a change in the data associated with the given item. Causes the client connection's **OnAdvise** member to be called. Returns TRUE if successful.

## wxConnection::Execute

**Bool Execute(char** **data*, **int** *size = -1*, **int** *format = wxCF_TEXT*)

Called by the client application to execute a command on the server. Can also be used to transfer arbitrary data to the server (similar to **Poke** in that respect). Causes the server connection's **OnExecute** member to be called. Returns TRUE if successful.

## wxConnection::Disconnect

**Bool Disconnect(void)**

Called by the client or server application to disconnect from the other program; it causes the **OnDisconnect** message to be sent to the corresponding connection object in the other program. The default behaviour of **OnDisconnect** is to delete the connection, but the calling application must explicitly delete its side of the connection having called **Disconnect**. Returns TRUE if successful.

## wxConnection::OnAdvise

**Bool OnAdvise(char** **topic*, **char** **item*, **char** **data*, **int** *size*, **int** *format*)

Message sent to the client application when the server notifies it of a change in the data associated with the given item.

## wxConnection::OnDisconnect

**Bool OnDisconnect(void)**

Message sent to the client or server application when the other application notifies it to delete the connection. Default behaviour is to delete the connection object.

# wxConnection::OnExecute

**Bool OnExecute(char** **topic*, **char** **data*, **int** *size*, **int** *format*)

Message sent to the server application when the client notifies it to execute the given data. Note that there is no item associated with this message.

# wxConnection::OnPoke

**Bool OnPoke(char** **topic*, **char** **item*, **char** **data*, **int** *size*, **int** *format*)

Message sent to the server application when the client notifies it to accept the given data.

# wxConnection::OnRequest

**char * OnRequest(char** **topic*, **char** **item*, **int** **size*, **int** *format*)

Message sent to the server application when the client calls **Request**. The server should respond by returning a character string from **OnRequest**, or NULL to indicate no data.

# wxConnection::OnStartAdvise

**Bool OnStartAdvise(char** **topic*, **char** **item*)

Message sent to the server application by the client, when the client wishes to start an 'advise loop' for the given topic and item. The server can refuse to participate by returning FALSE.

# wxConnection::OnStopAdvise

**Bool OnStopAdvise(char** **topic*, **char** **item*)

Message sent to the server application by the client, when the client wishes to stop an 'advise loop' for the given topic and item. The server can refuse to stop the advise loop by returning FALSE, although this doesn't have much meaning in practice.

# wxConnection::Poke

**Bool Poke(char** **item*, **char** **data*, **int** *size = -1*, **int** *format = wxCF_TEXT*)

Called by the client application to poke data into the server. Can be used to transfer arbitrary data to the server. Causes the server connection's **OnPoke** member to be called. Returns TRUE if successful.

## wxConnection::Request

**char * Request**(**char** **item*, **int** **size*, **int** *format = wxCF_TEXT*)

Called by the client application to request data from the server. Causes the server connection's **OnRequest** member to be called. Returns a character string (actually a pointer to the connection's buffer) if successful, NULL otherwise.

## wxConnection::StartAdvise

**Bool StartAdvise**(**char** **item*)

Called by the client application to ask if an advise loop can be started with the server. Causes the server connection's **OnStartAdvise** member to be called. Returns TRUE if the server okays it, FALSE otherwise.

## wxConnection::StopAdvise

**Bool StopAdvise**(**char** **item*)

Called by the client application to ask if an advise loop can be stopped. Causes the server connection's **OnStopAdvise** member to be called. Returns TRUE if the server okays it, FALSE otherwise.

# 5.13 wxCursor: wxObject

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 Windows 3 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** for an example).

A single cursor object may be used in many windows (any subwindow type). The wxWindows convention is to set the cursor for a window, as in X, rather than to set it globally as in Windows 3, although a global **wxSetCursor** is also available for Windows 3 use.

Run the *hello* demo program to see what stock cursors are available.

## wxCursor::wxCursor

**void wxCursor**(**short** *bits[]*, **int***width*, **int** *height*)

**void wxCursor**(**char** * *cursor_name*)

**void wxCursor**(**int** *id*)

Constructor. A cursor can be created by passing an array of bits (XView and Motif only) by passing a string name, or by passing a stock cursor id. *cursor_name* refers to a filename in X, or a resource name in Windows 3.

The following stock cursor ids may be used:

- wxCURSOR_ARROW
- wxCURSOR_BULLSEYE
- wxCURSOR_CHAR
- wxCURSOR_CROSS
- wxCURSOR_HAND
- wxCURSOR_IBEAM
- wxCURSOR_LEFT_BUTTON
- wxCURSOR_MAGNIFIER
- wxCURSOR_MIDDLE_BUTTON
- wxCURSOR_NO_ENTRY
- wxCURSOR_PAINT_BRUSH
- wxCURSOR_PENCIL
- wxCURSOR_POINT_LEFT
- wxCURSOR_POINT_RIGHT
- wxCURSOR_QUESTION_ARROW
- wxCURSOR_RIGHT_BUTTON
- wxCURSOR_SIZENESW
- wxCURSOR_SIZENS
- wxCURSOR_SIZENWSE
- wxCURSOR_SIZEWE
- wxCURSOR_SIZING
- wxCURSOR_SPRAYCAN
- wxCURSOR_WAIT
- wxCURSOR_WATCH

## wxCursor::~wxCursor

**void ~wxCursor(void)**

Destroys the cursor. Unlike an icon, a cursor can be reused for more than one window, and does not get destroyed when the window is destroyed. wxWindows destroys all cursors on application exit.

## 5.14 wxDC: wxObject

A wxDC is *device context* onto which graphics and text can be drawn. It is intended to represent a number of output devices in a generic way, so a canvas has a device context and a printer also has a device context. In this way, the same piece of code may write to a number of different devices, if the device context is used as a parameter.

To determine whether a device context is colour or monochrome, test the **Colour** Bool member variable. To override wxWindows monochrome graphics drawing behaviour, set this member to TRUE.

# wxDC::wxDC

**void wxDC(void)**

**void wxDC(wxCanvas** **canvas*)

**void wxDC(char** **driver*, **char** **device*, **char** **output*, **Bool** *interactive = TRUE*)

Constructor. The third form may be called with three NULLs to put up a default printer dialog in Windows or UNIX. The only supported printer type in UNIX is "PostScript"; in Windows, specifying "PostScript" uses wxWindow's own Encapsulated PostScript driver, writing to a file only. Otherwise, *device* indicates the type of printer and *output* is an optional file for printing to. The *driver* parameter is currently unused. Use the *Ok* member to test whether the constructor was successful in creating a useable device context.

The following global variables are defined in wxWindows, edited by the user in the printer dialog and used by the PostScript driver. An application may wish to set them to appropriate default values.

- char *wx_portrait = TRUE
- char *wx_printer_command = "lpr"
- char *wx_printer_flags = ""
- Bool wx_preview = TRUE
- char *wx_preview_command = "ghostview"
- float wx_printer_scale_x = 1.0
- float wx_printer_scale_y = 1.0
- float wx_printer_translate_x = 0.0
- float wx_printer_translate_y = 0.0
- Bool wx_print_to_file = FALSE

# wxDC::~wxDC

**void ~wxDC(void)**

Destructor.

## wxDC::Blit

**Bool Blit**(**float** *xdest*, **float** *ydest*, **float** *width*, **float** *height*,
**wxDC** **source*, **float** *xsrc*, **float** *ysrc*, **int** *logical_func*)

Copy from a source DC to this DC, specifying the destination coordinates, size of area to copy, source DC, source coordinates, and logical function (see **SetLogicalFunction**). See **CreateCompatibleDC** for typical usage.

## wxDC::Clear

**void Clear**(**void**)

## wxDC::CreateCompatibleDC

**wxDC * CreateCompatibleDC**(**void**)

Creates and returns a device context compatible with this DC. A bitmap must be selected into the new DC before it may be used for anything. Typical usage is as follows:

```
wxDC *temp_dc = dc.CreateCompatibleDC();
temp_dc->SelectObject(test_bitmap);
dc.Blit(250, 50, BITMAP_WIDTH, BITMAP_HEIGHT, temp_dc, 0, 0);
delete temp_dc;
```

In future versions of wxWindows, this kind of DC will be used for drawing graphics in memory, then copying to visible windows.

## wxDC::DestroyClippingRegion

**void DestroyClippingRegion**(**void**)

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

## wxDC::DeviceToLogicalX

**float DeviceToLogicalX**(**int** *x*)

Convert device X coordinate to logical coordinate, using the current mapping mode.

## wxDC::DeviceToLogicalY

**float DeviceToLogicalY**(**int** *y*)

Converts device Y coordinate to logical coordinate, using the current mapping mode.

## wxDC::DrawEllipse

**void DrawEllipse(float** *x*, **float** *y*, **float** *width*, **float** *height*)

Draws an ellipse contained in the rectangle with the given top left corner, and with the given size. The current pen is used for the outline and the current brush for filling the shape.

## wxDC::DrawIcon

**void DrawIcon(wxIcon** **icon*, **float** *x*, **float** *y*)

Draw an icon on the display (does nothing if the device context is PostScript). This can be the simplest way of drawing bitmaps on a canvas. Icons and bitmaps in X are currently monochrome only.

## wxDC::DrawLine

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

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

## wxDC::DrawLines

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

**void DrawLines(wxList** **points*, **float** *xoffset = 0*, **float** *yoffset = 0*)

Draws lines using an array of *points* of size *n*, or list of pointers to points, adding the optional offset coordinate. The current pen is used for drawing the lines. The programmer is responsible for deleting the list of points.

## wxDC::DrawPolygon

**void DrawPolygon(int** *n*, **wxPoint** *points[]*, **float** *xoffset = 0*, **float** *yoffset = 0*)

**void DrawPolygon(wxList** **points*, **float** *xoffset = 0*, **float** *yoffset = 0*)

Draws a filled polygon 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 outline, and the current brush for filling the shape. Using a transparent brush suppresses filling. The programmer is responsible for deleting the list of points.

Note that wxWindows does not close the first and last points automatically.

## wxDC::DrawPoint

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

Draws a point using the current pen.

## wxDC::DrawRectangle

**void DrawRectangle**(**float** *x*, **float** *y*, **float** *width*, **float** *height*)

Draws a rectangle with the given top left corner, and with the given size. The current pen is used for the outline and the current brush for filling the shape.

## wxDC::DrawRoundedRectangle

**void DrawRoundedRectangle**(**float** *x*, **float** *y*, **float** *width*, **float** *height*, **float** *radius = 20*)

Draws a rectangle with the given top left corner, and with the given size. The corners are quarter-circles using the given radius. The current pen is used for the outline and the current brush for filling the shape.

## wxDC::DrawSpline

**void DrawSpline**(**wxList** **points*)

Draws a spline between all given control points, using the current pen. Doesn't delete the wxList and contents. The spline is drawn using a series of lines, using an algorithm taken from the X drawing program 'XFIG'.

**void DrawSpline**(**float** *x1*, **float** *y1*, **float** *x2*, **float** *y2*, **float** *x3*, **float** *y3*)

Draws a three-point spline using the current pen.

## wxDC::DrawText

**void DrawText**(**char** **text*, **float** *x*, **float** *y*)

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

## wxDC::EndDoc

**void EndDoc**(**void**)

Ends a document (only relevant when outputting to a printer).

## wxDC::EndPage

**void EndPage**(**void**)

Ends a document page (only relevant when outputting to a printer).

## wxDC::GetMapMode

**int GetMapMode**(**void**)

Gets the *mapping mode* for the device context (see **SetMapMode**).

## wxDC::LogicalToDeviceX

**int LogicalToDeviceX**(**float** *x*)

Converts logical X coordinate to device coordinate, using the current mapping mode.

