wxRegisterExternalEventHandlers, 127
wxRemoveFile, 123
wxRemoveSecondaryEventHandler, 128
wxRenameFile, 123
wxSendEvent, 128
wxServer, 105
wxSetCursor, 126
wxSleep, 131
wxSlider, 106
wxStartTimer, 131
wxStringList, 107
wxStripMenuCodes, 131
wxSubType, 132
wxText, 108
wxTextWindow, 109
wxTimer, 111, 112
wxTypeTree, 112, 114
wxWindow, 115
wxWriteResource, 132
wxYield, 132

StatusLineExists, 66
Stop, 112
StopAdvise, 38
StringMatch, 123

Tab, 97

Unix2DosFilename, 122
UpdateValues, 58

ViewStart, 24

WarpPointer, 25
WriteEvent, 51
WriteText, 111
wxAddPrimaryEventHandler, 126
wxAddSecondaryEventHandler, 127
wxApp, 10
wxBell, 128
wxBitmap, 12, 13
wxBrush, 14
wxBrushList, 15
wxButton, 16
wxCanvas, 17, 18
wxCanvasDC, 25
wxChoice, 27
wxClient, 29, 30
wxColour, 31
wxColourDatabase, 32
wxColourDisplay, 126
wxColourMap, 33
wxCommandEvent, 33, 34
wxConcatFiles, 122
wxConnection, 35
wxConstraintFunction, 59
wxCopyFile, 122
wxCursor, 38
wxDC, 39, 40
wxDebugMsg, 128
wxDialogBox, 48
wxDisplayDepth, 126
wxDisplaySize, 129
wxError, 129
wxEvent, 50
wxExecute, 129
wxExit, 129
wxFatalError, 129
wxFileSelector, 123
wxFindMenuItemId, 129
wxFindWindowByLabel, 130
wxFont, 52
wxFontList, 53
wxForm, 54, 56
wxFrame, 60
wxFunction, 66
wxGetElapsedTime, 130
wxGetFreeMemory, 130
wxGetHostName, 130
wxGetResource, 130
wxGetSingleChoice, 124
wxGetSingleChoiceData, 125
wxGetSingleChoiceIndex, 125
wxGetTempFileName, 122
wxGetTextFromUser, 124
wxGetUserId, 131
wxGetUserName, 131
wxHashTable, 67, 68
wxHelpInstance, 69, 70
wxIcon, 66, 67
wxIPCInitialize, 131
wxIsWild, 122
wxItem, 71, 72
wxKeyEvent, 72, 73
wxList, 73, 74
wxListBox, 77
wxMakeConstraintFunction, 59
wxMakeConstraintRange, 60
wxMakeConstraintStrings, 59
wxMakeFormBool, 59
wxMakeFormButton, 58
wxMakeFormFloat, 59
wxMakeFormLong, 58
wxMakeFormMessage, 58
wxMakeFormNewLine, 58
wxMakeFormShort, 59
wxMakeFormString, 59
wxMatchWild, 122
wxMemoryDC, 80, 81
wxMenu, 81
wxMenuBar, 83, 84
wxMessage, 86
wxMessageBox, 125
wxMetaFile, 87
wxMetaFileDC, 87, 88
wxMouseEvent, 88, 89
wxMultiText, 93
wxNode, 93
wxNotifyEvent, 127
wxObject, 94
wxPanel, 94
wxPathList, 98
wxPen, 99
wxPenList, 100
wxPoint, 101
wxPostScriptDC, 101, 102
wxRadioBox, 102, 103
wxRegisterEventClass, 127
wxRegisterEventName, 127

NewId, 128
NewLine, 96
Next, 69, 93
Notify, 112
Nth, 76
Number, 76, 104

Ok, 13, 31, 45
OnAcceptConnection, 105
OnActivate, 118
OnAdvise, 36
OnCancel, 57
OnChar, 21, 118
OnClose, 62
OnDefaultAction, 96
OnDisconnect, 36
OnDropFiles, 118
OnEvent, 21, 118
OnExecute, 36
OnExit, 12
OnInit, 12
OnKillFocus, 119
OnMakeConnection, 31
OnMenuCommand, 63
OnMenuSelect, 64
OnOk, 57
OnPaint, 22, 119
OnPoke, 37
OnQuit, 71
OnRequest, 37
OnRevert, 58
OnSetFocus, 119
OnSize, 64, 119
OnStartAdvise, 37
OnStopAdvise, 37
OnUpdate, 58

PathOnly, 122
Poke, 37
Position, 92
Previous, 93
Put, 69

Quit, 71

ReadEvent, 51
RegisterId, 128
ReleaseMouse, 119
RemoveBrush, 16
RemoveFont, 54
RemovePen, 101
Request, 37
RevertValues, 58
RightDown, 92
RightIsDown, 92
RightUp, 92

SaveFile, 111
Scroll, 22
Selected, 79
SelectObject, 81
Set, 31, 57, 80
SetBackground, 22, 45
SetBackgroundMode, 45
SetBrush, 22, 45
SetButtonFont, 97
SetClientData, 120
SetClientSize, 120
SetClipboard, 87
SetClippingRegion, 22, 45
SetColour, 15, 100
SetColourMap, 120
SetCursor, 120
SetData, 93
SetDefault, 17
SetFocus, 119
SetFont, 23, 45
SetHelpString, 83, 85
SetHorizontalSpacing, 97
SetIcon, 65
SetLabel, 17, 26, 83, 85
SetLabelFont, 97
SetLabelPosition, 97
SetLabelTop, 86
SetLogicalFunction, 23, 46
SetMapMode, 46
SetMenuBar, 65
SetPen, 23, 47
SetScrollbars, 24
SetSelection, 28, 80, 104
SetSize, 119
SetStatusText, 65
SetStringSelection, 29, 80, 105
SetStyle, 15, 100
SetTextBackground, 24, 47
SetTextForeground, 24, 47
SetTitle, 50, 66, 83, 120
SetUserScale, 47
SetValue, 26, 107, 109
SetVerticalSpacing, 97
SetWidth, 100
ShiftDown, 73, 92
Show, 50, 120
Sort, 108
Start, 112
StartAdvise, 38
StartDoc, 47
StartPage, 48

DrawPoint, 20, 42
DrawPolygon, 19, 42
DrawRectangle, 20, 42
DrawRoundedRectangle, 20, 42
DrawSpline, 20, 42, 43
DrawText, 20, 43

Enable, 82, 84, 104, 116
EnableScrolling, 20
EnableTop, 84
EndDoc, 43
EndPage, 43
Execute, 36

FileExists, 121
FileNameFromPath, 121
Find, 76
FindColour, 32
FindItem, 57, 82
FindItemById, 82, 85
FindMenuItem, 85
FindName, 32
FindOrCreateBrush, 15
FindOrCreateFont, 54
FindOrCreatePen, 101
FindString, 28, 79, 104
FindValidPath, 98
First, 76
Fit, 62, 95

Get, 31, 69
GetButtonFont, 95
GetCharHeight, 116
GetCharWidth, 116
GetChildren, 116
GetClientData, 34, 79, 116
GetClientSize, 117
GetColour, 14, 99
GetCursor, 95
GetDC, 21
GetDefaultItem, 96
GetDepth, 13
GetFamily, 53
GetGrandParent, 117
GetHandle, 117
GetHeight, 13
GetHelpString, 83, 85
GetHorizontalSpacing, 96
GetLabel, 83, 85, 117
GetLabelFont, 96
GetLabelTop, 85
GetMapMode, 43
GetName, 114
GetParent, 117
GetPointSize, 53
GetPosition, 117
GetScrollUnitsPerPage, 21
GetSelection, 28, 35, 79, 104
GetSelections, 79
GetSize, 43, 117
GetString, 29, 35, 80, 105
GetStringSelection, 28, 79, 104
GetStyle, 14, 53, 99
GetTextExtent, 44, 118
GetTitle, 49, 62, 83
GetUnderlined, 53
GetValue, 26, 93, 106, 109
GetVerticalSpacing, 96
GetVirtualSize, 21
GetWeight, 53
GetWidth, 13, 100

Iconize, 50, 62
Iconized, 50, 62
Initialize, 33, 70
Initialized, 11
Insert, 76
IsAbsolutePath, 121
IsButton, 90
IsRetained, 21
IsSelection, 35

KeyCode, 73
KeywordSearch, 71

Last, 76
LeftDown, 91
LeftIsDown, 91
LeftUp, 91
ListToArray, 108
LoadFile, 71, 110
LogicalToDeviceX, 44
LogicalToDeviceY, 44

MainLoop, 11
MakeConnection, 30
MakeKey, 69
Maximize, 62
MaxX, 44
MaxY, 44
Member, 76, 98, 108
MiddleDown, 91
MiddleIsDown, 91
MiddleUp, 91
MinX, 44
MinY, 44
Modified, 110
Moving, 91

Index

<<, 111
~wxApp, 11
~wxBitmap, 13
~wxBrush, 14
~wxButton, 16
~wxCanvas, 18
~wxCheckBox, 26
~wxChoice, 27
~wxColour, 33
~wxCursor, 39
~wxDC, 40
~wxDialogBox, 49
~wxEvent, 51
~wxFont, 52
~wxForm, 56
~wxFrame, 61
~wxHashTable, 68
~wxIcon, 67
~wxList, 75
~wxListBox, 78
~wxMenu, 81
~wxMenuBar, 84
~wxMessage, 86
~wxMetaFile, 87
~wxMetaFileDC, 88
~wxPanel, 95
~wxPen, 99
~wxRadioBox, 103
~wxSlider, 106
~wxStringList, 107
~wxText, 109
~wxTextWindow, 110
~wxTimer, 112
~wxWindow, 115

Add, 56, 107
AddBrush, 15
AddChild, 115
AddEnvList, 98
AddFont, 54
AddPath, 98
AddPen, 101
AddType, 114
Advise, 36
Append, 28, 75, 78, 82, 84
AppendSeparator, 82
AssociatePanel, 57

BeginFind, 68
Blit, 40
Button, 90
ButtonDown, 90

CaptureMouse, 115
Center, 115
Centre, 49, 61, 72, 115
Check, 82, 84
Checked, 34
Clear, 18, 28, 40, 68, 75, 78, 110
Close, 88
Command, 61, 72
ControlDown, 73, 90
copystring, 123
Create, 17, 18, 26, 28, 49, 61, 78, 86, 95, 103, 105, 106, 109, 110
CreateStatusLine, 61

Data, 93
Delete, 57, 68, 78, 107
DeleteContents, 75
DeleteNode, 75
DeleteObject, 75
Deselect, 79
DestroyChildren, 116
DestroyClippingRegion, 18, 40
DeviceToLogicalX, 41
DeviceToLogicalY, 41
DiscardEdits, 110
Disconnect, 36
DisplayBlock, 70
DisplayContents, 70
DisplaySection, 71
Dos2UnixFilename, 121
DragAcceptFiles, 116
Dragging, 90
DrawArc, 19, 41
DrawEllipse, 19, 41
DrawIcon, 41
DrawLine, 19, 41
DrawLines, 19, 42

Yields control to other applications (has no effect under XView or Motif).

See also **wxTimer** (section 2.57) .

::wxSubType

Bool **wxSubType**(*WXTYPE* **type1**, *WXTYPE* **type2**)

TRUE if *type1* is a subtype of, or the same type as, *type2*. This can be useful when determining whether an object is an instance of a class derived from some other class, and its usage can make for generic code.

See also **wxTypeTree** (section 2.58) and **wxObject::_type** (section 2.43) .

Example:

```
wxNode *node = GetChildren()->First();
while (node)
{
  // Find a child that's a subwindow, but not a dialog box.
  wxWindow *child = (wxWindow *)node->Data();
  if ((wxSubType(child->__type, wxTYPE_PANEL) &&
       !wxSubType(child->__type, wxTYPE_DIALOG_BOX)) ||
      wxSubType(child->__type, wxTYPE_TEXT_WINDOW) ||
      wxSubType(child->__type, wxTYPE_CANVAS))
  {
    child->SetFocus();
    return;
  }
  node = node->Next();
}
```

::wxWriteResource

Bool **wxWriteResource**(*char* ***section**, *char* ***entry**, *char* ***value**, *char* ***file = NULL**)

Bool **wxWriteResource**(*char* ***section**, *char* ***entry**, *float* **value**, *char* ***file = NULL**)

Bool **wxWriteResource**(*char* ***section**, *char* ***entry**, *long* **value**, *char* ***file = NULL**)

Bool **wxWriteResource**(*char* ***section**, *char* ***entry**, *int* **value**, *char* ***file = NULL**)

Writes a resource value into the resource database (for example, WIN.INI, or .Xdefaults). If *file* is NULL, WIN.INI or .Xdefaults is used, otherwise the specified file is used.

Under X, the resource databases are cached until the internal function **wxFlushResources** is called automatically on exit, when all updated resource databases are written to their files.

See also **wxGetResource** (section 3.6) .

::wxYield

void **wxYield**(*void*)

See also **wxWriteResource** (section 3.6) .

::wxGetUserId

Bool **wxGetUserId**(*char* ***buf**, *int* **bufSize**)

Copies the user's login identity (such as "jacs") into the buffer *buf*, of maximum size *bufSize*, returning TRUE if successful. Under Windows, this returns "user".

::wxGetUserName

Bool **wxGetUserName**(*char* ***buf**, *int* **bufSize**)

Copies the user's name (such as "Julian Smart") into the buffer *buf*, of maximum size *bufSize*, returning TRUE if successful. Under Windows, this returns "unknown".

::wxIPCInitialize

void **wxIPCInitialize**(*void*)

Initializes for interprocess communication operation. May be called multiple times without harm.

See also **wxServer** (section 2.52) , **wxClient** (section 2.12) , **wxConnection** (section 2.17) and the relevant section of the user manual.

::wxSleep

void **wxSleep**(*int* **secs**)

Under X, sleeps for the specified number of seconds using the technique specified in the XView manual, not UNIX **sleep**.

::wxStripMenuCodes

void **wxStripMenuCodes**(*char* ***in**, *char* ***out**)

Strips any menu codes from *in* and places the result in *out*. Menu codes include & (mark the next character with an underline as a keyboard shortkey in Windows and Motif) and \t (tab in Windows).

::wxStartTimer

void **wxStartTimer**(*void*)

Starts a stopwatch; use **::wxGetElapsedTime** (section 3.6) to get the elapsed time.

::wxFindWindowByLabel

wxWindow * **wxFindWindowByLabel**(*wxWindow* ***parent**, *char* ***label**)

Find a window by its label. Depending on the type of window, the label may be a window title or panel item label. If *parent* is NULL, the search will start from all top-level frames and dialog boces; if non-NULL, the search will be limited to the given window hierarchy. The search is recursive in both cases.

::wxGetHostName

Bool **wxGetHostName**(*char* ***buf**, *int* **bufSize**)

Copies the host name of the machine the program is running on into the buffer *buf*, of maximum size *bufSize*, returning TRUE if successful. Under UNIX, this will return a machine name. Under Windows, this returns "windows".

::wxGetElapsedTime

long **wxGetElapsedTime**(*void*)

Gets the time in milliseconds since the last **::wxGetElapsedTime** or **::wxStartTimer** (section 3.6) .

See also **wxTimer** (section 2.57) .

::wxGetFreeMemory

long **wxGetFreeMemory**(*void*)

Returns the amount of free memory in Kbytes under environments which support it, and -1 if not supported. Currently, returns a positive value under Windows, and -1 under UNIX.

::wxGetResource

Bool **wxGetResource**(*char* ***section**, *char* ***entry**, *char* ****value**, *char* ***file = NULL**)

Bool **wxGetResource**(*char* ***section**, *char* ***entry**, *float* ***value**, *char* ***file = NULL**)

Bool **wxGetResource**(*char* ***section**, *char* ***entry**, *long* ***value**, *char* ***file = NULL**)

Bool **wxGetResource**(*char* ***section**, *char* ***entry**, *int* ***value**, *char* ***file = NULL**)

Gets a resource value from the resource database (for example, WIN.INI, or .Xdefaults). If *file* is NULL, WIN.INI or .Xdefaults is used, otherwise the specified file is used.

Under X, if an application class (wxApp::wx_class) has been defined, it is appended to the string /usr/lib/X11/app-defaults/ to try to find an applications default file when merging all resource databases.

The syntax is identical to **printf**: pass a format string and a variable list of arguments.

::wxDisplaySize

void **wxDisplaySize**(*int* ***width**, *int* ***height**)

Gets the physical size of the display in pixels.

::wxError

void **wxError**(*char* ***msg**, *char* ***title = "wxWindows Internal Error"**)

Displays *msg* and continues. This writes to standard error under UNIX, and pops up a message box under Windows. Used for internal wxWindows errors. See also **wxFatalError** (section 3.6) .

::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).

::wxExit

void **wxExit**(*void*)

Exits application after calling **wxApp::OnExit** (section 2.3) . Should only be used in an emergency: normally the top-level frame should be deleted (after deleting all other frames) to terminate the application. See **wxFrame::OnClose** (section 2.25) and **wxApp** (section 2.3) .

::wxFatalError

void **wxFatalError**(*char* ***msg**, *char* ***title = "wxWindows Fatal Error"**)

Displays *msg* and exits. This writes to standard error under UNIX, and pops up a message box under Windows. Used for fatal internal wxWindows errors. See also **wxError** (section 3.6) .

::wxFindMenuItemId

int **wxFindMenuItemId**(*wxFrame* ***frame**, *char* ***menuString**, *char* ***itemString**)

Find a menu item identifier associated with the given frame's menu bar.

Define this and link before wxWindows library to allow registering events from 'outside' the main application.

::wxRemoveSecondaryEventHandler

Bool **wxRemoveSecondaryEventHandler**(*wxEventHandler* **handlerFunc**, *Bool* **pre**)

Remove a secondary event handler. Returns TRUE if it succeeds.

::wxSendEvent

Bool **wxSendEvent**(*wxEvent&* **event**, *Bool* **external**)

Send an event to the system; usually it will be external, but set external to FALSE if calling from within the main application in response to other events.

Returns TRUE if the event was processed.

3.6 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**.

::wxBell

void **wxBell**(*void*)

Ring the system bell.

::wxDebugMsg

void **wxDebugMsg**(*char* ***fmt**, ...)

Display a debugging message; under Windows, this will appear on the debugger command window, and under UNIX, it will be written to standard error.

::wxAddSecondaryEventHandler

Bool **wxAddSecondaryEventHandler**(*wxEventHandler* **handlerFunc**, *Bool* **pre**,
Bool **override**, *Bool* **append**)

Add a secondary event handler, pre = TRUE iff it should be called before the event is executed. override = TRUE iff the handler is allowed to override all subsequent events by returning TRUE. Returns TRUE if succeeds.

A secondary event handler is an application-defined handler that may intercept normal events, possibly overriding them. A primary event handler provides the normal behaviour for the event.

An event handler takes a pointer to a wxEvent and a boolean flag which is TRUE if the event was externally generated, and returns a boolean which is TRUE if that event was handled.

::wxNotifyEvent

Bool **wxNotifyEvent**(*wxEvent&* **event**, *Bool* **pre**)

Notify the system of the event you are about to execute/have just executed. If TRUE is returned and pre = TRUE, the calling code should not execute the event (since it has been intercepted by a handler and vetoed).

These events are always internal, because they're generated from within the main application code.

::wxRegisterEventClass

void **wxRegisterEventClass**(*WXTYPE* **eventClassId**,*WXTYPE* **superClassId**,
wxEventConstructor **constructor**, *char* ***description**)

Register a new event class (derived from wxEvent), giving the new event class type, its superclass, a function for creating a new event object of this class, and an optional description.

::wxRegisterEventName

void **wxRegisterEventName**(*WXTYPE* **eventTypeId**,*WXTYPE* **eventClassId**,
char ***eventName**)

Register the name of the event. This will allow a simple command language where giving the event type name and some arguments will cause a new event of class *eventClassId* to be created, with given event type, and some arguments, allows an event to be dynamically constructed and sent.

::wxRegisterExternalEventHandlers

void **wxRegisterExternalEventHandlers**(*void*)

::wxColourDisplay

Bool **wxColourDisplay**(*void*)

Returns TRUE if the display is colour, FALSE otherwise.

::wxDisplayDepth

int **wxDisplayDepth**(*void*)

Returns the depth of the display (a value of 1 denotes a monochrome display).

::wxSetCursor

void **wxSetCursor**(*wxCursor* ***cursor**)

Globally sets the cursor; only has an effect in Windows 3. See also **wxCursor** (section 2.18) , **wxWindow::SetCursor** (section 2.59) .

3.5 System event functions

The wxWindows system event implementation is incomplete and experimental, but is intended to be a platform-independent way of intercepting and sending events, including defining application-specific events and handlers.

Ultimately it is intended to be used as a way of testing wxWindows applications using scripts, although there are currently problems with this (especially with modal dialogs).

All this is documented more to provoke comments and suggestions, and jog my own memory, rather than to be used, since it has not been tested. However **wxSendEvent** will probably work if you instantiate the event structure properly for a command event type (see the code in `wb_panel.cc` for **wxPanel::OnDefaultAction** (section 2.44) which uses **wxSendEvent** to send a command to the default button).

::wxAddPrimaryEventHandler

Bool **wxAddPrimaryEventHandler**(*wxEventHandler* **handlerFunc**)

Add a primary event handler—the normal event handler for this event. For built-in events, these would include moving and resizing windows. User-defined primary events might include the code to select an image in a diagram (which could of course be achieved by a series of external events for mouse-clicking, but would be more difficult to specify and less robust).

Returns TRUE if it succeeds.

An event handler takes a pointer to a wxEvent and a boolean flag which is TRUE if the event was externally generated, and returns a boolean which is TRUE if that event was handled.

::wxGetSingleChoiceIndex

int **wxGetSingleChoiceIndex**(*char* ***message**, *char* ***caption**, *int* **n**, *char* ***choices**[],
wxFrame ***parent = NULL**, *int* **x = -1**, *int* **y = -1**,
Bool **centre = TRUE**)

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**, *Bool* **centre = TRUE**)

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**,
Bool **centre = TRUE**)

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.

If *centre* is TRUE, the message text is centred; if FALSE, the message is left-justified.

3.4 GDI functions

The following are relevant to the GDI (Graphics Device Interface).