## wxDC::LogicalToDeviceY

**int LogicalToDeviceY**(**float** *y*)

Converts logical Y coordinate to device coordinate, using the current mapping mode.

## wxDC::Ok

**Bool Ok**(**void**)

Returns TRUE if the DC is ok to use.

## wxDC::SelectObject

**void SelectObject**(**wxBitmap** **bitmap*)

Selects the given bitmap into the device context, to use as the memory bitmap. Drawing onto the DC is not yet allowed, but selecting the bitmap into a DC created by using **CreateCompatibleDC** allows you to then use **Blit** to copy the bitmap to a canvas. For this purpose, you may find **DrawIcon** easier to use instead, if your Windows bitmaps can be converted to icon format.

## wxDC::SetBackground

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

Sets the current background brush for the DC. Do not delete the brush - it will be deleted automatically when the application terminates.

## wxDC::SetClippingRegion

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

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

## wxDC::SetBrush

**void SetBrush**(**wxBrush** **brush*)

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

## wxDC::SetFont

**void SetFont**(**wxFont** **font*)

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

## wxDC::SetLogicalFunction

**void SetLogicalFunction**(**int** *function*)

Sets the current logical function for the DC. The possible values are:

- wxXOR
- wxINVERT
- wxOR_REVERSE
- wxAND_REVERSE
- wxCOPY

The default is wxCOPY, which simply draws with the current colour. The others combine the current colour and the background using a logical operation. wxXOR is commonly used for drawing rubber bands or moving outlines, since drawing twice reverts to the original colour.

## wxDC::SetMapMode

**void SetMapMode**(**int** *int*)

The *mapping mode* of the device context defines the unit of measurement used to convert logical units to device units. Note that in X, text drawing isn't handled consistently with the mapping mode; a font is always specified in point size. However, setting the *user scale* (see **SetUserScale**) scales the text appropriately. In Windows, scaleable TrueType fonts are always used; in X, results depend on availability of fonts, but usually a reasonable match is found.

Note that the coordinate origin should ideally be selectable, but for now is always at the top left of the screen/printer.

Drawing to a Windows printer device context under UNIX uses the current mapping mode, but mapping mode is currently ignored for PostScript output.

The mapping mode can be one of the following:

- MM_TWIPS - each logical unit is 1/20 of a point, or 1/1440 of an inch
- MM_POINTS - each logical unit is a point, or 1/72 of an inch
- MM_METRIC - each logical unit is 1 mm
- MM_LOMETRIC - each logical unit is 1/10 of a mm

- MM_TEXT - each logical unit is 1 pixel

## wxDC::SetPen

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

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

## wxDC::SetTextBackground

**void SetTextBackground**(**wxColour** **colour*)

Sets the current text background colour for the DC. Do not delete *colour* while selected for use by a DC.

## wxDC::SetTextForeground

**void SetTextForeground**(**wxColour** **colour*)

Sets the current text foreground colour for the DC. Do not delete *colour* while selected for use by a DC.

## wxDC::SetUserScale

**void SetUserScale**(**float** *x_scale*, **float***y_scale*)

Sets the user scaling factor, useful for applications which require 'zooming'.

## wxDC::StartDoc

**Bool StartDoc**(**char** **message*)

Starts a document (only relevant when outputting to a printer). Message is a message to show whilst printing.

## wxDC::StartPage

**Bool StartPage**(**void**)

Starts a document page (only relevant when outputting to a printer).

## 5.15 wxDialogBox: wxPanel

A dialog box is similar to a panel, with the following exceptions:

1. A surrounding frame is implicitly created.
2. Extra functionality is automatically given to the dialog box, such as tabbing between items (currently Windows only).
3. If the dialog box is *modal*, the calling program is blocked until the dialog box is dismissed.

Under XView, some panel items may display incorrectly in a modal dialog. An XView bug-fix for list boxes is supplied with wxWindows, but some items remain a problem.

Note that under Windows 3, modal dialogs have to be emulated using modeless dialogs and a message loop. This is because Windows 3 expects the contents of a modal dialog to be loaded from a resource file or created on receipt of a dialog initialization message. This is too restrictive for wxWindows, where any window may be created and displayed before its contents are created.

It would be easy to add a facility for loading Windows 3 dialog resources instead of building them programmatically, but of course this method is very non-portable. See the discussion in section 2.2.

For a set of dialog convenience functions, including file selection, see Section 6.3.

# wxDialogBox::wxDialogBox

**void wxDialogBox(wxFrame *****parent*, **char *****title*, **Bool** *modal=FALSE*,
**int** *x=300*, **int** *y=300*, **int** *width=500*, **int** *height=500*)

Constructor. If *modal* is TRUE, the dialog box will wait to be dismissed (using `Show(FALSE)`) before returning control to the calling program.

# wxDialogBox::~wxDialogBox

**void ~wxDialogBox(void)**

Destructor. Deletes any panel items before deleting the physical window.

# wxDialogBox::Iconize

**void Iconize(Bool** *iconize*)

If TRUE, iconizes the dialog box; if FALSE, shows and restores it. Note that in Windows, iconization has no effect since dialog boxes cannot be iconized. However, applications may need to explicitly restore dialog boxes under XView and Motif which have user-iconizable frames, and under Windows calling `Iconize(FALSE)` will bring the window to the front, as does `Show(TRUE)`.

# wxDialogBox::Iconized

**Bool Iconized(void)**

Returns TRUE if the dialog box is iconized. Always returns FALSE under Windows for the reasons given above.

## wxDialogBox::Show

**void Show(Bool** *show***)**

If *show* is TRUE, the dialog box is shown and brought to the front; otherwise the box is hidden. If *show* is FALSE and the dialog is modal, control is returned to the calling program.

# 5.16 wxEvent: wxObject

An event is a general-purpose structure holding information about an event passed to a callback or member function. Call member functions of **wxEvent** to find out information appropriate to the kind of event, or query the wxWindow object itself (for example list box, canvas) to find out the status of the object.

## wxEvent::wxEvent

**void wxEvent(void)**

Constructor. Should not need to be used by an application.

## wxEvent::~wxEvent

**void ~wxEvent(void)**

Destructor. Should not need to be used by an application.

## wxEvent::Button

**Bool Button(int** *button***)**

Returns TRUE if the identified mouse button is changing state. Valid values of *button* are:

1. Left button
2. Middle button
3. Right button

Not all mice have middle buttons so a portable application should avoid this one.

## wxEvent::ButtonDown

**Bool ButtonDown(void)**

Returns TRUE if the event was a mouse button down event.

## wxEvent::ControlDown

**Bool ControlDown(void)**

Returns TRUE if the control button was down at the time of the event.

## wxEvent::Dragging

**Bool Dragging(void)**

Returns TRUE if this was a dragging event.

## wxEvent::IsButton

**Bool IsButton(void)**

Returns TRUE if the event was a mouse button event (not necessarily a button down event - that may be tested using *ButtonDown*).

## wxEvent::LeftDown

**Bool LeftDown(void)**

Returns TRUE if the left mouse button changed to down.

## wxEvent::LeftUp

**Bool LeftUp(void)**

Returns TRUE if the left mouse button changed to up.

## wxEvent::MiddleDown

**Bool MiddleDown(void)**

Returns TRUE if the middle mouse button changed to down.

## wxEvent::MiddleUp

**Bool MiddleUp(void)**

Returns TRUE if the middle mouse button changed to up.

## wxEvent::Position

**void Position(float** **x*, **float** **y*)

Sets *x and *y to the position at which the event occurred. If the window is a canvas, the position is converted to logical units (according to the current mapping mode) with scrolling taken into account. To get back to device units (for example to calculate where on the screen to place a dialog box associated with a canvas mouse event), use **wxDC::LogicalToDeviceX** and **wxDC::LogicalToDeviceY**.

For example, the following code calculates screen pixel coordinates from the frame position, canvas view start (assuming the canvas is the only subwindow on the frame and therefore at the top left of it), and the logical event position. A menu is popped up at the position where the mouse click occurred. (Note that the application should also check that the dialog box will be visible on the screen, since the click could have occurred near the screen edge!)

```
float event_x, event_y;
event.Position(&event_x, &event_y);
frame->GetPosition(&x, &y);
canvas->ViewStart(&x1, &y1);
int mouse_x = (int)(canvas->GetDC()->LogicalToDeviceX(event_x + x - x1);
int mouse_y = (int)(canvas->GetDC()->LogicalToDeviceY(event_y + y - y1);

char *choice = wxGetSingleChoice("Menu", "Pick a node action",
                                 no_choices, choices, frame, mouse_x, mouse_y);
```

## wxEvent::RightDown

**Bool RightDown(void)**

Returns TRUE if the right mouse button changed to down.

## wxEvent::RightUp

**Bool RightUp(void)**

Returns TRUE if the right mouse button changed to up.

## wxEvent::ShiftDown

**Bool ShiftDown(void)**

Returns TRUE if the shift button was down at the time of the event.

# 5.17 wxFont: wxObject

A font is an object which determines the appearance of text, primarily when drawing text to a canvas or device context. A font is determined by four parameters:

- Point size. This is the standard way of referring to text size.

- Family. Supported families are: wxDEFAULT, wxDECORATIVE, wxROMAN, wxSCRIPT, wxSWISS, wxMODERN. wxMODERN is a fixed pitch font; the others are either fixed or variable pitch.
- Style. The value can be wxNORMAL, wxSLANT or wxITALIC.
- Weight. The value can be wxNORMAL, wxLIGHT or wxBOLD.

There is currently a difference between the appearance of fonts on the two platforms, if the mapping mode is anything other than MM_TEXT. Under X, font size is always specified in points. Under Windows, the unit for text is points but the text is scaled according to the current mapping mode. However, user scaling on a device canvas will also scale fonts under both environments. A future version of wxWindows will attempt to make font appearance more consistent across platforms.

## wxFont::wxFont

**void wxFont(void)**

**void wxFont(int** *point_size*, **int** *family*, **int** *style*, **int** *weight*)

Creates a font object. If the desired font does not exist, the closest match will be chosen. Under XView, this may result in a number of XView warnings during the matching process; these should be ignored, and will only occur the first time wxWindows attempts to use an absent font in a given size. wxWindows under Motif does the same thing, but silently. Under Windows 3, only scaleable TrueType fonts are used.

## wxFont::~wxFont

**void ~wxFont(void)**

Destroys a font object. Do not manually destroy a font which has been assigned to a canvas. All GDI objects, including fonts, are automatically destroyed on program exit, so there is no danger of memory leakage as in conventional Windows programming.

## wxFont::GetPointSize

**int GetPointSize(void)**

Gets the point size.

## 5.18 wxForm: wxObject

### 5.18.1 The purpose of the form class

The wxForm is a stab at providing form-like functionality, relieving the programmer of the tedium of defining all the physical panel items and the callbacks handling out-of-range data. It allows the application writer to write form dialogs quickly (albeit programmatically) with panel items being chosen automatically according to the given constraints. The supplied form demo shows how succinct a form definition can be. A form gets laid out from left to right; the programmer can intersperse new lines and specify item sizes, but for brevity no more control is allowed.

A form does not presuppose a particular type of panel: any window derived from wxPanel may be associated with a form, once the form has been built by adding form items. Also, a form reads from and writes to any C++ variables in your program - just supply pointers to the variables, and the form handles the rest.

### 5.18.2 Constraints on form items

Each item in a form may be supplied with zero or more constraints, where the range of possible constraints depends on the data type, and the displayed panel item depends upon the data type and the constraint(s) given. For example, a string form item with a list of possible strings as a constraint will produce a list box on the panel; an integer form item with a range constraint will result in a slider being displayed. The user may define his or her own constraint by passing a function as a constraint which returns FALSE if the constraint was violated, TRUE otherwise. The function should write an appropriate message into the buffer passed to it if the constraint was violated.