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)
{
  ...
}
```

::wxGetTextFromUser

char * **wxGetTextFromUser**(*char* ***message**, *char* ***caption = "Input text"**,
char ***default_value = ""**, *wxFrame* ***parent = NULL**,
int **x = -1**, *int* **y = -1**, *Bool* **centre = TRUE**)

Pop up a dialog box with title set to *caption*, message *message*, and a *default_value*. The user may type in text and press OK to return this text, or press Cancel to return NULL.

If *centre* is TRUE, the message text (which may include new line characters) is centred; if FALSE, the message is left-justified.

::wxGetSingleChoice

char * **wxGetSingleChoice**(*char* ***message**, *char* ***caption**, *int* **n**, *char* ***choices**[],
wxFrame ***parent = NULL**, *int* **x = -1**, *int* **y = -1**,
Bool **centre = TRUE**)

Pops up a dialog box containing a message, OK/Cancel buttons and a single-selection listbox. The user may choose an item and press OK to return a string or Cancel to return NULL.

choices is an array of *n* strings for the listbox.

If *centre* is TRUE, the message text (which may include new line characters) is centred; if FALSE, the message is left-justified.

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** (section 3.1) .

::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.

3.2 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.

::StringMatch

Bool **StringMatch**(*char* ***s1**, *char* ***s2**)

Returns TRUE if the substring *s1* is found within *s2*, ignoring case.

3.3 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.

::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**)

::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.

::wxGetTempFileName

void **wxGetTempFileName**(*char **__prefix__, *char **__buf__)

Makes a temporary filename based on *prefix*, opens and closes the file, and places the name in *buf*.

Under Windows, the filename will include the drive and name of the directory allocated for temporary files (usually the contents of the TEMP variable). Under UNIX, the `/tmp` directory is used.

It is the application's responsibility to create and delete the file.

::wxIsWild

Bool **wxIsWild**(*char **__pattern__)

Returns TRUE if the pattern contains wildcards. See **wxMatchWild** (section 3.1) .

::wxMatchWild

Bool **wxMatchWild**(*char **__pattern__, *char **__text__, *Bool* __dot_special__)

Chapter 3

Miscellaneous functions

3.1 File functions

See also **wxPathList** (section 2.45) .

::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.

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::SetColourMap

void **SetColourMap**(*wxColourMap* ***colourMap**)

Assigns the given colourmap to the window.

wxWindow::SetCursor

wxCursor * **SetCursor**(*wxCursor* ***cursor**)

Sets the window's cursor, returning the previous cursor (if any). This function applies to all subwindows.

See also **::wxSetCursor** (section 3.4) , **wxCursor** (section 2.18) .

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.

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::ReleaseMouse

void **ReleaseMouse**(*void*)

Releases mouse input captured with **CaptureMouse**.

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.

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**(*wxKeyEvent&* **ch**)

Sent to the window when the user has pressed a key. See **wxKeyEvent** (section 2.31) for details.

For a list of the wxWindows virtual key codes, see the include file `base/common.h`. Note that the ASCII values do not have explicit key codes: they are passed as ASCII values.

See also **wxWindow::OnEvent** (section 2.59) for mouse event notification. **OnChar** is currently applicable to canvas subwindows only.

wxWindow::OnDropFiles

void **OnDropFiles**(*int* **n**, *char* ***files**[], *int* **x**, *int* **y**)

Under Windows, called when files have been dragged from the file manager to the window. *files* is an array of *n* strings, and *x* and *y* give the mouse position where the drop occurred. The window must have previously been enabled for dropping by calling **wxWindow::DragAcceptFiles** (section 2.59) .

wxWindow::OnEvent

void **OnEvent**(*wxMouseEvent&* **event**)

Sent to the window when the user has initiated an event with the mouse. Derive your own class to handle this message. So far, only relevant to the wxCanvas class. See **wxWindow::OnChar** (section 2.59) for character events, and also **wxMouseEvent** (section 2.40) for how to access event information.

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::GetGrandParent

wxWindow * **GetGrandParent**(*void*)

Returns the grandparent of a window, if any.

wxWindow::GetHandle

char * **GetHandle**(*void*)

Gets the platform-specific handle of the physical window.

wxWindow::GetPosition

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

This gets the position of the window in pixels, relative to the parent window or if no parent, relative to the whole display.

wxWindow::GetLabel

char * **GetLabel**(*void*)

Generic way of getting a label from any window, perhaps for identification purposes. Some windows do not have labels (such as subwindows), but frames, dialogs and panel items all have labels, where the interpretation of what constitutes a label differs from class to class. This can be useful for meta-programs (such as testing tools or special-needs access programs) which need to identify windows by name rather than visually.

wxWindow::GetParent

wxWindow * **GetParent**(*void*)

Returns the parent of a window, if any.

wxWindow::GetSize

void **GetSize**(*int* ***width**, *int* ***height**)

wxWindow::DestroyChildren

void **DestroyChildren**(*void*)

Destroys all children of a window. Called automatically by the destructor.

wxWindow::DragAcceptFiles

void **DragAcceptFiles**(*void*)

Under Windows, makes the window eligible for a **OnDropFiles** event.

wxWindow::Enable

void **Enable**(*Bool* **enable**)

Enable or disable the window, so input has no effect.

wxWindow::GetCharHeight

float **GetCharHeight**(*void*)

Get the character height for this window.

wxWindow::GetCharWidth

float **GetCharWidth**(*void*)

Get the average character width for this window.

wxWindow::GetChildren

*wxList ** **GetChildren**(*void*)

Returns a temporary pointer to a list of the window's children.

wxWindow::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.

2.59 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::CaptureMouse

void **CaptureMouse**(*void*)

Directs all mouse input to this window. Call **ReleaseMouse** to release the capture.

wxWindow::Center

void **Center**(*int* **direction**)

See **Centre** (section 2.59) .

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.

- wxTYPE_METAFILE
- wxTYPE_TIMER
- wxTYPE_COLOUR
- wxTYPE_COLOURMAP
- wxTYPE_CURSOR
- wxTYPE_DDE_CLIENT
- wxTYPE_DDE_SERVER
- wxTYPE_DDE_CONNECTION
- wxTYPE_HELP_INSTANCE
- wxTYPE_LIST
- wxTYPE_STRING_LIST
- wxTYPE_HASH_TABLE
- wxTYPE_NODE
- wxTYPE_APP

wxTypeTree::wxTypeTree

void **wxTypeTree**(*void*)

Constructor. Used by wxWindows only, since there only one instance of this class.

wxTypeTree::AddType

void **AddType**(*WXTYPE* **newType**, *WXTYPE* **parentType**, *char* ***name**)

Adds a type to the hierarchy. *newType* is the type being registered, *parentType* is the parent type, and *name* is an identifier (which can used in error messages). The top (root) type is wxTYPE_ANY.

Example:

```
wxAllTypes.AddType(wxTYPE_WINDOW, wxTYPE_ANY,     "window");
wxAllTypes.AddType(wxTYPE_PANEL,  wxTYPE_WINDOW,  "panel");
wxAllTypes.AddType(wxTYPE_CANVAS, wxTYPE_WINDOW,  "canvas");
```

wxTypeTree::GetName

char * **GetName**(*WXTYPE* **typ**)

Gets a temporary pointer to the name of the given type, or NULL if the type is not found.

- wxTYPE_WINDOW
- wxTYPE_DIALOG_BOX
- wxTYPE_ITEM
- wxTYPE_PANEL
- wxTYPE_CANVAS
- wxTYPE_TEXT_WINDOW
- wxTYPE_FRAME
- wxTYPE_BUTTON
- wxTYPE_TEXT
- wxTYPE_MESSAGE
- wxTYPE_CHOICE
- wxTYPE_LIST_BOX
- wxTYPE_SLIDER
- wxTYPE_CHECK_BOX
- wxTYPE_MENU
- wxTYPE_MENU_BAR
- wxTYPE_MULTI_TEXT
- wxTYPE_RADIO_BOX
- wxTYPE_EVENT
- wxTYPE_DC
- wxTYPE_DC_CANVAS
- wxTYPE_DC_POSTSCRIPT
- wxTYPE_DC_PRINTER
- wxTYPE_DC_METAFILE
- wxTYPE_DC_MEMORY
- wxTYPE_MOUSE_EVENT
- wxTYPE_KEY_EVENT
- wxTYPE_COMMAND_EVENT
- wxTYPE_PEN
- wxTYPE_BRUSH
- wxTYPE_FONT
- wxTYPE_ICON
- wxTYPE_BITMAP

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.

2.58 wxTypeTree: wxList

wxTypeTree implements an explicit type hierarchy which can be useful for querying C++ types at run-time, usually by calling **wxSubType** (section 3.6) using the **wxObject::_type** (section 2.43) member.

A type is added to the global variable **wxAllTypes**; wxWindows adds its own standard types on initialization, in **wxInitStandardTypes**, but the application can add its own.

The standard wxWindows types, grouped by functionality, are:

- wxTYPE_ANY
- wxTYPE_OBJECT (an alias for wxTYPE_ANY)

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 << (section 2.56) 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";
```

2.57 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.

See also **::wxStartTimer** (section 3.6) and **::wxGetElapsedTime** (section 3.6) for stopwatch functions.

Constructor.

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

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

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

wxTextWindow::~wxTextWindow

void **~wxTextWindow**(*void*)

Destructor. Deletes any stored text before deleting the physical window.

wxTextWindow::Clear

void **Clear**(*void*)

Clears the window and deletes the stored text.

wxTextWindow::Create

Bool **Create**(*wxFrame* ***parent**, *int* **x = -1**, *int* **y = -1**,
int **width = -1**, *int* **height = -1**, *int* **style = 0**, *char* ***name = "textWindow"**)

Creates the text item for two-step construction. Derived classes should call or replace this function. See **wxTextWindow::wxTextWindow** (section 2.56) for further details.

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*)

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

wxText::~wxText

void **~wxText**(*void*)

Destructor, destroying the text item.

wxText::Create

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

Creates the text item for two-step construction. Derived classes should call or replace this function. See **wxText::wxText** (section 2.55) for further details.

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.

2.56 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**(*void*)

Constructor, for deriving classes.

void **wxTextWindow**(*wxFrame* ***parent**, *int* **x = -1**, *int* **y = -1**,
int **width = -1**, *int* **height = -1**, *int* **style = 0**, *char* ***name = "textWindow"**)

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.

2.55 wxText: wxWindow

A text item is an area of editable text, with an optional label displayed in front of it.

wxText::wxText

void **wxText**(*void*)

Constructor, for deriving classes.

void **wxText**(*wxPanel* ***parent**, *wxFunction* **func**, *char* ***label**,
char ***value = ""**, *int* **x = -1**, *int* **y = -1**, *int* **width = -1**, *int* **height = -1**,
int **style = 0**, *char* ***name = "text"**)

Constructor, creating and showing a text item with the given string value.

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

If *label* is non-NULL, it will be used to label the text item.

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

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

Gets the current slider value.

wxSlider::SetValue

void **SetValue**(*int* **value**)

Sets the value (and displayed position) of the slider).

2.54 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.

2.53 wxSlider: wxItem

A slider is, as its name suggests, an item with a handle which can be pulled back and forth to change a value. It is currently horizontal only. In Windows 3, a scrollbar is used to simulate the slider.

wxSlider::wxSlider

void **wxSlider**(*wxPanel* ***parent**, *wxFunction* **func**, *char* ***label**,
int **value**, *int* **min_value**, *int* **max_value**, *int* **width**,
int **x = -1**, *int* **y = -1**, *int* **style = 0**, *char* ***name = "slider"**)

Constructor, creating and showing a horizontal slider.

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

If *label* is non-NULL, it will be used to label the slider.

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

The *width* is in pixels, and the scroll increment will be adjusted to a suitable value given the minimum and maximum integer values.

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

wxSlider::~wxSlider

void **~wxSlider**(*void*)

Destructor, destroying the slider.

wxSlider::Create

void **Create**(*wxPanel* ***parent**, *wxFunction* **func**, *char* ***label**,
int **value**, *int* **min_value**, *int* **max_value**, *int* **width**,
int **x = -1**, *int* **y = -1**, *int* **style = 0**, *char* ***name = "slider"**)

Used for two-step slider construction. See **wxSlider::wxSlider** (section 2.53) for further details.

wxSlider::GetValue

int **GetValue**(*void*)

Sets the choice by passing the desired string position.

wxRadioBox::SetStringSelection

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

Sets the choice by passing the desired string.

wxRadioBox::GetString

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

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

2.52 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 the chapter on IPC in the user manual.

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.

int **x = -1**, *int* **y = -1**, *int* **width = -1**, *int* **height = -1**,
int **n**, *wxBitmap* ***choices[]**, *int* **majorDim = 0**, *int* **style = 0**, *char* ***name = "radioBox"**)

Creates the choice for two-step construction. Derived classes should call or replace this function. See **wxRadioBox::wxRadioBox** (section 2.51) for further details.

wxRadioBox::Enable

void **Enable**(*Bool* **enable**)

Enables or disables the entire radiobox.

void **Enable**(*int* **n**, *Bool* **enable**)

Enables or disables an individual button in the radiobox (does nothing in XView).

wxRadioBox::FindString

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

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

wxRadioBox::GetSelection

int **GetSelection**(*void*)

Gets the id (position) of the selected string.

wxRadioBox::GetStringSelection

char * **GetStringSelection**(*void*)

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

wxRadioBox::Number

int **Number**(*void*)

Returns the number of choices in the radiobox.

wxRadioBox::SetSelection

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

wxRadioBox::wxRadioBox

void **wxRadioBox**(*void*)

Constructor, for use by derived classes.

void **wxRadioBox**(*wxPanel* ***parent**, *wxFunction* **func**, *char* ***label**,
int **x = -1**, *int* **y = -1**, *int* **width = -1**, *int* **height = -1**,
int **n**, *char* ***choices**[], *int* **majorDim = 0**, *int* **style = 0**, *char* ***name = "radioBox"**)

void **wxRadioBox**(*wxPanel* ***parent**, *wxFunction* **func**, *char* ***label**,
int **x = -1**, *int* **y = -1**, *int* **width = -1**, *int* **height = -1**,
int **n**, *wxBitmap* ***choices**[], *int* **majorDim = 0**, *int* **style = 0**, *char* ***name = "radioBox"**)

Constructor, creating and showing a radiobox.