### 5.18.3 Form appearance

Once displayed on a panel, a form shows Ok, Cancel, Update and Revert buttons along the top, with the user-supplied items below. When the user presses Ok, the form items are checked for violation of constraints; if any violations are found, an appropriate error message is displayed and the user must correct the mistake (or press Cancel, which leaves the item values as they were after the last Update). Pressing Update also checks the constraints and updates the values, but typically does not dismiss the dialog. Revert causes the displayed values to take on the values at the last Update. By default, the OnOk and OnCancel messages dismiss and delete the dialog box and form, but these may be overridden by the application (see below).

The display-type values which may be passed to a form-item creation function are as follows:

- wxFORM_DEFAULT: let wxWindows choose a suitable panel item
- wxFORM_SINGLE_LIST: use a single-selection listbox. Default for string item with a one-of constraint
- wxFORM_CHOICE: use a choice item
- wxFORM_CHECKBOX: use a checkbox. Default for boolean item
- wxFORM_TEXT: use a single-line text item. Default for floating point item, and for string and integer items with no constraints
- wxFORM_MULTITEXT: use a multi-line text item
- wxFORM_SLIDER: use a slider. Default for integer item with range constraint

The wxFormItem and wxFormItemConstraint classes are not detailed in this manual since their members do not need to be directly accessed by the user. Functions for creating form items and constraints for passing to **wxForm::Add** are given in the next subsection.

### 5.18.4 Example

The following is an example of a form definition, taken from the form demo. Here, a new form **MyForm** has been derived, and a new member **EditForm** has been defined to edit objects of the type **MyObject**, given a panel to display it on.

```
void MyForm::EditForm(MyObject *object, wxPanel *panel)
{
  Add(wxMakeFormString("string 1", &(object->string1), wxFORM_DEFAULT,
                       new wxList(wxMakeConstraintFunction(MyConstraint), 0)));
  Add(wxMakeFormNewLine());

  Add(wxMakeFormString("string 2", &(object->string2), wxFORM_DEFAULT,
                  new wxList(wxMakeConstraintStrings("One", "Two", "Three", 0), 0)));
  Add(wxMakeFormString("string 3", &(object->string3), wxFORM_CHOICE,
                       new wxList(wxMakeConstraintStrings("Pig", "Cow",
                                  "Aardvark", "Gorilla", 0), 0)));
  Add(wxMakeFormNewLine());
  Add(wxMakeFormShort("int 1", &(object->int1), wxFORM_DEFAULT,
                       new wxList(wxMakeConstraintRange(0.0, 50.0), 0)));
  Add(wxMakeFormNewLine());

  Add(wxMakeFormFloat("float 1", &(object->float1), wxFORM_DEFAULT,
                       new wxList(wxMakeConstraintRange(-100.0, 100.0), 0)));
  Add(wxMakeFormBool("bool 1", &(object->bool1)));
  Add(wxMakeFormNewLine());

  Add(wxMakeFormButton("Test button", (wxFunction)MyButtonProc));

  AssociatePanel(panel);
}
```

# wxForm::wxForm

**void wxForm(void)**

Constructor.

# wxForm::~wxForm

**void ~wxForm(void)**

Destructor. Does not delete the associated panel or any panel items, but does delete all form items.

# wxForm::Add

**void Add(wxFormItem** **item*, **long** *id = -1*)

Adds a form item to the form. If an id is given this is associated with the form item; otherwise a new id is generated, by which the item may be identified later.

## wxForm::FindItem

**wxNode * FindItem(long** *id***)**

Given a form item id, returns a list node containing the form item.

## wxForm::Set

**Bool Set(long** *id*, **wxFormItem** **item***)**

Given a form item id, replaces an existing item with that id with the given form item. Returns TRUE if successful.

## wxForm::Delete

**Bool Delete(long** *id***)**

Deletes the given form item by id. Returns TRUE if successful.

## wxForm::AssociatePanel

**void AssociatePanel(wxPanel** **panel***)**

Associates the form with the given panel (or window derived from wxPanel, such as wxDialogBox). This causes a number of items to be created on the panel using information from the list of form items. The panel should be shown after this has been called.

## wxForm::OnCancel

**void OnCancel(void)**

This member may be derived by the application. When the user presses the Cancel button, this is called, allowing the application to take action. By default, **OnCancel** deletes the form and the panel associated with it, probably the normal desired behaviour.

## wxForm::OnOk

**void OnOk(void)**

This member may be derived by the application. When the user presses the OK button, this is called, allowing the application to take action. By default, **OnOk** deletes the form and the panel associated with it, probably the normal desired behaviour. Note that if any form item constraints were violated when the user pressed OK, the member does not get called.

## wxForm::OnRevert

**void OnRevert(void)**

This member may be derived by the application. When the user presses the Revert button, the C++ form item variable values in effect before the last Update are restored. Then this member is called, allowing the application to take further action.

# wxForm::OnUpdate

**void OnUpdate(void)**

This member may be derived by the application. When the user presses the Update button, the C++ form item variable values are updated to the values on the panel. Then this member is called, allowing the application to take further action.

# wxForm::RevertValues

**void RevertValues(void)**

Internal function for displaying the C++ form item values in the displayed panel items. Should not need to be called by the user.

# wxForm::UpdateValues

**Bool UpdateValues(void)**

Internal function for setting the C++ form item values to the values set in the panel items. Should not need to be called by the user.

### 5.18.5 Functions for making form items and constraints

These functions make form items and their associated constraints for passing to **wxForm::Add**.

**wxFormItem * wxMakeFormButton(char** **label*, **wxFunction** *fun*)

Makes a button with a conventional callback.

**wxFormItem * wxMakeFormMessage(char** **label*)

Makes a message.

**wxFormItem * wxMakeFormNewLine(void)**

Adds a newline.

**wxFormItem * wxMakeFormLong(char** **label*, **long** **var*,
**int** *item_type = wxFORM_DEFAULT*, **wxList** **constraints = NULL*,
**char** **help_string = NULL*, **wxEditFunction** *editor = NULL*, **int** *width = -1*,
**int** *height = -1*)

Makes a long integer form item, given a label, a pointer to the variable holding the value, an item type, and a list of constraints (see below). *help_string* and *editor* are currently not used.

**wxFormItem * wxMakeFormShort**(**char** **label*, **int** **var*,
**int** *item_type = wxFORM_DEFAULT*, **wxList** **constraints = NULL*,
**char** **help_string = NULL*, **wxEditFunction** *editor = NULL*, **int** *width = -1*,
**int** *height = -1*)

Makes an integer form item, given a label, a pointer to the variable holding the value, an item type, and a list of constraints (see below). *help_string* and *editor* are currently not used.

**wxFormItem * wxMakeFormFloat**(**char** **label*, **float** **var*,
**int** *item_type = wxFORM_DEFAULT*, **wxList** **constraints = NULL*,
**char** **help_string = NULL*, **wxEditFunction** *editor = NULL*, **int** *width = -1*,
**int** *height = -1*)

Makes a floating-point form item, given a label, a pointer to the variable holding the value, an item type, and a list of constraints (see below). *help_string* and *editor* are currently not used.

**wxFormItem * wxMakeFormBool**(**char** **label*, **Bool** **var*,
**int** *item_type = wxFORM_DEFAULT*, **wxList** **constraints = NULL*,
**char** **help_string = NULL*, **wxEditFunction** *editor = NULL*, **int** *width = -1*,
**int** *height = -1*)

Makes a boolean form item, given a label, a pointer to the variable holding the value, an item type, and a list of constraints (see below). *help_string* and *editor* are currently not used.

**wxFormItem * wxMakeFormString**(**char** **label*, **char** ***var*,
**int** *item_type = wxFORM_DEFAULT*, **wxList** **constraints = NULL*,
**char** **help_string = NULL*, **wxEditFunction** *editor = NULL*, **int** *width = -1*,
**int** *height = -1*)

Makes a string form item, given a label, a pointer to the variable holding the value, an item type, and a list of constraints (see below). *help_string* and *editor* are currently not used.

**wxFormItemConstraint * wxMakeConstraintStrings**(**wxList** **list*)

Makes a constraint specifying that the value must be one of the strings given in the list.

**wxFormItemConstraint * wxMakeConstraintStrings**(**char** **first, ...*)

Makes a constraint specifying that the value must be one of the strings given in the variable-length argument list,*terminated with a zero.*

**wxFormItemConstraint * wxMakeConstraintFunction**(**wxConstraintFunction** *func*)

Makes a constraint with a function that gets called when the value is being checked. The function should return FALSE if the constraint was violated, TRUE otherwise. The function should also write an appropriate message into the buffer passed to it if the constraint was violated. The type **wxConstraintFunction** is defined as follows:

**typedef Bool (*wxConstraintFunction)**(**int** *type*, **char** **value*, **char** **label*, **char** **msg*)

*type* is the type of the item, for instance wxFORM_STRING. *value* is the address of the variable containing the value, and should be coerced to the correct type, except for wxFORM_STRING, where no coercion is required.

**wxFormItemConstraint * wxMakeConstraintRange**(**float** *lo*, **float** *hi*)

Makes a range constraint; can be used for integer and floating point form items.

## 5.19 wxFrame: wxWindow

A frame is a window which contains subwindows of various kinds. It has a title bar and, optionally, a menu bar, and a status line. Depending on the platform, the frame has further menus or buttons relating to window movement, sizing, closing, etc. Most of these events are handled by the host system without need for special handling by the application. However, the application should normally define an **OnClose** handler for the frame so that related data and subwindows can be cleaned up.

The Windows 3 issues of Multiple Document Interface (MDI) versus Single Document Interface (SDI) frames are covered in section 2.1.1 and in the tutorial.

### wxFrame::wxFrame

**void wxFrame**(**wxFrame** **parent*, **char** **title*, **int** *x = -1*, **int** *y = -1*,
**int** *width = -1*, **int** *height = -1*, **int** *style = wxSDI*)

Constructor. The *parent* parameter can be NULL or an existing frame. The final parameter determines whether, under Windows, the frame is an SDI frame (wxSDI), an MDI parent frame (wxMDI_PARENT) or an MDI child frame (wxMDI_CHILD).

### wxFrame::~wxFrame

**void ~wxFrame**(**void**)

Destructor. Destroys all child windows and menu bar if present.

### wxFrame::Centre

**void Centre**(**int** *direction = wxBOTH*)

Centres the frame on the display. The parameter may be `wxHORIZONTAL`, `wxVERTICAL` or `wxBOTH`.

### wxFrame::CreateStatusLine

**void CreateStatusLine**(**void**)

Creates a status line at the bottom of the frame. The width of the status line is the whole width of the frame (adjusted automatically when resizing), and the height and text size are chosen by the host system. Does not work for MDI parent frames.

### wxFrame::Iconize

**void Iconize**(**Bool** *iconize*)

If TRUE, iconizes the frame; if FALSE, shows and restores it.

## wxFrame::Iconized

**Bool Iconized**(**void**)

Returns TRUE if the frame is iconized.

## wxFrame::OnMenuCommand

**void OnMenuCommand**(**int** *id*)

Sent to the window when an item on the window's menu has been chosen. Derive your own frame class to handle this message.

## wxFrame::OnMenuSelect

**void OnMenuSelect**(**int** *id*)

Sent to the window when an item on the window's menu has been selected (i.e. the cursor is on the item, but the left button has not been released). Derive your own frame class to handle this message. See the *hello* sample for an example of using this to implement a line of explanation about each menu item.

This function is only called under Windows 3.

## wxFrame::SetIcon

**void SetIcon**(**wxIcon *** *icon*)

Sets the icon for this frame, deleting any existing one. Note an important difference between XView and Windows 3 behaviour. In Windows 3, the title of the frame is the icon label, wrapping if necessary for a long title. If the frame title changes, the icon label changes. In XView, the icon label cannot be changed once the icon has been associated with the frame. Also, there is no wrapping, and icon labels must therefore be short.

The best thing to do to accommodate both situations is to have the frame title set to a short string when setting the icon. Then, set the frame title to the desired text. In XView, the icon will keep its short text. In Windows 3, the longer (probably more meaningful) title will be shown.

Note also that in Windows 3, icons cannot be associated with a window after window initialization, except by explicitly drawing the icon onto the iconized window, which is what wxWindows does. Because of this workaround, the background of the icon will be white rather than the usual transparent. It was felt limiting to have to pass an icon name at frame create time.

However, drawing the icon like this does not work (for some unknown reason) with MDI parent and child frames, and so for MDI applications the following lines need to be added to the Windows 3 resource file:

```
wxSTD_FRAME         ICON icon1.ico
wxSTD_MDICHILDFRAME ICON icon2.ico
```

where icon1.ico will be used for SDI frames or the MDI parent frame, and icon2.ico will be used for MDI child frames.

## wxFrame::SetMenuBar

**void SetMenuBar(wxMenuBar ****frame*)

Tells the frame to show the given menu bar. If the frame is destroyed, the menu bar and its menus will be destroyed also, so do not delete the menu bar explicitly (except by resetting the frame's menu bar to another frame or NULL).

## wxFrame::SetStatusText

**void SetStatusText(char *** *text*)

Sets the status line text and redraws the status line. Use an empty (not NULL) string to clear the status line.

## wxFrame::StatusLineExists

**Bool StatusLineExists(void)**

Returns TRUE if the status line has previously been created.

# 5.20 wxFunction

**typedef void (*wxFunction)(wxObject&, wxEvent&)**

The type of a callback function. See the comments in section 3.8.3.

# 5.21 wxIcon: wxObject

An icon is a small rectangular bitmap usually used for denoting a minimized application. It is optional (but desirable) to associate a pertinent icon with a frame. Obviously icons in X and Windows 3 are created in a different manner, and colour icons in X are difficult to arrange. Therefore, separate icons will be created for the different environments. Platform-specific methods for creating a **wxIcon** structure are catered for, and this is an occasion where conditional compilation will probably be required.

Note that a new icon must be created for every time the icon is to be used for a new window. In X, this will ensure that fresh X resources are allocated for this frame. In Windows 3, the icon will not be reloaded if it has already been used. An icon allocated to a frame will be deleted when the frame is deleted.

The following shows the conditional compilation required to define an icon in X and in Windows 3. The alternative is to use the string version of the icon constructor, which loads a file under X and a resource under Windows 3, but has the disadvantage of requiring the X icon file to be available at run-time. If anyone can invent a scheme or macro which does the following more elegantly and platform-independently, I'd like to see it!

```
#ifdef wx_x
short aiai_bits[] ={
#include "aiai.icon"
};
#endif
#ifdef wx_msw
  wxIcon *icon = new wxIcon("aiai");
#endif
#ifdef wx_x
  wxIcon *icon = new wxIcon(aiai_bits, 64, 64);
#endif
```

## wxIcon::wxIcon

**void wxIcon**(**short** *bits[]*, **int** *width*, **int** *height*)

**void wxIcon**(**char** * *icon_name*)

Constructor. An icon can be created by passing an array of bits (X only) or by passing a string name. *icon_name* refers to a filename in X, a resource name in Windows 3.

## wxIcon::~wxIcon

**void ~wxIcon**(**void**)

Destroys the icon. Do not explicitly delete an icon pointer which has been passed to a frame - the frame will delete the icon when it is destroyed. If assigning a new icon to a frame, the old icon will be destroyed.

# 5.22 wxHashTable: wxObject

This class provides hash table functionality for wxWindows, and for an application if it wishes. Data can be hashed on an integer or string key. Below is an example of using a hash table.

```
  wxHashTable table(KEY_STRING);

  wxPoint *point = new wxPoint(100, 200);
  table.Put("point 1", point);

  ....

  wxPoint *found_point = (wxPoint *)table.Get("point 1");
```

A hash table is implemented as an array of pointers to lists. When no data has been stored, the hash table takes only a little more space than this array (default size is 1000). When a data item is added, an integer is constructed from the integer or string key that is within the bounds of the array. If the array element is NULL, a new (keyed) list is created for the element. Then the data object is appended to the list, storing the key in case other data objects need to be stored in the list also (when a 'collision' occurs).

Retrieval involves recalculating the array index from the key, and searching along the keyed list for the data object whose stored key matches the passed key. Obviously this is quicker when there are fewer collisions, so hashing will become inefficient if the number of items to be stored greatly exceeds the size of the hash table.

## wxHashTable::wxHashTable

**void wxHashTable**(**unsigned int** *key_type*, **int** *size = 1000*)

Constructor. *key_type* is one of wxKEY_INTEGER, or wxKEY_STRING, and indicates what sort of keying is required. *size* is optional.

## wxHashTable::~wxHashTable

**void ~wxHashTable**(**void**)

Destroys the hash table.

## wxHashTable::BeginFind

**void BeginFind**(**void**)

The counterpart of *Next*. If the application wishes to iterate through all the data in the hash table, it can call *BeginFind* and then loop on *Next*.

## wxHashTable::Clear

**void Clear**(**void**)

Clears the hash table of all nodes (but as usual, doesn't delete user data).

## wxHashTable::Delete

**wxObject * Delete**(**long** *key*)

**wxObject * Delete**(**char *** *key*)

Deletes entry in hash table and returns the user's data (if found).

## wxHashTable::Get

**wxObject * Get**(**long** *key*)

**wxObject * Get(char *** *key***)**

Gets data from the hash table, using an integer or string key (depending on which has table constructor was used).

## wxHashTable::MakeKey

**long MakeKey(char ****string***)**

Makes an integer key out of a string. An application may wish to make a key explicitly (for instance when combining two data values to form a key).

## wxHashTable::Next

**wxNode * Next(void)**

If the application wishes to iterate through all the data in the hash table, it can call *BeginFind* and then loop on *Next*. This function returns a **wxNode** pointer (or NULL if there are no more nodes). See the **wxNode** description. The user will probably only wish to use the **wxNode::Data** function to retrieve the data; the node may also be deleted.

## wxHashTable::Put

**void Put(long** *key*, **wxObject ****object***)**

**void Put(char *** *key*, **wxObject ****object***)**

Inserts data into the hash table, using an integer or string key (depending on which has table constructor was used). Note that only the pointer to the string key is stored, so it should not be deallocated by the calling program.

# 5.23 wxHelpInstance: wxClient

The **wxHelpInstance** class implements the interface by which applications may invoke wxHelp to provide on-line help. Each instance of the class maintains one connection to an instance of wxHelp which belongs to the application, and which is shut down when the Quit member of **wxHelpInstance** is called (for example in the **OnClose** member of an application's main frame). Under Windows 3, there is currently only one instance of wxHelp which is used by all applications.

Since there is a DDE link between the two programs, each subsequent request to display a file or section uses the existing instance of wxHelp, rather than starting a new instance each time. wxHelp thus appears to the user to be an extension of the current application. wxHelp may also be invoked independently of a client application.

Normally an application will create an instance of **wxHelpInstance** when it starts, and immediately call **Initialize** to associate a filename with it. wxHelp will only get run, however, just before the first call to display something. See the test program supplied with the wxHelp source.

Include the file `wx_help.h` to use this API, even if you have included `wx.h`.

## wxHelpInstance::wxHelpInstance

**void wxHelpInstance(void)**

Constructs a help instance object, but does not invoke wxHelp.

## wxHelpInstance::~wxHelpInstance

Destroys the help instance, closing down wxHelp for this application if it is running.

## wxHelpInstance::Initialize

**void Initialize(char** **file*, **int** *server = -1*)

Initializes the help instance with a help filename, and optionally a server (socket) number (one is chosen at random if this parameter is omitted). Does not invoke wxHelp. This must be called directly after the help instance object is created and before any attempts to communicate with wxHelp.

## wxHelpInstance::DisplayBlock

**Bool DisplayBlock(long** *blockNo*)

If wxHelp is not running, runs wxHelp and displays the file at the given block number.

## wxHelpInstance::DisplayContents

**Bool DisplayContents(void)**

If wxHelp is not running, runs wxHelp and displays the contents (the first section of the file).

## wxHelpInstance::DisplaySection

**Bool DisplaySection(int** *sectionNo*)

If wxHelp is not running, runs wxHelp and displays the given section. Sections are numbered starting from 1, and section numbers may be viewed by running wxHelp in edit mode.

## wxHelpInstance::KeywordSearch

**Bool KeywordSearch(char** **keyWord*)

If wxHelp is not running, runs wxHelp, and searches for sections matching the given keyword. If one match is found, the file is displayed at this section. If more than one match is found, the Search dialog is displayed with the matches.

## wxHelpInstance::LoadFile

**Bool LoadFile**(**char** **file = NULL*)

If wxHelp is not running, runs wxHelp, and loads the given file. If the filename is not supplied or is NULL, the file specified in **Initialize** is used. If wxHelp is already displaying the specified file, it will not be reloaded. This member function may be used before each display call in case the user has opened another file.

## wxHelpInstance::OnQuit

**Bool OnQuit**(**void**)

Overridable member called when this application's wxHelp is quit.

## wxHelpInstance::Quit

**Bool Quit**(**void**)

If wxHelp is running, quits wxHelp by disconnecting.

# 5.24 wxItem: wxWindow

## wxItem::Centre

**void Centre**(**int** *direction = wxHORIZONTAL*)

Centres the frame on the panel or dialog box. The parameter may be `wxHORIZONTAL`, `wxVERTICAL` or `wxBOTH`.

You may still use **Fit** in conjunction with this call, but call **Fit** first before centring items.

## wxItem::SetDefault

**void SetDefault**(**void**)

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

Under Windows 3, only dialog box buttons respond to this function. As normal under Windows 3, pressing return causes the default button to be depressed when the return key is pressed. See also **wxWindow::SetFocus** which sets the keyboard focus for windows and text panel items.

## wxItem::SetLabel

**void wxItem**(**char** **label*)

Sets the item's label. A copy of the label is taken.

## wxItem::GetLabel

**char * wxItem(void)**

Gets a temporary pointer to the item's label.

## 5.25 wxList: wxObject

This class provides linked list functionality for wxWindows, and for an application if it wishes. Depending on the form of constructor used, a list can be keyed on integer or string keys to provide a primitive look-up ability. See **wxHashTable** for a faster method of storage when random access is required. It is very common to iterate on a list as follows:

```
  ...
  wxPoint *point1 = new wxPoint(100, 100);
  wxPoint *point2 = new wxPoint(200, 200);

  wxList SomeList;
  SomeList.Append(point1);
  SomeList.Append(point2);

  ...

  wxNode *node = SomeList.First();
  while (node)
 {
    wxPoint *point = (wxPoint *)node->Data();
    ...
    node = node->Next();
  }
```

To delete nodes in a list as the list is being traversed, replace

```
    ...
    node = node->Next();
    ...
```

with

```
    ...
    delete point;
    delete node;
    node = SomeList.First();
    ...
```

See **wxNode** for members that retrieve the data associated with a node, and members for getting to the next or previous node.

Note that a cast is required when retrieving the data from a node. Although a node is defined to store objects of type **wxObject** and derived types, other types (such as char *) may be used with appropriate casting.

## wxList::wxList

**void wxList(void)**

**void wxList(unsigned int** *key_type***)**

**void wxList(int** *n*, **wxObject ****objects[]***)**

**void wxList(wxObject ****object*, ...**)**

Constructors. *key_type* is one of wxKEY_NONE, wxKEY_INTEGER, or wxKEY_STRING, and indicates what sort of keying is required (if any).

*objects* is an array of *n* objects with which to initialize the list.

The variable-length argument list constructor must be supplied with a terminating NULL.