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

If *label* is non-NULL, it will be used to label the radiobox.

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

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

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

majorDim specifies the number of rows for a two-dimensional radiobox.

style specifies a bitwise-or list of styles (none so far).

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

wxRadioBox::~wxRadioBox

void **~wxRadioBox**(*void*)

Destructor, destroying the radiobox item.

wxRadioBox::Create

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

Bool **Create**(*wxPanel* ***parent**, *wxFunction* **func**, *char* ***label**,

wxPostScriptDC::wxPostScriptDC

void **wxPostScriptDC**(*char* ***output**, *Bool* **interactive = TRUE**)

Constructor. *output* is an optional file for printing to, and if *interactive* is TRUE a dialog box will be displayed for adjusting various parameters.

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

2.50 wxPrinterDC: wxDC

A printer device context is specific to Windows, and allows access to any printer with a Windows driver. See **wxDC** (section 2.19) for further information on device contexts.

wxPrinterDC::wxPrinterDC

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

Constructor. With three NULLs, the default printer dialog is displayed. *device* indicates the type of printer and *output* is an optional file for printing to. The *driver* parameter is currently unused. Use the *Ok* member to test whether the constructor was successful in creating a useable device context.

2.51 wxRadioBox: wxItem

A radio box item is used to select one of number of mutually exclusive choices. It is displayed as a vertical column or horizontal row of labelled buttons.

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.

2.48 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.

2.49 wxPostScriptDC: wxDC

This defines the wxWindows Encapsulated PostScript device context, which can write PostScript files on any platform. See **wxDC** (section 2.19) for descriptions of the member functions.

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.

2.47 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**.

2.46 wxPen: wxObject

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

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

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

An application may wish to dynamically create pens with different characteristics, and there is the consequent danger that a large number of duplicate pens will be created. Therefore an application may wish to get a pointer to a pen by using the global list of pens **wxThePenList**, and calling the member function **FindOrCreatePen**. See the entry for **wxPenList** (section 2.47) .

wxPen::wxPen

void **wxPen**(*void*)

void **wxPen**(*wxColour &***colour**, *int* **width**, *int* **style**)

void **wxPen**(*char **colour_name**, *int* **width**, *int* **style**)

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

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*)

Tabs by the given number of pixels.

2.45 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).

```
    wxCommandEvent event(wxEVENT_TYPE_BUTTON_COMMAND);
    but->Command(event);
  }
}
```

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::SetButtonFont

void **SetButtonFont**(*wxFont* ***font**)

Specify the font for drawing panel item values (not XView).

wxPanel::SetLabelFont

void **SetLabelFont**(*wxFont* ***font**)

Specify the font for drawing panel item labels (not XView).

wxPanel::SetVerticalSpacing

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

Sets the vertical spacing for placing items on a panel.

wxPanel::Tab

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

wxPanel::GetDefaultItem

wxButton * **GetDefaultItem**(*void*)

Retrieves the default button, previously set with **wxButton::SetDefault** (section 2.7) .

wxPanel::GetHorizontalSpacing

int **GetHorizontalSpacing**(*void*)

Gets the horizontal spacing for placing items on a panel.

wxPanel::GetLabelFont

wxFont * **GetLabelFont**(*void*)

Get the current font for drawing panel item labels.

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::OnDefaultAction

void **OnDefaultAction**(*wxItem* ***item**)

Called when the user initiates the default action for a panel or dialog box, for example by double clicking on a listbox. *item* is the panel item which caused the default action.

The default behaviour for this member is to retrieve the default button for the panel, and send it a command event, as if the user had clicked on the button. This gives default listbox double-click behaviour under Motif and Windows 3. The default code is as follows:

```
void wxbPanel::OnDefaultAction(wxItem *initiatingItem)
{
  wxButton *but = GetDefaultItem();
  if (but)
  {
```

- **wxASBOLUTE_POSITIONING**: a hint to the windowing system not to try automatic layout (Motif only).
- **wxBORDER**: draw a thin border around the panel.
- **wxVSCROLL**: give the dialog box a vertical scrollbar (XView only).

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

The parent window may be a panel in Motif and Windows, but not XView.

wxPanel::~wxPanel

void **~wxPanel**(*void*)

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

wxPanel::Create

void **Create**(*wxFrame* ***parent**, *int* **x = -1**, *int* **y = -1**, *int* **width = -1**, *int* **height = -1**, *int* **style = 0**, *char* ***name = "panel"**)

Used in two-step panel construction. See **wxPanel::wxPanel** (section 2.44) for further details.

wxPanel::Fit

void **Fit**(*void*)

Resize the panel to just fit around the panel items. Also works for dialog boxes.

wxPanel::GetButtonFont

wxFont * **GetButtonFont**(*void*)

Get the current font for drawing panel item values.

wxPanel::GetCursor

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

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

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

2.43 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.

wxObject::_type

WXTYPE *_type*

Data member used for storing dynamic type information. Most wxWindows classes set this member to an appropriate type, which may be overridden in derived classes. Optionally set this in your constructor. The type may be checked using **::wxSubType** (section 3.6) .

Note the double underscore prefixing this name, in order to minimize clashes with application code. There is no accessor function for this member, and its scope is public.

See also **wxTypeTree** (section 2.58) .

2.44 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**(*void*)

Constructor, for deriving classes.

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

void **wxPanel**(*wxPanel* ***parent**, *int* **x = -1**, *int* **y = -1**, *int* **width = -1**, *int* **height = -1**, *int* **style = 0**, *char* ***name = "panel"**)

Constructor.

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

The style parameter may be a combination of the following, using the bitwise 'or' operator.

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

2.41 wxMultiText: wxText

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

wxMultiText::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).

2.42 wxNode: wxObject

A node structure used in linked lists (see **wxList** (section 2.32)).

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*)

wxMouseEvent::Position

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

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

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

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

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

wxMouseEvent::RightDown

Bool **RightDown**(*void*)

Returns TRUE if the right mouse button changed to down.

wxMouseEvent::RightIsDown

Bool **RightIsDown**(*void*)

Returns TRUE if the right mouse button is currently down, independent of the current event type.

wxMouseEvent::RightUp

Bool **RightUp**(*void*)

Returns TRUE if the right mouse button changed to up.

wxMouseEvent::ShiftDown

Bool **ShiftDown**(*void*)

wxMouseEvent::LeftDown

Bool **LeftDown**(*void*)

Returns TRUE if the left mouse button changed to down.

wxMouseEvent::LeftIsDown

Bool **LeftIsDown**(*void*)

Returns TRUE if the left mouse button is currently down, independent of the current event type.

wxMouseEvent::LeftUp

Bool **LeftUp**(*void*)

Returns TRUE if the left mouse button changed to up.

wxMouseEvent::MiddleDown

Bool **MiddleDown**(*void*)

Returns TRUE if the middle mouse button changed to down.

wxMouseEvent::MiddleIsDown

Bool **MiddleIsDown**(*void*)

Returns TRUE if the middle mouse button is currently down, independent of the current event type.

wxMouseEvent::MiddleUp

Bool **MiddleUp**(*void*)

Returns TRUE if the middle mouse button changed to up.

wxMouseEvent::Moving

Bool **Moving**(*void*)

Returns TRUE if this was a motion event (no buttons depressed).

- **wxEVENT_TYPE_MIDDLE_DOWN**
- **wxEVENT_TYPE_MIDDLE_UP**
- **wxEVENT_TYPE_RIGHT_DOWN**
- **wxEVENT_TYPE_RIGHT_UP**
- **wxEVENT_TYPE_MOTION**

wxMouseEvent::Button

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

Returns TRUE if the identified mouse button is changing state. Valid values of *button* are 1, 2 or 3 for left, middle and right buttons respectively.

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

wxMouseEvent::ButtonDown

Bool **ButtonDown**(*int* **but = -1**)

If the argument is omitted, this returns TRUE if the event was a mouse button down event. Otherwise the argument specifies which button-down event was generated (1, 2 or 3 for left, middle and right buttons respectively).

wxMouseEvent::ControlDown

Bool **ControlDown**(*void*)

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

wxMouseEvent::Dragging

Bool **Dragging**(*void*)

Returns TRUE if this was a dragging event (motion while a button is depressed).

wxMouseEvent::IsButton

Bool **IsButton**(*void*)

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

wxMouseEvent::middleDown

Bool *middleDown*

TRUE if the middle mouse button is currently pressed down.

wxMouseEvent::rightDown

Bool *rightDown*

TRUE if the right mouse button is currently pressed down.

wxMouseEvent::leftDown

Bool *leftDown*

TRUE if the left mouse button is currently pressed down.

wxMouseEvent::shiftDown

Bool *shiftDown*

TRUE if shift is pressed down.

wxMouseEvent::x

float *x*

X-coordinate of the event.

wxMouseEvent::y

float *y*

Y-coordinate of the event.

wxMouseEvent::wxMouseEvent

void **wxMouseEvent**(*WXTYPE* **mouseEventType**)

Constructor. Valid event types are:

- **wxEVENT_TYPE_LEFT_DOWN**
- **wxEVENT_TYPE_LEFT_UP**

purpose for this at present is to allow the metafile to be copied to the clipboard (see **wxMetaFile** (section 2.38)).

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).

2.40 wxMouseEvent: wxEvent

This event class contains information about mouse events, particularly events received by canvases. See **wxCanvas::OnEvent** (section 2.8) .

wxMouseEvent::controlDown

Bool *controlDown*

TRUE if control key is pressed down.

wxMouseEvent::leftDown

Bool *leftDown*

TRUE if the left mouse button is currently pressed down.

2.38 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.

Below is a example of metafle, metafile device context and clipboard use from the `hello.cc` example. Note the way the metafile dimensions are passed to the clipboard, making use of the device context's ability to keep track of the maximum extent of drawing commands.

```
wxMetaFileDC dc;
if (dc.Ok())
{
  Draw(dc, FALSE);
  wxMetaFile *mf = dc.Close();
  if (mf)
  {
    Bool success = mf->SetClipboard((int)(dc.MaxX() + 10), (int)(dc.MaxY() + 10));
    delete mf;
  }
}
```

2.39 wxMetaFileDC: wxDC

This is a type of device context that allows a metafile object to be created (Windows only), and has most of the characteristics of a normal **wxDC**. The **wxMetaFileDC::Close** (section 2.39) member must be called after drawing into the device context, to return a metafile. The only

Sets the label of the given menu item. Use only after the menubar has been associated with a frame with **wxFrame::SetMenuBar**.

wxMenuBar::SetLabelTop

void **SetLabelTop**(*int* **pos**, *char* ***label**)

Sets the label of the given top-level menu. *pos* is the position of a menu on the menu bar. Use only after the menubar has been associated with a frame with **wxFrame::SetMenuBar**.

2.37 wxMessage: wxItem

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

wxMessage::wxMessage

void **wxMessage**(*void*)

Constructor, used for deriving classes.

void **wxMessage**(*wxPanel* ***panel**, *char* ***message**, *int* **x = -1**, *int* **y = -1**,
int **style = 0**, *char* ***name = "message"**)

Creates and displays a simple text message. *message* is the initial value of the message.