## wxList::~wxList

**void ~wxList(void)**

Destroys list. Also destroys any remaining nodes, but does not destroy client data held in the nodes.

## wxList::Append

**wxNode * Append(wxObject ****object***)**

**wxNode * Append(long** *key*, **wxObject ****object***)**

**wxNode * Append(char ****key*, **wxObject ****object***)**

Appends a new **wxNode** to the end of the list and puts a pointer to the *object* in the node. The last two forms store a key with the object for later retrieval using the key. The new node is returned in each case.

## wxList::Clear

**void Clear(void)**

Clears the list (but does not delete the client data stored with each node).

## wxList::DeleteContents

**void DeleteContents(Bool** *destroy***)**

If *destroy* is TRUE, instructs the list to call *delete* on the client contents of a node whenever the node is destroyed. The default is FALSE.

## wxList::DeleteNode

**Bool DeleteNode(wxNode ****node*)

Deletes the given node from the list, returning TRUE if successful.

# wxList::DeleteObject

**Bool DeleteObject(wxObject ****object*)

Finds the given client *object* and deletes the appropriate node from the list, returning TRUE if successful. The application must delete the actual object separately.

# wxList::Find

**wxNode * Find(long** *key*)

**wxNode * Find(char ****key*)

Returns the node whose stored key matches *key*. Use on a keyed list only.

# wxList::First

**wxNode * First(void)**

Returns the first node in the list (NULL if the list is empty).

# wxList::Insert

**wxNode * Insert(wxObject ****object*)

Insert object at front of list.

**wxNode * Insert(wxNode ****position*, **wxObject ****object*)

Insert object before *position*.

# wxList::Last

**wxNode * Last(void)**

Returns the last node in the list (NULL if the list is empty).

# wxList::Member

**Bool Member(wxObject ****object*)

Returns TRUE if the client data *object* is in the list.

# wxList::Nth

**wxNode * Nth(int** *n***)**

Returns the *nth* node in the list, indexing from zero (NULL if the list is empty or the nth node could not be found).

## wxList::Number

**int Number(void)**

Returns the number of elements in the list.

# 5.26 wxListBox: wxItem

A list box 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 list item 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).

## wxListBox::wxListBox

**void wxListBox(wxPanel ***parent*, **wxFunction** *func*, **char ***label*,
  **Bool** *multiple_selection = FALSE*, **int** *x = -1*, **int** *y = -1*,
  **int** *width = -1*, **int** *height = -1*, **int** *n*, **char ***choices[]*)

Constructor, creating and showing a list box. If *width* or *height* are omitted (or are less than zero), an appropriate size will be used for the list box. *func* may be NULL; otherwise it is used as the callback for the list box. Note that the cast (wxFunction) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration.

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

*multiple_selection* is TRUE for a multiple selection list box, FALSE for a single selection list box.

## wxListBox::~wxListBox

**void ~wxListBox(void)**

Destructor, destroying the list box.

## wxListBox::Append

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

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

**void Append(char *** *item*, **char ****client_data*)

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

## wxListBox::Clear

**void Clear(void)**

Clears all strings from the list box.

## wxListBox::GetClientData

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

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

## wxListBox::Deselect

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

Deselects the given item in the list box.

## wxListBox::FindString

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

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

## wxListBox::GetSelection

**int GetSelection(void)**

Gets the id (position) of the selected string - for single selection list boxes only.

## wxListBox::GetSelections

**int GetSelections(int ****selections***)**

Gets an array containing the positions of the selected strings. The number of selections is returned. Pass a pointer to an integer array, and do not deallocate the returned array.

## wxListBox::GetStringSelection

**char * GetStringSelection(void)**

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

## wxListBox::Set

**void Set(int** *n*, **char** ***choices[]*)

Clears the list box and adds the given strings. Deallocate the array from the calling program after this function has been called.

## wxListBox::SetSelection

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

Sets the choice by passing the desired string position.

## wxListBox::SetStringSelection

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

Sets the choice by passing the desired string.

## wxListBox::String

**char * String(int** *n*)

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

# 5.27 wxMenu: wxItem

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). At present, menus may only be used to construct menu bars, but will eventually have wider use.

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.

## wxMenu::wxMenu

**void wxMenu(char ***title = NULL*, **wxFunction** *func = NULL*)

Both arguments are presently ignored.

## wxMenu::~wxMenu

**void ~wxMenu(void)**

Destructor, destroying the menu.

## wxMenu::Append

**void Append(int** *id*, **char *** *item*)

**void Append(int** *id*, **char *** *item*, **wxMenu ****submenu*)

Adds the item to the end of the menu. *item* must be deallocated by the calling program. If the second form is used, the given menu will be a pullright submenu (must be created already). Do not use *submenu* after this call: it will be deallocated by wxWindows.

## wxMenu::AppendSeparator

**void AppendSeparator(void)**

Adds a separator to the end of the menu. Under XView, this appears as a space.

## wxMenu::Enable

**void Enable(int** *id*, **Bool** *flag*)

If *flag* is TRUE, enables the given menu item, else disables it (greys it).

## wxMenu::Check

**void Check(int** *id*, **Bool** *flag*)

If *flag* is TRUE, checks the given menu item, else unchecks it. Works in Windows 3 but has no effect in XView or Motif.

# 5.28 wxMenuBar: wxItem

A menu bar is a series of menus accessible from the top of a frame. Selecting a title pulls down a menu; selecting a menu item causes a *MenuSelection* message to be passed to the frame with the menu item integer id as the only argument.

## wxMenuBar::wxMenuBar

**void wxMenuBar(void)**

**void wxMenuBar(int** *n*, **wxMenu ****menus[]*, **char ****titles[]*)

Construct a menu bar. In the second form, the calling program must have created an array of menus and an array of titles. Do not use the submenus again after this call.

## wxMenuBar::~wxMenuBar

**void ~wxMenuBar(void)**

Destructor, destroying the menu bar and removing it from the parent frame (if any).

## wxMenuBar::Append

**void Append(wxMenu** **menu*, **char** **title*)

Adds the item to the end of the menu bar. Do not use *menu* after this call: it will be deallocated by wxWindows.

## wxMenuBar::Enable

**void Enable(int** *id*, **Bool** *flag*)

If *flag* is TRUE, enables the given menu item, else disables it (greys it). Only use this when the menu bar has been associated with a frame; otherwise, use the wxMenu equivalent call.

## wxMenuBar::Check

**void Check(int** *id*, **Bool** *flag*)

If *flag* is TRUE, checks the given menu item, else unchecks it. Works in Windows but has no effect in XView or Motif. Only use this when the menu bar has been associated with a frame; otherwise, use the wxMenu equivalent call.

# 5.29 wxMessage: wxItem

A message is a simple line of text which may be displayed in a panel. It does not respond to mouse clicks.

## wxMessage::wxMessage

**void wxMessage(wxPanel** **panel*, **char** **message*, **int** *x = -1*, **int** *y = -1*)

Creates and displays the message at the given coordinate.

## wxMessage::~wxMessage

**void ~wxMessage(void)**

Destroys the message.

## 5.30 wxMetaFile: wxObject

A wxMetaFile represents the Windows 3 metafile object, so metafile operations have no effect in X. In wxWindows, only sufficient functionality has been provided for copying a graphic to the clipboard; this may be extended in a future version. Presently, the only way of creating a metafile is to use a wxMetafileDC.

### wxMetaFile::wxMetaFile

**void wxMetaFile**(**void**)

Constructor.

### wxMetaFile::~wxMetaFile

**void ~wxMetaFile**(**void**)

Destructor.

### 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.

## 5.31 wxMetaFileDC: wxDC

This is a type of device context that allows a metafile object to be created (Windows only), and has most of the characteristics of a normal wxDC. The **Close** 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**).

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).

### wxMetaFileDC::wxMetaFileDC

**void wxMetaFileDC**(**char** **filename = NULL*)

Constructor. If no filename is passed, the metafile is created in memory.

### wxMetaFileDC::~wxMetaFileDC

**void ~wxMetaFileDC**(**void**)

Destructor.

## wxMetaFileDC::Close

**wxMetaFile * Close**(**void**)

This must be called after the device context is finished with. A metafile is returned, and ownership of it passes to the calling application (so it should be destroyed explicitly).

# 5.32 wxMultiText: wxText

Members as for **wxText**, but allowing multiple lines of text.

## wxText::GetValue

**char * GetValue**(**void**) **void GetValue**(**char ***_buffer_, **int** _bufferSize_)

The first form gets a pointer to the current value – this is allocated using _new_, so should be deleted by the calling program. The second form copies the value into a buffer, for situations where a lot of text is returned (more than the capacity of the small buffer used for the first form).

# 5.33 wxNode: wxObject

A node structure used in linked lists (see **wxList**).

## wxNode::Data

**wxObject * Data**(**void**)

Retrieves the client data pointer associated with the node. This will have to be cast to the correct type.

## wxNode::Next

**wxNode * Next**(**void**)

Retrieves the next node (NULL if at end of list).

## wxNode::Previous

**wxNode * Previous**(**void**)

Retrieves the previous node (NULL if at start of list).

## wxNode::SetData

**void SetData(void)**

Sets the data associated with the node (usually the pointer will have been set when the node was created).

## 5.34 wxObject

This is the root class of all wxWindows classes, and has no base functionality. It declares a virtual destructor which ensures that destructors get called for all derived class objects where necessary.

## 5.35 wxPanel: wxWindow

A panel is a subwindow of a frame in which *panel items* can be placed to allow the user to view and set controls. Panel items include messages, text items, list items, and check boxes. Use **Fit** to fit the panel around its items.

### wxPanel::wxPanel

**void wxPanel(wxFrame** ***parent**, **int** *x = -1*, **int** *y = -1*, **int** *width = -1*, **int** *height = -1*, **int** *style = 0*)

Constructor. Set style to wxBORDER to draw a thin border in Windows 3.

### wxPanel::~wxPanel

**void ~wxPanel(void)**

Destructor. Deletes any panel items before deleting the physical window.

### wxPanel::GetCursor

**void GetCursor(int** ***x**, **int** ***y**)

Gets the current panel 'cursor' position, i.e. where the next panel item will be placed.

### wxPanel::GetHorizontalSpacing

**int GetHorizontalSpacing(void)**

Gets the horizontal spacing for placing items on a panel.

### wxPanel::GetVerticalSpacing

**int GetVerticalSpacing(void)**

Gets the vertical spacing for placing items on a panel.

## wxPanel::NewLine

**void NewLine**(**void**)

Cause the next item to be positioned at the beginning of the next line, using the current vertical spacing. More than one new line in succession causes extra vertical spacing to be inserted.

## wxPanel::SetHorizontalSpacing

**void SetHorizontalSpacing**(**int** *sp*)

Sets the horizontal spacing for placing items on a panel.

## wxPanel::SetLabelPosition

**void SetLabelPosition**(**int** *position*)

Determines the current method of placing labels on panel items: if *position* is `wxHORIZONTAL`, labels are placed to the left of the item value. If *position* is `wxVERTICAL`, the label is placed above the item value. The default behaviour is to have horizontal label placing.

Under Windows 3, this function words for **wxText**, **wxChoice** and **wxListBox**. Under XView, absolute positioning must be used for the wxVERTICAL position to work in some cases. This is because of some strange behaviour in XView where setting a horizontal layout orientation but a vertical label position causes items after list box to appear too low on the panel. So, where it is necessary to have vertical labels, use absolute positioning where results are not as expected.

## wxPanel::SetVerticalSpacing

**void SetVerticalSpacing**(**int** *sp*)

Sets the vertical spacing for placing items on a panel.

## wxPanel::Tab

**void Tab**(**int** *pixels*)

Tabs by the given number of pixels.

## 5.36 wxPathList: wxList

The path list is a convenient way of storing a number of directories, and when presented with a filename without a directory, searching for an existing file in those directories. Storing the filename only in an application's files and using a locally-defined list of directories makes the application and its files more portable.