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

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

wxMessage::~wxMessage

void **~wxMessage**(*void*)

Destroys the message.

wxMessage::Create

Bool **Create**(*wxPanel* ***panel**, *char* ***message**, *int* **x = -1**, *int* **y = -1**,
int **style = 0**, *char* ***name = "message"**)

Creates the listbox for two-step construction. Derived classes should call or replace this function. See **wxMessage::wxMessage** (section 2.37) for further details.

wxMenuBar::FindMenuItem

int **FindMenuItem**(*char* ***menuString**, *char* ***itemString**)

Finds the menu item id for a menu name/menu item string pair, or -1 if none found. Any special menu codes are stripped out of source and target strings before matching.

wxMenuBar::FindItemById

wxMenuItem * **FindItemById**(*int* **itemId**)

Finds the menu item object associated with the given menu item identifier, returning NULL if not found.

wxMenuBar::GetHelpString

char * **GetHelpString**(*int* **itemId**)

Gets the help string associated with the menu item identifer (or NULL if there is no help string or the item was not found).

wxMenuBar::GetLabel

char * **GetLabel**(*int* **itemId**)

Returns a pointer to a new piece of memory containing the label of the given menu item. Use only after the menubar has been associated with a frame with **wxFrame::SetMenuBar**.

wxMenuBar::SetLabelTop

char * **GetLabelTop**(*int* **pos**)

Gets a pointer to a new piece of memory containing the label of the given top-level menu. *pos* is the position of a menu on the menu bar. Use only after the menubar has been associated with a frame with **wxFrame::SetMenuBar**.

wxMenuBar::SetHelpString

void **SetHelpString**(*int* **itemId**, *char* ***helpString**)

Sets the help string associated with the menu item identifer.

wxMenuBar::SetLabel

void **SetLabel**(*int* **itemId**, *char* ***label**)

wxMenuBar::wxMenuBar

void **wxMenuBar**(*void*)

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

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

wxMenuBar::~wxMenuBar

void **~wxMenuBar**(*void*)

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

wxMenuBar::Append

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

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

wxMenuBar::Check

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

If *flag* is TRUE, checks the given menu item, else unchecks it. 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.

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::EnableTop

void **EnableTop**(*int* **pos**, *Bool* **flag**)

If *flag* is TRUE, enables the menu at the given position, else disables it (greys it). Only use this when the menu bar has been associated with a frame.

wxMenu::GetHelpString

*char ** **GetHelpString**(*int* **itemId**)

Gets the help string associated with the menu item identifer (or NULL if there is no help string or the item was not found).

wxMenu::GetLabel

*char ** **GetLabel**(*int* **id**)

Gets a pointer to a new piece of memory containing the label of the given menu item (using the identifier used to append the item to the menu).

wxMenu::GetTitle

*char ** **GetTitle**(*void*)

Gets a temporary pointer the title of the menu.

wxMenu::SetHelpString

void **SetHelpString**(*int* **itemId**, *char **__helpString__)

Sets the help string associated with the menu item identifer.

wxMenu::SetLabel

void **SetLabel**(*int* **id**, *char **__label__)

Sets the label of the given menu item (using the identifier used to append the item to the menu).

wxMenu::SetTitle

void **SetTitle**(*char **__title__)

Sets the title of the menu.

2.36 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.

wxMenu::Append

void **Append**(*int* **id**, *char ** **item**, *char **__helpString = NULL__)

void **Append**(*int* **id**, *char ** **item**, *wxMenu **__submenu__, *char **__helpString = NULL__)

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

Each form can take an optional help string, which can be accessed using **wxMenu::GetHelpString**. The default **wxFrame::OnMenuSelect** member uses this help string to give help on the menu item currently under the cursor.

wxMenu::AppendSeparator

void **AppendSeparator**(*void*)

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

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.

wxMenu::Enable

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

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

wxMenu::FindItem

int **FindItem**(*char **__itemString__)

Finds the menu item id for a menu item string, or -1 if none found. Any special menu codes are stripped out of source and target strings before matching.

wxMenu::FindItemById

*wxMenuItem ** **FindItemById**(*int* **itemId**)

Finds the menu item object associated with the given menu item identifier, returning NULL if not found.

wxMemoryDC::wxMemoryDC

void **wxMemoryDC**(*void*)

Constructs a new memory device context.

void **wxMemoryDC**(*wxCanvasDC* ***oldDC**)

Constructs a new memory device context with similar attributes to the given canvas device context.

Use the *Ok* member to test whether the constructor was successful in creating a useable device context. Don't forget to select a bitmap into the DC before drawing on it.

wxMemoryDC::SelectObject

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

Selects the given bitmap into the device context, to use as the memory bitmap. Selecting the bitmap into a memory DC allows you to draw into the DC (and therefore the bitmap) and also to use **Blit** to copy the bitmap to a canvas. For this purpose, you may find **wxDC::DrawIcon** (section 2.19) easier to use instead.

2.35 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.

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**, *Bool***select = TRUE**)

Selects or deselects the given item.

wxListBox::SetStringSelection

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

Sets the choice by passing the desired string.

wxListBox::GetString

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

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

2.34 wxMemoryDC: wxObject

A memory device context provides a means to draw graphics onto a bitmap.

A bitmap must be selected into the new memory DC before it may be used for anything. Typical usage is as follows:

```
// Create a memory DC
wxMemoryDC temp_dc;
temp_dc.SelectObject(test_bitmap);

// We can now draw into the memory DC...
// Copy from this DC to another DC.
old_dc.Blit(250, 50, BITMAP_WIDTH, BITMAP_HEIGHT, temp_dc, 0, 0);
```

Note that the memory DC *must* be deleted before a bitmap can be reselected into another memory DC.

See also **wxBitmap** (section 2.4) , **wxDC** (section 2.19) , **wxCanvasDC** (section 2.9) .

wxListBox::Deselect

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

Deselects the given item in the list box.

wxListBox::FindString

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

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

wxListBox::GetClientData

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

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

wxListBox::GetSelection

int **GetSelection**(*void*)

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

wxListBox::GetSelections

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

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

wxListBox::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::Selected

Bool **Selected**(*int* **n**)

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

wxALWAYS_SB Created scrollbars immediately.

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

wxListBox::~wxListBox

void **~wxListBox**(*void*)

Destructor, destroying the list box.

wxListBox::Append

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

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

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

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

wxListBox::Clear

void **Clear**(*void*)

Clears all strings from the list box.

wxListBox::Create

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

Creates the listbox for two-step construction. Derived classes should call or replace this function. See **wxListBox::wxListBox** (section 2.33) for further details.

wxListBox::Delete

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

Delete the nth element in the list box.

Returns the number of elements in the list.

2.33 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).

See also **wxChoice** (section 2.11) .

wxListBox::wxListBox

void **wxListBox**(*void*)

Constructor, for deriving classes.

void **wxListBox**(*wxPanel* ***parent**, *wxFunction* **func**, *char* ***label**,
Bool **multiple_selection = wxSINGLE**, *int* **x = -1**, *int* **y = -1**,
int **width = -1**, *int* **height = -1**, *int* **n**, *char* ***choices**[],
int **style = 0**, *char* ***name = "listBox"**)

Constructor, creating and showing a list box.

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

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

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

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

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

multiple_selection is one of the following:

wxSINGLE Single-selection list.

wxMULTIPLE Multiple-selection list.

wxEXTENDED Extended-selection list (Motif only).

style is a bit-wise or list of flags:

wxNEEDED_SB Create scrollbars if needed.

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*)

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.

```
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** (section 2.42) 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.

wxKeyEvent::shiftDown

Bool *shiftDown*

TRUE if shift is pressed down.

wxKeyEvent::wxKeyEvent

void **wxKeyEvent**(*WXTYPE* **keyEventType**)

Constructor. Currently, the only valid event type is wxEVENT_TYPE_CHAR.

wxKeyEvent::ControlDown

Bool **ControlDown**(*void*)

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

wxKeyEvent::KeyCode

long **KeyCode**(*void*)

Returns the virtual key code. ASCII events return normal ASCII values, while non-ASCII events return values such as **WXK_LEFT** for the left cursor key. See `common.h` for a full list of the virtual key codes.

wxKeyEvent::ShiftDown

Bool **ShiftDown**(*void*)

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

2.32 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** (section 2.28) 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;
```

wxItem::Centre

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

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

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

wxItem::Command

void **Command**(*wxCommandEvent* **event**)

Simulate the effect of the user issuing a command to the item. See **wxCommandEvent** (section 2.16) .

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.

2.31 wxKeyEvent: wxEvent

This event class contains information about key events. See **wxCanvas::OnChar** (section 2.8) .

wxKeyEvent::controlDown

Bool *controlDown*

TRUE if control is pressed down.

wxKeyEvent::keyCode

long *keyCode*

Virtual keycode.

wxHelpInstance::DisplaySection

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

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

wxHelpInstance::KeywordSearch

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

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

wxHelpInstance::LoadFile

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

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

wxHelpInstance::OnQuit

Bool **OnQuit**(*void*)

Overridable member called when this application's wxHelp is quit (no effect if Windows Help is being used instead).

wxHelpInstance::Quit

Bool **Quit**(*void*)

If wxHelp is running, quits wxHelp by disconnecting (no effect for Windows Help).

2.30 wxItem: wxWindow

This is the base class for any widget or control which can be placed on a panel or dialog box.

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

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

If you give TRUE to the constructor, you can use the native help system where appropriate (currently under Windows only). Omit the file extension to allow wxWindows to choose the appropriate file for the platform.

wxHelpInstance::wxHelpInstance

void **wxHelpInstance**(*Bool* **native**)

Constructs a help instance object, but does not invoke wxHelp. If *native* is TRUE, tries to use the native help system where possible (Windows Help under Windows 3.1, wxHelp on other platforms).

wxHelpInstance::~wxHelpInstance

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

wxHelpInstance::Initialize

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

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

You may omit the file extension, and in fact this is recommended if you wish to support .xlp files under X and .hlp under Windows.

wxHelpInstance::DisplayBlock

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

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

wxHelpInstance::DisplayContents

Bool **DisplayContents**(*void*)

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

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 description for **wxNode** (section 2.42) . 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.

2.29 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.

data object is appended to the list, storing the key in case other data objects need to be stored in the list also (when a 'collision' occurs).

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

See also **wxList** (section 2.32) .

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).

```
#ifdef wx_x
#include "aiai.xbm"
#endif
#ifdef wx_msw
  wxIcon *icon = new wxIcon("aiai");
#endif
#ifdef wx_x
  wxIcon *icon = new wxIcon(aiai_bits, aiai_width, aiai_height);
#endif
```

See also **wxDC::DrawIcon** (section 2.19) , **wxBitmap** (section 2.4) .

wxIcon::wxIcon

void **wxIcon**(*char* **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.

2.28 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

wxFrame::SetTitle

void **SetTitle**(*char ** **title**)

Sets the frame title. See **wxFrame::GetTitle** (section 2.25) .

wxFrame::StatusLineExists

Bool **StatusLineExists**(*void*)

Returns TRUE if the status line has previously been created. See **wxFrame::CreateStatusLine** (section 2.25) , **wxFrame::SetStatusText** (section 2.25) .

2.26 wxFunction

.

The type of a panel item callback function.

wxFunction

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

The type of a panel item callback function. The first parameter is a reference to the object, and the second is a command event structure. See **wxItem** (section 2.30) , **wxCommandEvent** (section 2.16) .

2.27 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!

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_MDIPARENTFRAME ICON icon1.ico
wxSTD_MDICHILDFRAME  ICON icon2.ico
```

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

To assign an icon to all SDI frames, include a line similar to the following:

```
wxSTD_FRAME      ICON icon1.ico
```

See also **wxIcon** (section 2.27) .

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).

See also **wxMenuBar** (section 2.36) , **wxMenu** (section 2.35) .

wxFrame::SetStatusText

void **SetStatusText**(*char ** **text**, *int* **number = 0**)

Sets the status line text and redraws the status line. Use an empty (not NULL) string to clear the status line. Optionally use *number* to specify a field in the status line, starting from zero and consistent with the number of fields specified in **wxFrame::CreateStatusLine** (section 2.25) .