Use the *FileNameFromPath* global function to extract the filename from the path.

## wxPathList::wxPathList

**void wxPathList**(**void**)

Constructor.

## wxPathList::AddEnvList

**void AddEnvList**(**char ***_env_variable_)

Finds the value of the given environment variable, and adds all paths to the path list. Useful for finding files in the PATH variable, for example.

## wxPathList::AddPath

**void AddPath**(**char ***_path_)

Adds the given directory to the path list.

## wxPathList::FindValidPath

**char * FindValidPath**(**char ***_file_)

Searches for a full path for an existing file by appending _file_ to successive members of the path list. If the file exists, a temporary pointer to the full path is returned.

## wxPathList::Member

**Bool Member**(**char ***_file_)

TRUE if the path is in the path list (ignoring case).

# 5.37 wxPen: wxObject

A pen is a drawing tool for drawing outlines. It is used for drawing lines and painting the outline of rectangles, ellipses, etc. It has a colour, a width and a style. On a monochrome display, the default behaviour is to show all non-white pens as black. To change this, set the **Colour** member of the device context to TRUE, and select appropriate colours.

The style may be one of wxSOLID, wxDOT, wxLONG_DASH, wxSHORT_DASH and wx-DOT_DASH. The names of these styles should be self explanatory.

Do not initialize objects on the stack before the program commences, since other required structures may not have been set up yet. Instead, define global pointers to objects and create them in _OnInit_ or when required.

An application may wish to dynamically create pens with different characteristics, and there is the consequent danger that a large number of duplicate pens will be created. Therefore an application may wish to get a pointer to a pen by using the global list of pens **wxThePenList**, and calling the member function **FindOrCreatePen**. See the entry for the **wxPenList** class.

## wxPen::wxPen

**void wxPen**(**void**)

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

**void wxPen**(**char ***colour_name*, **int** *style*)

Constructs a pen, uninitialized, initialized with a width, initialized with an RGB colour, a width and a style, or initialized using a colour name, a width and a style. If the named colour form is used, an appropriate **wxColour** structure is found in the colour database.

## wxPen::~wxPen

**void ~wxPen**(**void**)

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

## wxPen::GetColour

**wxColour& GetColour**(**void**)

Returns a reference to the pen colour.

## wxPen::GetStyle

**int GetStyle**(**void**)

Returns the pen style.

## wxPen::GetWidth

**int GetWidth**(**void**)

Returns the pen width.

## wxPen::SetColour

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

**void SetColour**(**char ***colour_name*)

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

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

## wxPen::SetStyle

**void SetStyle(int** *style***)**

Set the pen style (wxSOLID or wxTRANSPARENT).

## wxPen::SetWidth

**void SetWidth(int** *width***)**

Set the pen width.

# 5.38 wxPenList: wxList

A pen list is a list containing all pens which have been created. There is only one instance of this class: **wxThePenList**. Use this object to search for a previously created pen of the desired type and create it if not already found. In some windowing systems, the pen may be a scarce resource, so it is best to reuse old resources if possible. When an application finishes, all pens will be deleted and their resources freed, eliminating the possibility of 'memory leaks'.

## wxPenList::wxPenList

**void wxPenList(void)**

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

## wxPenList::AddPen

**void AddPen(wxPen** **pen***)**

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

## wxPenList::FindOrCreatePen

**wxPen * FindOrCreatePen(wxColour** **colour*, **int** *width*, **int** *style***)**

**wxPen * FindOrCreatePen(char** **colour_name*, **int** *width*, **int** *style***)**

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

## wxPenList::RemovePen

**void RemovePen(wxPen** **pen***)**

Used by wxWindows to remove a pen from the list.

## 5.39 wxPoint: wxObject

A **wxPoint** is a useful data structure for graphics operations. It simply contains floating point $x$ and $y$ members.

### wxPoint::wxPoint

**void wxPoint**(**void**)

**void wxPoint**(**float** *x*, **float** *y*)

Create a point.

**float** *x*

**float** *y*

Members of the **wxPoint** object.

## 5.40 wxServer: wxIPCObject

A wxServer object represents the server part of a client-server DDE (Dynamic Data Exchange) conversation (available in *both* Windows and UNIX). See section 2.12.

### wxServer::wxServer

**void wxServer**(**void**)

Constructs a server object.

### wxServer::Create

**Bool Create**(**char** **service*)

Registers the server using the given service name. Under UNIX, the string must contain an integer id which is used as an Internet port number. FALSE is returned if the call failed (for example, the port number is already in use).

### wxServer::OnAcceptConnection

**wxConnection * OnAcceptConnection**(**char** **topic*)

When a client calls **MakeConnection**, the server receives the message and this member is called. The application should derive a member to intercept this message and return a connection object of either the standard wxConnection type, or of a user-derived type. If the topic is "STDIO", the application may wish to refuse the connection. Under UNIX, when a server is created the OnAcceptConnection message is always sent for standard input and output, but in the context of DDE messages it doesn't make a lot of sense.

## 5.41 wxSlider: wxItem

### wxSlider::wxSlider

A slider is, as its name suggests, an item with a handle which can be pulled back and forth to change a value. It is currently horizontal only. In Windows 3, a scrollbar is used to simulate the slider.

**void wxSlider(wxPanel** **parent*, **wxFunction** *func*, **char** **label*,
**int** *value*, **int** *min_value*, **int** *max_value*, **int** *width*, **int** *x = -1*, **int** *y = -1*)

Constructor, creating and showing a horizontal slider. The *width* is in pixels, and the scroll increment will be adjusted to a suitable value given the minimum and maximum integer values.

### wxSlider::~wxSlider

**void ~wxSlider(void)**

Destructor, destroying the slider.

### wxSlider::GetValue

**int GetValue(void)**

Gets the current slider value.

### wxSlider::SetValue

**void SetValue(int** *value*)

Sets the value (and displayed position) of the slider).

## 5.42 wxStringList: wxList

A string list is a list which is assumed to contain strings, with a specific member functions. Memory is allocated when strings are added to the list, and deallocated by the destructor or by the **Delete** member.

### wxStringList::wxStringList

**void wxStringList(void)**

Constructor.

**void wxStringList(char** **first*, ...)

Constructor, taking NULL-terminated string argument list. wxStringList allocates memory for the strings.

## wxStringList::~wxStringList

**void ~wxStringList(void)**

Deletes string list, deallocating strings.

## wxStringList::Add

**void Add(char** **s*)

Adds string to list, allocating memory.

## wxStringList::Delete

**void Delete(char** **s*)

Searches for string and deletes from list, deallocating memory.

## wxStringList::ListToArray

**char ** ListToArray(Bool** *new_copies = FALSE*)

Converts the list to an array of strings, only allocating new memory if **new_copies** is TRUE.

## wxStringList::Member

**Bool Member(char** **s*)

Returns TRUE if **s** is a member of the list (tested using **strcmp**).

## wxStringList::Sort

**void Sort(void)**

Sorts the strings in ascending alphabetical order. Note that all nodes (but not strings) get deallocated and new ones allocated.

## 5.43 wxText: wxWindow

## wxText::wxText

A text item is an area of editable text, with an optional label displayed in front of it.

**void wxText(wxPanel** **parent*, **wxFunction** *func*, **char** **label*,
**char** **value = ""*, **int** *x = -1*, **int** *y = -1*, **int** *width = -1*, **int** *height = -1*)

Constructor, creating and showing a text item with the given string value. If *width* or *height* are omitted (or are less than zero), an appropriate size will be used for the item. *func* may be NULL; otherwise it is used as the callback for the list box. Note that the cast (wxFunction) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration.

## wxText::~wxText

**void ~wxText(void)**

Destructor, destroying the text item.

## wxText::GetValue

**char * GetValue(void)**

Gets a pointer to the current value – this is allocated using *new*, so should be deleted by the calling program.

## wxText::SetValue

**void SetValue(char *** *value***)**

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

# 5.44 wxTextWindow: wxWindow

A text window is a subwindow of a frame, offering some basic ability to display scrolling text. At present, editing is only possible using the XView and Motif implementations.

## wxTextWindow::wxTextWindow

**void wxTextWindow(wxFrame ***parent*, **int** *x = -1*, **int** *y = -1*,
**int** *width = -1*, **int** *height = -1*, **int** *style = 0*)

Constructor. Set style to wxBORDER to draw a thin border in Windows 3.

## wxTextWindow::~wxTextWindow

**void ~wxTextWindow(void)**

Destructor. Deletes any stored text before deleting the physical window.

## wxTextWindow::Clear

**void Clear**(**void**)

Clears the window and deletes the stored text.

## wxTextWindow::DiscardEdits

**void DiscardEdits**(**void**)

Clears the window and deletes the stored text (same as **Clear**).

## wxTextWindow::LoadFile

**Bool LoadFile**(**char *** *file*)

Loads and displays the named file, if it exists. Success is indicated by a return value of TRUE.

## wxTextWindow::Modified

**Bool Modified**(**void**)

Returns TRUE if the text has been modified. Under Windows 3, this always returns FALSE.

## wxTextWindow::SaveFile

**Bool SaveFile**(**char *** *file*)

Saves the text in the named file. Success is indicated by a return value of TRUE.

## wxTextWindow::WriteText

**void WriteText**(**char *** *text*)

Writes the text into the text window. Presently there is no means of writing text to other than the end of the existing text. Newlines in the text string are the only control characters allowed, and they will cause appropriate line breaks. See the <<operators for more convenient ways of writing to the window.

## wxTextWindow::<<

**wxTextWindow& <<**(**char ***$s$)

**wxTextWindow& <<**(**int** $i$)

**wxTextWindow& <<**(**long** $i$)

**wxTextWindow& <<**(**float** $f$)

**wxTextWindow& <<**(**double** $d$)

**wxTextWindow& <<(char** *c***)**

Operator definitions for writing to a text window, for example:

```
wxTextWindow wnd(my_frame);

wnd << "Welcome to text window number " << 1 << ".\n";
```

## 5.45 wxTimer: wxObject

The wxTimer object is an abstraction of Windows 3, XView and X toolkit timers. To use it, derive a new class and override the **Notify** member to perform the required action. Start with **Start**, stop with **Stop**, it's as simple as that.

### wxTimer::wxTimer

**void wxTimer(void)**

Constructor.

### wxTimer::~wxTimer

**void ~wxTimer(void)**

Destructor. Stops the timer if activated.

### wxTimer::Notify

**void Notify(void)**

This member should be overridden by the user. It is called on timeout.

### wxTimer::Start

**Bool Start(int** *milliseconds = -1***)**

(Re)starts the timer. If *milliseconds* is absent or -1, the previous value is used. Returns FALSE if the timer could not be started, TRUE otherwise (in Windows 3 timers are a limited resource).

### wxTimer::Stop

**void Stop(void)**

Stops the timer.

## 5.46 wxWindow: wxObject

wxWindow is the base class for all windows and panel items. Any children of the window will be deleted automatically by the destructor before the window itself is deleted.

## wxWindow::wxWindow

**void wxWindow(void)**

Constructor.

## wxWindow::~wxWindow

**void ~wxWindow(void)**

Destructor. Deletes all subwindows, then deletes itself.

## wxWindow::AddChild

**void AddChild(wxWindow** **child*)

Adds a child window. This is called automatically by window creation functions so should not be required by the application programmer.

## wxWindow::Center

**void Center(int** *direction*)

See **Centre**.

## wxWindow::Centre

**void Centre(int** *direction*)

Centres the window. The parameter may be `wxHORIZONTAL`, `wxVERTICAL` or `wxBOTH`.

The actual behaviour depends on the derived window. For a frame or dialog box, centring is relative to the whole display. For a panel item, centring is relative to the panel.

## wxWindow::DestroyChildren

**void DestroyChildren(void)**

Destroys all children of a window. Called automatically by the destructor.

## wxWindow::Fit

**void Fit**(**void**)

Sizes the window to fit the content (for panels and frames).

## wxWindow::GetClientData

**char * GetClientData**(**void**)

Gets user-supplied client data. Normally, any extra data the programmer wishes to associate with the window should be made available by deriving a new class with new data members.

## wxWindow::GetClientSize

**void GetClientSize**(**int** ***width**, **int** ***height**)

This gets the size of the window 'client area' in pixels. The client area is the area which may be drawn on by the programmer, excluding title bar, border etc.

## wxWindow::GetHandle

**char * GetHandle**(**void**)

Gets the platform-specific handle of the physical window.

## wxWindow::GetPosition

**void GetPosition**(**int** **x*, **int** **y*)

This gets the position of the window in pixels, relative to the parent window or if no parent, relative to the whole display.

## wxWindow::GetSize

**void GetSize**(**int** **width*, **int** **height*)

This gets the size of the entire window in pixels.

## wxWindow::GetTextExtent

**void GetTextExtent**(**char** **string*, **int** **x*, **int** **y*)

Gets the width and height of the string as it would be drawn on the window with the currently selected font.

## wxWindow::OnActivate

**void OnActivate(Bool** *active***)**

Called when a window is activated or deactivated (Windows 3 only). If the window is being activated, *active* is TRUE, else it is FALSE.

# wxWindow::OnChar

**void OnChar(int** *ch***)**

Sent to the window when the user has pressed a key. *ch* gives the ASCII code (function key identifiers not yet implemented). See **OnEvent** for mouse event notification. Currently applicable to canvas subwindows only.

# wxWindow::OnClose

**Bool OnClose(void)**

Sent to the window when the user has tried to close the window. If TRUE is returned, the window will be deleted by the system, otherwise the attempt will be ignored. Derive your own class to handle this message; the default handler returns TRUE. Really only relevant to wxFrames.

# wxWindow::OnEvent

**void OnEvent(wxEvent&** *event***)**

Sent to the window when the user has initiated an event with the mouse. Derive your own class to handle this message. So far, only relevant to the wxCanvas class. See **OnChar** for character events, and also **wxEvent** for how to access event information.

# wxWindow::OnKillFocus

**void OnKillFocus(void)**

Called when a window's focus is being killed.

# wxWindow::OnPaint

**void OnPaint(void)**

Sent to the window when the window must be refreshed. Derive your own class to handle this message. So far, only relevant to the wxCanvas class.

# wxWindow::OnSetFocus

**void OnSetFocus(void)**

Called when a window's focus is being set.

## wxWindow::OnSize

**void OnSize(int** *x*, **int** *y*)

Sent to the window when the window has been resized. You may wish to use this for frames to resize their child windows as appropriate. Derive your own class to handle this message. Note that the size passed is of the whole window: call **GetClientSize** for the area which may be used by the application. A window is sent both an OnPaint and an OnSize message when a resize occurs.

## wxWindow::SetFocus

**void SetFocus(void)**

This sets the window to receive keyboard input. The only panel item that will respond to this under XView is the **wxText** item and derived items.

## wxWindow::SetSize

**void SetSize(int** *x*, **int** *y*, **int** *width*, **int** *height*)

This sets the size of the entire window in pixels.

## wxWindow::SetClientData

**void SetClientData(char ***\ *data*)

Sets user-supplied client data. Normally, any extra data the programmer wishes to associate with the window should be made available by deriving a new class with new data members.

## wxWindow::SetClientSize

**void SetClientSize(int** *x*, **int** *y*, **int** *width*, **int** *height*)

This sets the size of the window client area in pixels. Using this function to size a window tends to be more device-independent than **SetSize**, since the application need not worry about what dimensions the border or title bar have when trying to fit the window around panel items, for example.

## wxWindow::SetCursor

**wxCursor * SetCursor(wxCursor ***\ *cursor*)

Sets the window's cursor, returning the previous cursor (if any). This function applies to all subwindows.

## wxWindow::SetTitle

**void SetTitle(char ****title*)

Sets the window's title, allocating its own string storage. Currently applicable only to frames.

## wxWindow::Show

**void Show(Bool** *show*)

If *show* is TRUE, displays the window and brings it to the front. Otherwise, hides the window.

# Chapter 6

# Miscellaneous functions

## 6.1 File functions

See also the **wxPathList** class.

### ::Dos2UnixFilename

**void Dos2UnixFilename(char ****s*)

Converts a DOS to a UNIX filename by replacing backslashes with forward slashes.

### ::FileExists

**Bool FileExists(char ****filename*)

Returns TRUE if the file exists.

### ::FileNameFromPath

**char * FileNameFromPath(char ****path*)

Returns the filename for a full path (returns a new string).

### ::IsAbsolutePath

**Bool IsAbsolutePath(char ****filename*)

Returns TRUE if the argument is an absolute filename, i.e. with a slash or drive name at the beginning.

### ::PathOnly

**char * PathOnly**(**char** **path*)

Returns the directory part of the filename (returns a new string).

# ::Unix2DosFilename

**void Unix2DosFilename**(**char** **s*)

Converts a UNIX to a DOS filename by replacing forward slashes with backslashes.

# ::wxConcatFiles

**Bool wxConcatFiles**(**char** **file1*, **char** **file2*, **char** **file3*)

Concatenates *file1* and *file2* to *file3*, returning TRUE if successful.

# ::wxCopyFile

**Bool wxCopyFile**(**char** **file1*, **char** **file2*)

Copies *file1* to *file2*, returning TRUE if successful.

# ::wxIsWild

**Bool wxIsWild**(**char** **pattern*)

Returns TRUE if the pattern contains wildcards. See **wxMatchWild**.

# ::wxMatchWild

**Bool wxMatchWild**(**char** **pattern*, **char** **text*, **Bool** *dot_special*)

Returns TRUE if the *pattern* matches the *text*; if *dot_special* is TRUE, filenames beginning with a dot are not matched with wildcard characters. See **wxIsWild**.

# ::wxRemoveFile

**Bool wxRemoveFile**(**char** **file*)

Removes *file*, returning TRUE if successful.

# ::wxRenameFile

**Bool wxRenameFile**(**char** **file1*, **char** **file2*)

Renames *file1* to *file2*, returning TRUE if successful.

## 6.2 String functions

### ::copystring

**char * copystring(char** **s*)

Makes a copy of the string *s* using the C++ new operator, so it can be deleted with the delete operator.

## 6.3 Dialog functions

Below are a number of convenience functions for getting input from the user or displaying messages. Note that in these functions the last three parameters are optional. However, it is recommended to pass a parent frame parameter, or (in Windows 3) the wrong window frame may be brought to the front when the dialog box is popped up.

### ::wxGetTextFromUser

**char * wxGetTextFromUser(char** **message*, **char** **caption = "Input text"*,
**char** **default_value = ""*, **wxFrame** **parent = NULL*, **int** *x = -1*, **int** *y = -1*)

Pop up a dialog box with title set to *caption*, message *message*, and a *default_value*. The user may type in text and press OK to return this text, or press Cancel to return NULL.

### ::wxGetSingleChoice

**char * wxGetSingleChoice(char** **message*, **char** **caption*, **int** *n*, **char** **choices[]*,
**wxFrame** **parent = NULL*, **int** *x = -1*, **int** *y = -1*)

Pops up a dialog box containing a message, OK/Cancel buttons and a single-selection listbox. The user may choose an item and press OK to return a string or Cancel to return NULL.

*choices* is an array of *n* strings for the listbox.

### ::wxGetSingleChoiceIndex

**int wxGetSingleChoiceIndex(char** **message*, **char** **caption*, **int** *n*, **char** **choices[]*,
**wxFrame** **parent = NULL*, **int** *x = -1*, **int** *y = -1*)

As **wxGetSingleChoice** but returns the index representing the selected string.

### ::wxGetSingleChoiceData

**char * wxGetSingleChoiceData(char** **message*, **char** **caption*, **int** *n*, **char** **choices[]*,
**char** **client_data[]*, **wxFrame** **parent = NULL*, **int** *x = -1*, **int** *y = -1*)

As **wxGetSingleChoice** but takes an array of client data pointers corresponding to the strings, and returns one of these pointers.

## ::wxMessageBox

**int wxMessageBox(char** **message*, **char** **caption = "Message"*, **int** *type = wxOK*,
**wxFrame** **parent = NULL*, **int** *x = -1*, **int** *y = -1*)

General purpose message dialog. *type* may be one or more of the following identifiers or'ed together: wxYES_NO, wxCANCEL, wxOK.

The return value is one of: wxYES, wxNO, wxCANCEL, wxOK.

For example:

```
...
int answer = wxMessageBox("Quit program?", "Confirm",
                          wxYES_NO | wxCANCEL, main_frame);
if (answer == wxYES)
  delete main_frame;
...
```

*message* may contain newline characters, in which case the message will be split into separate lines and centred in the dialog box, to cater for large messages.

## ::wxFileSelector

**char * wxFileSelector(char** **message*, **char** **default_path = NULL*,
**char** **default_filename = NULL*, **char** **default_extension = NULL*,
**char** **wildcard = "*.*"*, **int** *flags = 0*, **wxFrame** **parent = NULL*,
**int** *x = -1*, **int** *y = -1*)

Pops up a file selector box. In Windows, this is the common file selector dialog. In X, this is a file selector box with somewhat less functionality. The path and filename are distinct elements of a full file pathname. If path is NULL, the current directory will be used. If filename is NULL, no default filename will be supplied. The wildcard determines what files are displayed in the file selector, and file extension supplies a type extension for the required filename. Flags may be a combination of wxOPEN, wxSAVE, wxOVERWRITE_PROMPT, wxHIDE_READONLY, or 0. They are only significant at present in Windows.

Both the X and Windows versions implement a wildcard filter. Typing a filename containing wildcards (*, ?) in the filename text item, and clicking on Ok, will result in only those files matching the pattern being displayed. In the X version, supplying no default name will result in the wildcard filter being inserted in the filename text item; the filter is ignored if a default name is supplied.

The application must check for a NULL return value (the user pressed Cancel). For example:

```
char *s = wxFileSelector("Choose a file to open");
if (s)
{
  ...
}
```

## 6.4 GDI functions

The following are relevant to the GDI (Graphics Device Interface).

### ::wxColourDisplay

**Bool wxColourDisplay(void)**

Returns TRUE if the display is colour, FALSE otherwise.

### ::SetCursor

**void wxSetCursor(wxCursor ***_cursor_)

Globally sets the cursor; only works in Windows 3.

## 6.5 Miscellaneous

### ::NewId

**long NewId(void)**

Generates an integer identifier unique to this run of the program.

### ::RegisterId

**void RegisterId(long** _id_)

Ensures that ids subsequently generated by **NewId** do not clash with the given **id**.

### ::wxDisplaySize

**void wxDisplaySize(int ***_width_, **int ***_height_)

Gets the physical size of the display in pixels.

### ::wxGetElapsedTime

**long wxGetElapsedTime(void)**

Gets the time in milliseconds since the last **wxGetElapsedTime** or **wxStartTimer**.

### ::wxSleep

**void wxSleep(int** *secs***)**

Under X, sleeps for the specified number of seconds using the technique specified in the XView manual, not UNIX **sleep**.

# ::wxStartTimer

**void wxStartTimer(void)**

Starts a stopwatch; use **wxGetElapsedTime** to get the elapsed time.

# ::wxYield

**void wxYield(void)**

Yields control to other applications (has no effect under XView or Motif).

# ::wxExecute

**void wxExecute(char *** *command***)**

Executes a command in UNIX or Windows. Note that under Windows, the function immediately returns, whereas under UNIX, it does returns only after the command has finished executing, unless the command is executed in the background (append an ampersand).