```
        frame->text_window->LoadFile(s);
      break;
    }
    case HELLO_QUIT:
    {
      OnClose();
      delete this;
      break;
    }
    case HELLO_PRINT_EPS:
    {
      wxPostScriptDC dc(NULL, TRUE);
      if (dc.Ok())
      {
        dc.StartDoc("Hello printout");
        dc.StartPage();
        Draw(dc, TRUE);
        dc.EndPage();
        dc.EndDoc();
      }
      break;
    }
    case HELLO_ABOUT:
    {
      (void)wxMessageBox("wxWindows GUI library demo", "About wxHello");
      break;
    }
  }
}
```

wxFrame::OnMenuSelect

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

Sent to the window when an item on the window's menu has been selected (i.e. the cursor is on the item, but the left button has not been released). Derive your own frame class to handle this message.

The default **OnMenuSelect** member puts the menu item help string on the status line, if a status line has been created.

This function is called under Windows 3 and Motif, but not XView.

wxFrame::OnSize

void **OnSize**(*int* **w**, *int* **h**)

Called when the user or application resizes the frame. The parameters give the total size of the frame. The default **OnSize** member looks for a single subwindow, and if one is found, resizes it to fit inside the frame. Override this member if more complex behaviour is required (for example, if there are several subwindows).

Sent to the frame when the user has tried to close the window using the window manager (X) or system menu (Windows). If TRUE is returned by **OnClose**, the frame will be deleted by the system, otherwise the attempt will be ignored. Derive your own class to handle this message; the default handler returns FALSE.

This member is a good place to delete other frames which are conceptually or actually subframes of this frame, since if all frames are not deleted, the application cannot exit. This is the top-level **OnClose** handler for the 'hello' demo:

```
Bool MyFrame::OnClose(void)
{
  if (subframe)
    delete subframe;

  return TRUE;
}
```

When quitting the application from inside the application, for example from a menu item, call the frame's **OnClose** member before deleting the frame. wxWindows then causes the application to exit without further ado. For example:

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

wxFrame::OnMenuCommand

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

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

```
// Intercept menu commands
void MyFrame::OnMenuCommand(int id)
{
  switch (id)
  {
    case HELLO_LOAD_FILE:
    {
      char *s = wxFileSelector("Load text file", NULL, NULL, NULL, "*.txt");
      if (s)
```

Creates a status line at the bottom of the frame. The width of the status line is the whole width of the frame (adjusted automatically when resizing), and the height and text size are chosen by the host system.

The default is to create one field the width of the frame; specify a value between 1 and 5 to create a multi-field status line.

See alo **wxFrame::SetStatusText** (section 2.25) .

wxFrame::Fit

void **Fit**(*void*)

Reize the frame to just fit around the subwindows. If a frame has only one subwindow, that subwindow will be resized by the default **wxFrame::OnSize** (section 2.25) member to fit inside the frame.

wxFrame::GetTitle

char * **GetTitle**(*void*)

Gets the frame title. See **wxFrame::SetTitle** (section 2.25) .

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::Maximize

void **Maximize**(*Bool* **maximize**)

Maximizes the frame if *maximize* is TRUE, otherwise restores it (Windows 3.1 only).

wxFrame::OnClose

Bool **OnClose**(*void*)

- **wxMAXIMIZE_BOX**: display a maximize box on the frame (Windows only).
- **wxSDI**: specifies a normal SDI (single document interface) frame.
- **wxSTAY_ON_TOP**: stay on top of other windows (Windows only).
- **wxSYSTEM_MENU**: display a system menu (manadatory under XView and Motif).
- **wxTHICK_FRAME**: display a thick frame around the window (manadatory under XView and Motif).

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

wxFrame::~wxFrame

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

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

wxFrame::Centre

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

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

wxFrame::Command

void **Command**(*int* **id**)

Simulate a menu command. *id* is the identifier for a menu item.

wxFrame::Create

void **Create**(*wxFrame* ***parent**, *char* ***title**, *int* **x = -1**, *int* **y = -1**,
int **width = -1**, *int* **height = -1**,
int **style = wxSDI || wxDEFAULT_FRAME**, *char* ***name = "frame"**)

Used in two-step frame construction. See **wxFrame::wxFrame** (section 2.25) for further details.

wxFrame::CreateStatusLine

void **CreateStatusLine**(*int* **number = 1**)

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.

2.25 wxFrame: wxWindow

A frame is a window which contains subwindows of various kinds. It has a title bar and, optionally, a menu bar, and a status line. Depending on the platform, the frame has further menus or buttons relating to window movement, sizing, closing, etc. Most of these events are handled by the host system without need for special handling by the application. However, the application should normally define an **wxFrame::OnClose** (section 2.25) 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 the user manual.

A frame may contain the subwindows **wxCanvas** (section 2.8) , **wxPanel** (section 2.44) and **wxTextWindow** (section 2.56) .

wxFrame::wxFrame

void **wxFrame**(*wxFrame* ***parent**, *char* ***title**, *int* **x = -1**, *int* **y = -1**,
int **width = -1**, *int* **height = -1**,
int **style = wxSDI || wxDEFAULT_FRAME**, *char* ***name = "frame"**)

Constructor. The *parent* parameter can be NULL or an existing frame; if an existing frame is used under Windows 3, the child frame is always on top of the parent, and will be iconized when the parent is iconized.

If *title* is non-NULL, it is placed on the window frame.

The style parameter may be a combination of the following, using the bitwise 'or' operator.

- **wxICONIZE**: display the frame iconized (minimized) (Windows only).
- **wxCAPTION**: puts a caption on the frame (under XView and Motif this is mandatory).
- **wxDEFAULT_FRAME**: defined as **(wxMINIMIZE_BOX || wxMAXIMIZE_BOX || wxTHICK_FRAME || wxSYSTEM_MENU || wxCAPTION)**.
- **wxMDI_CHILD**: specifies a Windows MDI (multiple document interface) child frame.
- **wxMDI_PARENT**: specifies a Windows MDI (multiple document interface) parent frame.
- **wxMINIMIZE**: identical to **wxICONIZE**.
- **wxMINIMIZE_BOX**: display a minimize box on the frame (Windows only).
- **wxMAXIMIZE**: display the frame maximized (Windows only).

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**)

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.

2.24.5 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**,

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 wxDialog-Box). 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.

```
  Add(wxMakeFormNewLine());

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

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

  AssociatePanel(panel);
}
```

wxForm::wxForm

void **wxForm**(*int* **useButtons = wxFORM_BUTTON_ALL**,
int **placeButtons = wxFORM_BUTTON_AT_TOP**)

Constructor. The value of *useButtons* determines which buttons are automatically created, and must be a bit list of the following identifiers:

- wxFORM_BUTTON_OK
- wxFORM_BUTTON_CANCEL
- wxFORM_BUTTON_UPDATE
- wxFORM_BUTTON_REVERT
- wxFORM_BUTTON_ALL

placeButtons may be used to specify whether the form buttons are placed the top or bottom of the form, and may be one of:

- wxFORM_BUTTON_AT_TOP
- wxFORM_BUTTON_AT_BOTTOM

wxForm::~wxForm

void **~wxForm**(*void*)

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

wxForm::Add

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

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

2.24.3 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.

2.24.4 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)));
```

wxFontList::AddFont

void **AddFont**(*wxFont* ***font**)

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

wxFontList::FindOrCreateFont

wxFont ***FindOrCreateFont**(*int* **point_size**, *int* **family**, *int* **style**, *int* **weight**, *Bool* **underline = FALSE**)

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

wxFontList::RemoveFont

void **RemoveFont**(*wxFont* ***font**)

Used by wxWindows to remove a font from the list.

2.24 wxForm: wxObject

2.24.1 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.

2.24.2 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.

wxFont::GetFamily

int **GetFamily**(*void*)

Gets the font family. See **wxFont** (section 2.22) for a list of valid family identifiers.

wxFont::GetPointSize

int **GetPointSize**(*void*)

Gets the point size.

wxFont::GetStyle

int **GetStyle**(*void*)

Gets the font style. See **wxFont** (section 2.22) for a list of valid styles.

wxFont::GetUnderlined

Bool **GetUnderlined**(*void*)

TRUE if the font is underlined.

wxFont::GetWeight

int **GetWeight**(*void*)

Gets the font weight. See **wxFont** (section 2.22) for a list of valid weight identifiers.

2.23 wxFontList: wxList

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

wxFontList::wxFontList

void **wxFontList**(*void*)

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

2.22 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**.
- Underlining. The value can be TRUE or FALSE.

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**, *Bool* **underline = FALSE**)

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.

Underlining only works under Windows 3 at present.

See also **wxDC::SetFont** (section 2.19) , **wxDC::DrawText** (section 2.19) and **wxDC::GetTextExtent** (section 2.19) .

All fonts are automatically added to the global pointer **wxTheFontList**. Call **wxFontList::FindOrCreateFont** (section 2.23) to return a previously-created font if possible.

wxFont::~wxFont

void **~wxFont**(*void*)

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

wxEvent::~wxEvent

void ~**wxEvent**(*void*)

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

wxEvent::eventClass

WXTYPE *eventClass*

The C++ class of the event, such as wxTYPE_COMMAND_EVENT. A single class may have many 'types'; it would be tedious to define a new C++ class for each type of similar event.

wxEvent::eventHandle

char * *eventHandle*

Handle of an underlying windowing system event handle, such as XEvent. Not guaranteed to be instantiated.

wxEvent::eventObject

wxObject * *eventObject*

The object (usually a window) that the event was generated from, or should be sent to.

wxEvent::eventType

WXTYPE *eventType*

The type of the event, such as wxEVENT_TYPE_BUTTON_COMMAND.

wxEvent::ReadEvent

pure virtual Bool **ReadEvent**(*istream&* **stream**)

Reads the event from the given input stream.

wxEvent::WriteEvent

pure virtual Bool **WriteEvent**(*ostream&* **stream**)

Writes the event to the given output stream.

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::SetTitle

void **SetTitle**(*char ** **title**)

Sets the title of the dialog box.

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.

2.21 wxEvent: wxObject

An event is a structure holding information about an event passed to a callback or member function. **wxEvent** used to be a multipurpose event object, and is now an abstract base class for events such as **wxCommandEvent** (section 2.16) (for panel item commands) and **wxMouseEvent** (section 2.40) (for mouse events on windows).

wxEvent::wxEvent

void **wxEvent**(*void*)

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

- **wxASBOLUTE_POSITIONING**: a hint to the windowing system not to try automatic layout (Motif only).
- **wxCAPTION**: puts a caption on the dialog box (under XView and Motif this is mandatory).
- **wxSTAY_ON_TOP**: stay on top of other windows (Windows only).
- **wxSYSTEM_MENU**: display a system menu (manadatory under XView and Motif).
- **wxTHICK_FRAME**: display a thick frame around the window (manadatory under XView and Motif).
- **wxVSCROLL**: give the dialog box a vertical scrollbar (XView only).

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

wxDialogBox::~wxDialogBox

void **~wxDialogBox**(*void*)

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

wxDialog::Centre

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

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

wxDialogBox::Create

void **Create**(*wxFrame* ***parent**, *char* ***title**, *Bool* **modal=FALSE**,
int **x=300**, *int* **y=300**, *int* **width=500**, *int* **height=500**,
int **style = wxCAPTION || wxSYSTEM_MENU || wxTHICK_FRAME**,
char ***name = "dialogBox"**)

Used for two-step dialog box construction. See **wxDialogBox::wxDialogBox** (section 2.20) for details.

wxDialogBox::GetTitle

char * **GetTitle**(*void*)

Gets the title of the dialog box.

wxDC::StartPage

Bool **StartPage**(*void*)

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

2.20 wxDialogBox: wxPanel

A dialog box is similar to a panel, in that it is a window which can be used for placing panel items, with the following exceptions:

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

Under XView, some panel items may display incorrectly in a modal dialog.

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 about dialogs in the user manual.

For a set of dialog convenience functions, including file selection, see **Dialog functions** (section 3.3) .

See also **wxPanel** (section 2.44) and **wxWindow** (section 2.59) for inherited member functions.

wxDialogBox::wxDialogBox

void **wxDialogBox**(*wxFrame* ***parent**, *char* ***title**, *Bool* **modal=FALSE**,
int **x=300**, *int* **y=300**, *int* **width=500**, *int* **height=500**,
int **style = wxCAPTION || wxSYSTEM_MENU || wxTHICK_FRAME**,
char ***name = "dialogBox"**)

Constructor.

If *title* is non-NULL, it is placed on the window frame.

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

The *style* parameter may be a combination of the following, using the bitwise 'or' operator:

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.

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.

See also **wxFont** (section 2.22) .

wxDC::SetLogicalFunction

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

Sets the current logical function for the canvas. This determines how a source pixel (from a pen or brush colour, or source device context if using **wxDC::Blit** (section 2.19)) combines with a destination pixel in the current device context.