# Glossary

**API** Application Programmer's Interface - a set of calls and classes defining how a library (in this case, wxWindows) can be used.

**Canvas** A canvas in XView and wxWindows is a subwindow on which graphics (but not panel items) can be drawn. It may be scrollable. A canvas has a *device context* associated with it.

**DDE** Dynamic Data Exchange - Microsoft's interprocess communication protocol. wxWindows provides an abstraction of DDE under both Windows and UNIX.

**Device context** A device context is an abstraction away from devices such as windows, printers and files. Code that draws to a device context is generic since that device context could be associated with a number of different real device. A canvas has a device context, although duplicate graphics calls are provided for the canvas, so the beginner doesn't have to think in terms of device contexts when starting out. wxWindows supports device contexts for canvas, Windows printer, and Encapsulated PostScript files on UNIX.

**Dialog box** In wxWindows a dialog box is a convenient way of popping up a window with panel items, without having to explicitly create a frame and a panel. A dialog box may be modal or modeless. A modal dialog does not return control back to the calling program until the user has dismissed it, and all other windows in the application are disabled until the dialog is dismissed. A modeless dialog is just like a normal window in that the user can access other windows while the dialog is displayed.

**Frame** Under XView (and wxWindows), a visible window usually consists of a frame which contains zero or more subwindows, such as text subwindow, canvas, and panel. Under Windows 3, windows can be nested arbitrarily, but this is not currently supported in wxWindows.

**GNU C++** A free, solid C++ compiler which may be used to compile the UNIX side of wxWindows applications.

**GUI** Graphical User Interface, such as Windows 3 or X.

**Menu bar** A menu bar is a series of labelled menus, usually placed near the top of a window. It is popular in Windows 3 and Motif applications, and as such is a supported feature, but wxWindows has to 'simulate' the menu bar under XView by using a panel and several menu buttons.

**Metafile** Microsoft Windows-specific object which may contain a restricted set of GDI primitives. It is device independent, since it may be scaled without losing precision, unlike a bitmap. A metafile may exist in a file or in memory. wxWindows implements enough metafile functionality to use it to pass graphics to other applications via the clipboard.

**Open Look** A specification for a GUI 'look and feel', initiated by Sun Microsystems. XView is one toolkit for writing Open Look applications under X, and wxWindows sits on top of XView.

**Panel** A panel in XView and wxWindows terminology is a subwindow on which a limited range of panel items (widgets or controls for user input) can be placed. wxWindows allows panel items to be placed explicitly, or laid out from left to right, top to bottom, which is a more platform independent method since spacing is calculated automatically at run time. Panel items cannot be placed on a canvas, which is specifically for drawing graphics.

**RPC** Remote Procedure Call - a method of interprocess communication akin to procedure call, where the client process makes a call to a server, which sends back a result. The AIAI-supplied PROLOGIO library supports a simple RPC protocol based on DDE (but working under both UNIX and Windows).

**Status line** A status line is often found at the base of a window, to keep the user informed (for instance, giving a line of description to menu items, as in the **hello** demo). XView has a status line (or footer) capability, but wxWindows implements the feature explicitly under Windows 3 and Motif.

**Text subwindow** In XView and Motif, a text subwindow is supplied for displaying and retrieving text from a window. It has a rich set of features, only a small subset of which is currently catered for by wxWindows.

**XView** An X toolkit supplied by Sun Microsystems, initially just for porting SunView applications to X, but which has become a popular toolkit in its own right due to its simplicity of use. XView implements Sun's Open Look 'look and feel' for X, but is not the only toolkit to do so.

# Index


<<, 92, 93
~wxApp, 30
~wxBrush, 32
~wxButton, 34
~wxCanvas, 34
~wxCheckBox, 39
~wxChoice, 40
~wxCursor, 47
~wxDC, 48
~wxDialogBox, 55
~wxEvent, 56
~wxFont, 59
~wxForm, 61
~wxFrame, 65
~wxHashTable, 69
~wxIcon, 68
~wxList, 74
~wxListBox, 76
~wxMenu, 78
~wxMenuBar, 80
~wxMessage, 80
~wxMetaFile, 81
~wxMetaFileDC, 81
~wxPanel, 83
~wxPen, 86
~wxSlider, 89
~wxStringList, 90
~wxText, 91
~wxTextWindow, 91
~wxTimer, 93
~wxWindow, 94

Add, 61, 90
AddBrush, 33
AddChild, 94
AddEnvList, 85
AddPath, 85
AddPen, 87
Advise, 44
API, 105
Append, 40, 74, 76, 77, 79, 80
AppendSeparator, 79
AssociatePanel, 62

BeginFind, 69
Blit, 49
Button, 56
ButtonDown, 56

Canvas, 105
Center, 94
Centre, 65, 72, 94
Check, 79, 80
Clear, 35, 40, 49, 69, 74, 77, 92
Close, 82
ControlDown, 57
copystring, 101
Create, 88
CreateCompatibleDC, 49
CreateStatusLine, 65

Data, 82
DDE, 105
Delete, 62, 69, 90
DeleteContents, 74
DeleteNode, 75
DeleteObject, 75
Deselect, 77
DestroyChildren, 94
DestroyClippingRegion, 35, 49
Device context, 105
DeviceToLogicalX, 49
DeviceToLogicalY, 49
Dialog box, 105
DiscardEdits, 92
Disconnect, 44
DisplayBlock, 71
DisplayContents, 71
DisplaySection, 71
Dos2UnixFilename, 99
Dragging, 57
DrawEllipse, 35, 50
DrawIcon, 50
DrawLine, 35, 50
DrawLines, 35, 50
DrawPoint, 36, 50
DrawPolygon, 35, 50
DrawRectangle, 36, 51
DrawRoundedRectangle, 36, 51
DrawSpline, 36, 51

DrawText, 36, 51

Enable, 79, 80
EndDoc, 51
EndPage, 51
Execute, 44

FileExists, 99
FileNameFromPath, 99
Find, 75
FindColour, 43
FindItem, 62
FindName, 43
FindOrCreateBrush, 33
FindOrCreatePen, 87
FindString, 40, 77
FindValidPath, 85
First, 75
Fit, 95
Frame, 105

Get, 42, 69, 70
GetClientData, 77, 95
GetClientSize, 95
GetColour, 32, 86
GetCursor, 83
GetDC, 36
GetHandle, 95
GetHorizontalSpacing, 83
GetMapMode, 51
GetPointSize, 59
GetPosition, 95
GetSelection, 41, 77
GetSelections, 77
GetSize, 95
GetStringSelection, 41, 77
GetStyle, 32, 86
GetTextExtent, 95
GetValue, 39, 82, 89, 91
GetVerticalSpacing, 83
GetWidth, 86
GNU C++, 105
GUI, 105

Iconize, 55, 66
Iconized, 55, 66
Initialize, 43, 71
Initialized, 31
Insert, 75
IsAbsolutePath, 99
IsButton, 57

KeywordSearch, 71

Last, 75
LeftDown, 57
LeftUp, 57
ListToArray, 90
LoadFile, 72, 92
LogicalToDeviceX, 52
LogicalToDeviceY, 52

MainLoop, 31
MakeConnection, 42
MakeKey, 70
Member, 75, 85, 90
Menu bar, 105
Metafile, 105
MiddleDown, 57
MiddleUp, 57
Modified, 92

NewId, 103
NewLine, 84
Next, 70, 82
Notify, 93
Nth, 76
Number, 76

Ok, 52
OnAcceptConnection, 88
OnActivate, 96
OnAdvise, 44
OnCancel, 62
OnChar, 96
OnClose, 96
OnDisconnect, 45
OnEvent, 96
OnExecute, 45
OnExit, 31
OnInit, 31
OnKillFocus, 96
OnMakeConnection, 42
OnMenuCommand, 66
OnMenuSelect, 66
OnOk, 62
OnPaint, 96
OnPoke, 45
OnQuit, 72
OnRequest, 45
OnRevert, 63
OnSetFocus, 96
OnSize, 97
OnStartAdvise, 45
OnStopAdvise, 45
OnUpdate, 63
Open Look, 105

Panel, 106

PathOnly, 100
Poke, 45
Position, 58
Previous, 82
Put, 70

Quit, 72

RegisterId, 103
RemoveBrush, 33
RemovePen, 87
Request, 46
ResetContext, 37
RevertValues, 63
RightDown, 58
RightUp, 58
RPC, 106

SaveFile, 92
Scroll, 37
SelectObject, 52
Set, 42, 62, 78
SetBackground, 37, 52
SetBrush, 37, 52
SetClientData, 97
SetClientSize, 97
SetClipboard, 81
SetClippingRegion, 37, 52
SetColour, 33, 86
SetContext, 37
SetCursor, 97
SetData, 83
SetDefault, 72
SetFocus, 97
SetFont, 37, 53
SetHorizontalSpacing, 84
SetIcon, 66
SetLabelPosition, 84
SetLogicalFunction, 38, 53
SetMapMode, 53
SetMenuBar, 67
SetPen, 38, 54
SetScrollbars, 38
SetSelection, 41, 78
SetSize, 97
SetStatusText, 67
SetStringSelection, 41, 78
SetStyle, 33, 87
SetTextBackground, 38, 54
SetTextForeground, 39, 54
SetTitle, 98
SetUserScale, 54
SetValue, 39, 89, 91
SetVerticalSpacing, 84
SetWidth, 87
ShiftDown, 58
Show, 56, 98
Sort, 90
Start, 93
StartAdvise, 46
StartDoc, 54
StartPage, 54
Status line, 106
StatusLineExists, 67
Stop, 93
StopAdvise, 46
String, 41, 78

Tab, 84
Text subwindow, 106

Unix2DosFilename, 100
UpdateValues, 63

ViewStart, 39

WriteText, 92
wxApp, 30
wxBrush, 32
wxBrushList, 33
wxButton, 34
wxCanvas, 34
wxCheckBox, 39
wxChoice, 40
wxClient, 41
wxColour, 42
wxColourDatabase, 42, 43
wxColourDisplay, 103
wxConcatFiles, 100
wxConnection, 43, 44
wxConstraintFunction, 64
wxCopyFile, 100
wxCursor, 46
wxDC, 48
wxDialogBox, 54, 55
wxDisplaySize, 103
wxEvent, 56
wxExecute, 104
wxFileSelector, 102
wxFont, 58, 59
wxForm, 59, 61
wxFrame, 65
wxFunction, 67
wxGetElapsedTime, 103
wxGetSingleChoice, 101
wxGetSingleChoiceData, 101
wxGetSingleChoiceIndex, 101
wxGetTextFromUser, 101

wxHashTable, 68, 69
wxHelpInstance, 70, 71
wxIcon, 67, 68
wxIsWild, 100
wxItem, 72, 73
wxList, 73, 74
wxListBox, 76
wxMakeConstraintFunction, 64
wxMakeConstraintRange, 64
wxMakeConstraintStrings, 64
wxMakeFormBool, 64
wxMakeFormButton, 63
wxMakeFormFloat, 64
wxMakeFormLong, 63
wxMakeFormMessage, 63
wxMakeFormNewLine, 63
wxMakeFormShort, 64
wxMakeFormString, 64
wxMatchWild, 100
wxMenu, 78
wxMenuBar, 79
wxMessage, 80
wxMessageBox, 102
wxMetaFile, 81
wxMetaFileDC, 81
wxMultiText, 82
wxNode, 82
wxObject, 83
wxPanel, 83
wxPathList, 85
wxPen, 85, 86
wxPenList, 87
wxPoint, 88
wxRemoveFile, 100
wxRenameFile, 100
wxServer, 88
wxSetCursor, 103
wxSleep, 104
wxSlider, 89
wxStartTimer, 104
wxStringList, 89
wxText, 90
wxTextWindow, 91
wxTimer, 93
wxWindow, 94
wxYield, 104

XView, 106