The possible values and their meaning in terms of source and destination pixel values are as follows:

```
wxAND                   src AND dst
wxAND_INVERT            (NOT src) AND dst
wxAND_REVERSE           src AND (NOT dst)
wxCLEAR                 0
wxCOPY                  src
wxEQUIV                 (NOT src) XOR dst
wxINVERT                NOT dst
wxNAND                  (NOT src) OR (NOT dst)
wxNOR                   (NOT src) AND (NOT dst)
wxNO_OP                 dst
wxOR                    src OR dst
wxOR_INVERT             (NOT src) OR dst
wxOR_REVERSE            src OR (NOT dst)
wxSET                   1
wxSRC_INVERT            NOT src
wxXOR                   src XOR dst
```

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

wxDC::SetMapMode

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

The *mapping mode* of the device context defines the unit of measurement used to convert logical units to device units. Note that in X, text drawing isn't handled consistently with the mapping mode; a font is always specified in point size. However, setting the *user scale* (see **wxDC::SetUserScale** (section 2.19)) 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.

Gets the minimum vertical extent used in drawing commands so far.

wxDC::Ok

Bool **Ok**(*void*)

Returns TRUE if the DC is ok to use.

wxDC::SetBackground

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

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

wxDC::SetBackgroundMode

void **SetBackgroundMode**(*int* **mode**)

mode may be one of wxSOLID and wxTRANSARENT. This setting determines whether text will be drawn with a background colour or not.

wxDC::SetClippingRegion

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

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

See also **wxDC::DestroyClippingRegion** (section 2.19) .

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.

See also **wxBrush** (section 2.5) .

wxDC::SetFont

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

wxDC::GetTextExtent

void **GetTextExtent**(*char* ***string**, *float* ***w**, *float* ***h**,
float ***descent = NULL**, *float* ***externalLeading = NULL**)

Gets the dimensions of the string using the currently selected font. *string* is the text string to measure, *w* and *h* are the total width and height respectively, *descent* is the dimension from the baseline of the font to the bottom of the descender, and *externalLeading* is any extra vertical space added to the font by the font designer (usually is zero).

See also **wxFont** (section 2.22) , **wxDC::SetFont** (section 2.19) .

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::MaxX

float **MaxX**(*void*)

Gets the maximum horizontal extent used in drawing commands so far.

wxDC::MaxY

float **MaxY**(*void*)

Gets the maximum vertical extent used in drawing commands so far.

wxDC::MinX

float **MinX**(*void*)

Gets the minimum horizontal extent used in drawing commands so far.

wxDC::MinY

float **MinY**(*void*)

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

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

Draws a three-point spline using the current pen.

wxDC::DrawText

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

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

The coordinates refer to the top-left corner of the rectangle bounding the string. See **wxDC::GetTextExtent** (section 2.19) for how to get the dimensions of a text string, which can be used to position the text more precisely.

wxDC::EndDoc

void **EndDoc**(*void*)

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

wxDC::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 **wxDC::SetMapMode** (section 2.19)).

wxDC::GetSize

void **GetSize**(*int* ***width**, *int* ***height**)

This gets the maximum size of graphics drawn so far on the device context.

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 automatically closes the first and last points.

wxDC::DrawPoint

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

Draws a point using the current pen.

wxDC::DrawRectangle

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

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

wxDC::DrawRoundedRectangle

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

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

wxDC::DrawSpline

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

Destroys the current clipping region so that none of the DC is clipped. See also **wxDC::SetClippingRegion** (section 2.19) .

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::DrawArc

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

Draws an arc, centred on (*xc, yc*), with starting point (*x1, y1*) and ending at (*x2, y2*). The current pen is used for the outline and the current brush for filling the shape.

wxDC::DrawEllipse

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

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

wxDC::DrawIcon

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

Draw an icon on the display (does nothing if the device context is PostScript). This can be the simplest way of drawing bitmaps on a canvas.

wxDC::DrawLine

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

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

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.

From version 1.5, this class is abstract and cannot be used to create device context objects. Instead, use the derived classes **wxPostScriptDC** (section 2.49) , **wxMetaFileDC** (section 2.39) , **wxMemoryDC** (section 2.34) or **wxPrinterDC** (section 2.50) (Windows only) for constructing new device contexts. **wxCanvasDC** (section 2.9) is a context that cannot be created by the user but can be retrieved from a **wxCanvas** (section 2.8) by using **wxCanvas::GetDC** (section 2.8) .

When writing code to draw into a device context, use **wxDC** as a parameter whenever possible, to allow the most general use of your drawing code. You can then pass a device context object of any derived type. See the demo in `samples/hello` for code which uses this device-independent method of drawing.

Derived types of **wxDC** have documentation for specific features only, so refer to this section for most device context information.

wxDC::wxDC

void **wxDC**(*void*)

Constructor.

wxDC::~wxDC

void ~**wxDC**(*void*)

Destructor.

wxDC::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 **wxDC::SetLogicalFunction** (section 2.19)). See **wxMemoryDC** (section 2.34) for typical usage.

wxDC::Clear

void **Clear**(*void*)

wxDC::DestroyClippingRegion

void **DestroyClippingRegion**(*void*)

- 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.

2.19 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.

wxConnection::StartAdvise

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

Called by the client application to ask if an advise loop can be started with the server. Causes the server connection's **wxConnection::OnStartAdvise** (section 2.17) member to be called. Returns TRUE if the server okays it, FALSE otherwise.

wxConnection::StopAdvise

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

Called by the client application to ask if an advise loop can be stopped. Causes the server connection's **wxConnection::OnStopAdvise** (section 2.17) member to be called. Returns TRUE if the server okays it, FALSE otherwise.

2.18 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** (section 2.27) 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** (section 3.4) 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

wxConnection::OnPoke

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

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

wxConnection::OnRequest

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

Message sent to the server application when the client calls **wxConnection::Request** (section 2.17) . The server should respond by returning a character string from **OnRequest**, or NULL to indicate no data.

wxConnection::OnStartAdvise

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

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

wxConnection::OnStopAdvise

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

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

wxConnection::Poke

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

Called by the client application to poke data into the server. Can be used to transfer arbitrary data to the server. Causes the server connection's **wxConnection::OnPoke** (section 2.17) member to be called. Returns TRUE if successful.

wxConnection::Request

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

Called by the client application to request data from the server. Causes the server connection's **wxConnection::OnRequest** (section 2.17) member to be called. Returns a character string (actually a pointer to the connection's buffer) if successful, NULL otherwise.

wxConnection::Advise

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

Called by the server application to advise the client of a change in the data associated with the given item. Causes the client connection's **wxConnection::OnAdvise** (section 2.17) member to be called. Returns TRUE if successful.

wxConnection::Execute

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

Called by the client application to execute a command on the server. Can also be used to transfer arbitrary data to the server (similar to **wxConnection::Poke** (section 2.17) in that respect). Causes the server connection's **wxConnection::OnExecute** (section 2.17) member to be called. Returns TRUE if successful.

wxConnection::Disconnect

Bool **Disconnect**(*void*)

Called by the client or server application to disconnect from the other program; it causes the **wxConnection::OnDisconnect** (section 2.17) 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.

wxCommandEvent::GetSelection

int **GetSelection**(*void*)

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

wxCommandEvent::GetString

*char ** **GetString**(*void*)

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

wxCommandEvent::IsSelection

Bool **IsSelection**(*void*)

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

2.17 wxConnection: wxObject

A wxConnection object represents the connection between a client and a server. It can be created by making a connection using a **wxClient** object, or by the acceptance of a connection by a **wxServer** object. The bulk of a DDE (Dynamic Data Exchange) conversation (available in both Windows and UNIX) is controlled by calling members in a **wxConnection** object or by overriding its members.

An application should normally derive a new connection class from wxConnection, in order to override the communication event handlers to do something interesting.

See **wxClient** (section 2.12) , **wxServer** (section 2.52) , and the chapter on interprocess communication in the user manual.

wxConnection::wxConnection

void **wxConnection**(*void*)

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

Constructs a connection object. If no user-defined connection object is to be derived from wxConnection, then the constructor should not be called directly, since the default connection object will be provided on requesting (or accepting) a connection. However, if the user defines his or her own derived connection object, the **wxServer::OnAcceptConnection** (section 2.52) and/or **wxClient::OnMakeConnection** (section 2.12) 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.

wxCommandEvent::commandString

char * *commandString*

Contains a string corresponding to a listbox or choice selection.

wxCommandEvent::extraLong

long *extraLong*

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

wxCommandEvent::wxCommandEvent

void **wxCommandEvent**(*WXTYPE* **commandEventType**)

Constructor. *commandEventType* may be one of the following:

- **wxEVENT_TYPE_BUTTON_COMMAND**
- **wxEVENT_TYPE_CHECKBOX_COMMAND**
- **wxEVENT_TYPE_CHOICE_COMMAND**
- **wxEVENT_TYPE_LISTBOX_COMMAND**
- **wxEVENT_TYPE_TEXT_COMMAND**
- **wxEVENT_TYPE_MULTITEXT_COMMAND**
- **wxEVENT_TYPE_MENU_COMMAND**
- **wxEVENT_TYPE_SLIDER_COMMAND**
- **wxEVENT_TYPE_RADIOBOX_COMMAND**

wxCommandEvent::Checked

Bool **Checked**(*void*)

Returns TRUE or FALSE for a checkbox selection event.

wxCommandEvent::GetClientData

*char ** **GetClientData**(*void*)

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

wxColourDatabase::Initialize

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

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

2.15 wxColourMap: wxObject

Colourmap functionality has yet to be implemented under wxWindows. This class will be extended in the future.

wxColourMap::wxColourMap

void **wxColourMap**(*void*)

Constructor.

wxColourMap::~wxColourMap

void **~wxColour**(*void*)

Destructor.

2.16 wxCommandEvent: wxEvent

This event class contains information about panel item command events. It is passed to **wxFunction** (section 2.26) panel item callbacks. It can also be constructed by an application and used with **wxSendEvent** (section 3.5) to simulate a user command in a panel item.

wxCommandEvent::clientData

char * *clientData*

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

wxCommandEvent::commandInt

int *commandInt*

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

2.14 wxColourDatabase: wxObject

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

The colours in the standard database are as follows:

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

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

See also **wxColour** (section 2.13) .

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.

wxClient::OnMakeConnection

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

The type of **wxConnection** (section 2.17) returned from a **wxClient::MakeConnection** (section 2.12) call can be altered by deriving the **OnMakeConnection** member to return your own derived connection object. By default, an ordinary wxConnection object is returned.

The advantage of deriving your own connection class is that it will enable you to intercept messages initiated by the server, such as **wxConnection::OnAdvise** (section 2.17) . You may also want to store application-specific data in instances of the new class.

2.13 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** (section 2.14) for how a pointer to a predefined, named colour may be returned instead of creating a new colour.

Valid RGB values are in the range 0 to 255.

wxColour::wxColour

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

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

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

wxColour::Get

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

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

wxColour::Ok

Bool **Ok**(*void*)

Returns TRUE if the colour object is valid.

wxColour::Set

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

Sets the RGB value.

(identifying the server process). Calling **wxClient::MakeConnection** (section 2.12) implicitly creates an instance of **MyConnection** if the request for a connection is accepted, and the client then requests an *Advise* loop from the server, where the server calls the client when data has changed.

```
wxIPCInitialize();

char *server = "4242";
char hostName[256];
wxGetHostName(hostName, sizeof(hostName));

char *host = hostName;

if (argc > 1)
  server = argv[1];
if (argc > 2)
  host = argv[2];

// Create a new client
MyClient *client = new MyClient;
the_connection = (MyConnection *)client->MakeConnection(host, server, "IPC TEST");

if (!the_connection)
{
  wxMessageBox("Failed to make connection to server", "Client Demo Error");
  return NULL;
}
the_connection->StartAdvise("Item");
```

See also **wxServer** (section 2.52) , **wxConnection** (section 2.17) , the chapter on interprocess communication in the user manual, and the programs in `samples/ipc`.

wxClient::wxClient

void **wxClient**(*void*)

Constructs a client object.

wxClient::MakeConnection

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

Tries to make a connection with a server specified by the host (machine name under UNIX, ignored under Windows), service name (must contain an integer port number under UNIX), and topic string. If the server allows a connection, a wxConnection object will be returned. The type of wxConnection returned can be altered by overriding the **wxClient::OnMakeConnection** (section 2.12) member to return your own derived connection object.

Sets the choice by passing the desired string position.

wxChoice::SetStringSelection

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

Sets the choice by passing the desired string.

wxChoice::GetString

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

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

2.12 wxClient: wxIPCObject

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

To create a client which can communicate with a suitable server, you need to derive a class from wxConnection and another from wxClient. The custom wxConnection class will intercept communications in a 'conversation' with a server, and the custom wxServer is required so that a user-overriden **wxClient::OnMakeConnection** (section 2.12) member can return a wxConnection of the required class, when a connection is made.

For example:

```
class MyConnection: public wxConnection
{
 public:
  MyConnection(void)::wxConnection(ipc_buffer, 3999) {}
  ~MyConnection(void) { }
  Bool OnAdvise(char *topic, char *item, char *data, int size, int format)
  { wxMessageBox(topic, data); }
};

class MyClient: public wxClient
{
 public:
  MyClient(void) {}
  wxConnection *OnMakeConnection(void) { return new MyConnection; }
};
```

Here, **MyConnection** will respond to **OnAdvise** (section 2.17) messages sent by the server.

When the client application starts, it must first call **wxIPCInitialize** (section 3.6) before creating an instance of the derived wxClient. In the following, command line arguments are used to pass the host name (the name of the machine the server is running on) and the server name

wxChoice::Append

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

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

wxChoice::Clear

void **Clear**(*void*)

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

wxChoice::Create

Bool **Create**(*wxPanel **__parent__, *wxFunction* **func**, *char **__label__,
int **x = -1**, *int* **y = -1**, *int* **width = -1**, *int* **height = -1**, *int* **n**, *char **__choices__[]
int **style = 0**, *char **__name__ = **"choice"**)

Creates the choice for two-step construction. Derived classes should call or replace this function. See **wxChoice::wxChoice** (section 2.11) for further details.

wxChoice::FindString

int **FindString**(*char **__s__)

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

wxChoice::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**)

2.11 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.

See also **wxListBox** (section 2.33) .

wxChoice::wxChoice

void **wxChoice**(*void*)

Constructor, for use by derived classes.

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

Constructor, creating and showing a choice.

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

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

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

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

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

The *style* parameter is reserved for future use.

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

wxChoice::~wxChoice

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

Destructor, destroying the choice item.

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

The *style* parameter is reserved for future use.

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

wxCheckBox::~wxCheckBox

void **~wxCheckBox**(*void*)

Destructor, destroying the checkbox.

wxCheckBox::Create

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

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

Creates the checkbox for two-step construction. Derived classes should call or replace this function. See **wxCheckBox::wxCheckBox** (section 2.10) for details.

wxCheckBox::GetValue

Bool **GetValue**(*void*)

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

wxCheckBox::SetLabel

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

Sets the bitmap for a bitmap checkbox.

wxCheckBox::SetValue

void **SetValue**(*Bool* **state**)

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

wxCanvas::WarpPointer

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

Moves the pointer to the given position on the canvas.

2.9 wxCanvasDC: wxDC

A canvas device context is automatically created when a canvas is created. It can be retrieved from a canvas with **wxCanvas::GetDC** (section 2.8) and then drawn into. See **wxDC** (section 2.19) for further information on device contexts.

wxCanvasDC::wxCanvasDC

void **wxCanvasDC**(*wxCanvas* ***canvas**)

Constructor for internal use only.

2.10 wxCheckBox: wxItem

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

wxCheckBox::wxCheckBox

void **wxCheckBox**(*void*)

Constructor, used when deriving from this class.

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

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

Constructor, creating and showing a checkbox.

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

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

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

See also **wxPen** (section 2.46) .

wxCanvas::SetScrollbars

void **SetScrollbars**(*int* **horiz_pixels**, *int* **vert_pixels**, *int* **x_length**, *int* **y_length**, *int* **x_page**, *int* **y_page**, *int* **x_pos = 0**, *int* **y_pos = 0**)

Sets up vertical and/or horizontal scrollbars. The first pair of parameters give the number of pixels per 'scroll step', i.e. amount moved when the up or down scroll arrows are pressed. 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.

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

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

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

See also **wxCanvas::EnableScrolling** (section 2.8) , **wxCanvas::GetScrollUnitsPerPage** (section 2.8) , **wxCanvas::GetVirtualSize** (section 2.8) .

wxCanvas::SetTextBackground

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

Sets the current text background colour for the canvas. Do not delete *colour*. See also **wxCanvas::SetTextForeground** (section 2.8) .

wxCanvas::SetTextForeground

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

Sets the current text foreground colour for the canvas. Do not delete *colour*. See also **wxCanvas::SetTextBackground** (section 2.8) .

wxCanvas::ViewStart

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

Get the position at which the visible portion of the canvas starts. If either of the scrollbars is not at the home position, *x* and/or *y* will be greater than zero. Combined with **wxWindow::GetClientSize** (section 2.59) , the application can use this function to efficiently redraw only the visible portion of the canvas. The positions are in logical scroll units, not pixels, so to convert to pixels you will have to multiply by the number of pixels per scroll increment.

See also **wxCanvas::SetScrollbars** (section 2.8) .

See also **wxBrush** (section 2.5) .

wxCanvas::SetFont

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

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

See also **wxFont** (section 2.22) , **wxCanvas::DrawText** (section 2.8) .

wxCanvas::SetLogicalFunction

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

Sets the current logical function for the canvas. This determines how a source pixel (from a pen or brush colour, or source device context if using **wxDC::Blit** (section 2.19)) combines with a destination pixel in the current device context.

The possible values and their meaning in terms of source and destination pixel values are as follows:

```
wxAND                   src AND dst
wxAND_INVERT            (NOT src) AND dst
wxAND_REVERSE           src AND (NOT dst)
wxCLEAR                 0
wxCOPY                  src
wxEQUIV                 (NOT src) XOR dst
wxINVERT                NOT dst
wxNAND                  (NOT src) OR (NOT dst)
wxNOR                   (NOT src) AND (NOT dst)
wxNO_OP                 dst
wxOR                    src OR dst
wxOR_INVERT             (NOT src) OR dst
wxOR_REVERSE            src OR (NOT dst)
wxSET                   1
wxSRC_INVERT            NOT src
wxXOR                   src XOR dst
```

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.

Sent to the canvas when the user has initiated an event with the mouse. Derive your own class to handle this message. See **wxCanvas::OnChar** (section 2.8) for character events, and also **wxMouseEvent** (section 2.40) for how to access event information.

wxCanvas::OnPaint

void **OnPaint**(*void*)

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

wxCanvas::Scroll

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

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

See also **wxCanvas::SetScrollbars** (section 2.8) .

wxCanvas::SetBackground

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

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

See also **wxBrush** (section 2.5) .

wxCanvas::SetClippingRegion

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

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

See also **wxCanvas::DestroyClippingRegion** (section 2.8) .

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.

scrolls by a variable amount (e.g. if there are different font sizes) then physical scrolling messes up the display.

wxCanvas::GetDC

*wxCanvasDC ** **GetDC**(*void*)

Get a pointer to the canvas's device context. See **wxDC** (section 2.19) and **wxCanvasDC** (section 2.9) .

wxCanvas::GetScrollUnitsPerPage

void **GetScrollUnitsPerPage**(*int* ***x_page**, *int* ***y_page**)

Get the number of units per page, in each direction, as set by **wxCanvas::SetScrollbars** (section 2.8) . A value of zero indicates no scrolling in that direction.

wxCanvas::GetVirtualSize

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

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

Use **wxDC::DeviceToLogicalX** (section 2.19) and **wxDC::DeviceToLogicalY** (section 2.19) to translate these units to logical units.

wxCanvas::IsRetained

Bool **IsRetained**(*void*)

TRUE if the canvas has a backing bitmap.

wxCanvas::OnChar

void **OnChar**(*int* **key**, *int* **flags**)

This default handler interprets cursor key movement and scrolls the canvas accordingly. Override the function to change this behaviour (see **wxWindow::OnChar** (section 2.59) for details).

wxCanvas::OnEvent

void **OnEvent**(*wxMouseEvent&* **event**)

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::EnableScrolling

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

Enable or disable Windows scrolling in the given direction, where in this context scrolling is the physical transfer of bits up or down the screen when a scroll event occurs. If the application

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

wxCanvas::DrawArc

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

Draws an arc, centred on (*xc, yc*), with starting point (*x1, y1*) and ending at (*x2, y2*). The current pen is used for the outline and the current brush for filling the shape.

wxCanvas::DrawEllipse

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

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

wxCanvas::DrawLine

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

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

wxCanvas::DrawLines

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

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

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

wxCanvas::DrawPolygon

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

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 automatically closes the first and last points.

wxCanvas::wxCanvas

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

Constructor.

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

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

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

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

wxCanvas::~wxCanvas

void **~wxCanvas**(*void*)

Destructor.

wxCanvas::Create

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

Creates the canvas for two-step construction. Derived classes should call or replace this function. See **wxCanvas::wxCanvas** (section 2.8) for details.

wxCanvas::Clear

void **Clear**(*void*)

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

wxCanvas::DestroyClippingRegion

void **DestroyClippingRegion**(*void*)

wxButton::Create

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

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

Button creation functions called by the button constructors. Call these when a derived button class uses the zero-argument **wxButton** constructor, but can reuse the existing button creation code. See **wxButton::wxButton** (section 2.7) for details.

wxButton::SetDefault

void **SetDefault**(*void*)

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

Under Windows 3, only dialog box buttons respond to this function. As normal under Windows 3 and Motif, pressing return causes the default button to be depressed when the return key is pressed. See also **wxWindow::SetFocus** (section 2.59) which sets the keyboard focus for windows and text panel items, **wxPanel::OnDefaultAction** (section 2.44) and **wxPanel::GetDefaultItem** (section 2.44) .

wxButton::SetLabel

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

Sets the bitmap for a bitmap button.

2.8 wxCanvas: wxWindow

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

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

When you draw onto a canvas, you are really drawing onto a *device context* (see **wxDC** (section 2.19) , **wxCanvasDC** (section 2.9)). Although you can use the members of wxCanvas for drawing, it is much better to get the device context from the canvas (see **GetDC** (section 2.8)) and draw into that. Then, code which can draw into one device context can be reused for others, such as PostScript or memory device contexts (see **wxPostScriptDC** (section 2.49) and **wxMemoryDC** (section 2.34)).

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.

2.7 wxButton: wxItem

A button is a *panel item* that contains a text string or bitmap, and is one of the commonest elements of a GUI. It may be placed on a **dialog box** (section 2.20) or **panel** (section 2.44) .

wxButton::wxButton

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

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

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

func may be NULL; otherwise it is used as the callback for the button. Note that the cast (wxFunction) must be used when passing your callback function name, or the compiler may complain that the function does not match the constructor declaration. See **wxFunction** (section 2.26) .

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

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

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

wxButton::~wxButton

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

Destructor, destroying the button.

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).

2.6 wxBrushList: wxList

A brush list is a list containing all brushes which have been created. There is only one instance of this class: **wxTheBrushList**. Use this object to search for a previously created brush of the desired type and create it if not already found. In some windowing systems, the brush may be a scarce resource, so it is best to reuse old resources if possible. When an application finishes, all brushes will be deleted and their resources freed, eliminating the possibility of 'memory leaks'. See **wxBrush** (section 2.5) .

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**)

2.5 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 unless the colour is really black. If you wish the policy to be 'all non-white colours are black', as with pens, uncomment the piece of code documented in **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 **wxBrushList** (section 2.6) and **wxDC** (section 2.19) .

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. See **wxColour** (section 2.13) .

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*)

wxBitmap::wxBitmap

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

Constructs a (usually monochrome) bitmap from an X bitmap file (XBM).

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

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

void **wxBitmap**(*char ** **icon_name**)

Constructs a bitmap by passing a string name. *icon_name* refers to an XBM filename in X, a resource name in Windows 3.

wxBitmap::~wxBitmap

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

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

wxBitmap::GetDepth

int **GetDepth**(*void*)

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

wxBitmap::GetHeight

int **GetHeight**(*void*)

Gets the height of the bitmap in pixels.

wxBitmap::GetWidth

int **GetWidth**(*void*)

Gets the width of the bitmap in pixels.

wxBitmap::Ok

Bool **Ok**(*void*)

Returns TRUE if the bitmap was successfully created.

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.

2.4 wxBitmap: wxObject

This class encapsulates the concept of a platform-dependent bitmap, either monochrome or colour. Platform-specific methods for creating a **wxBitmap** structure from an existing file are catered for, and this is an occasion where conditional compilation will probably be required.

A bitmap created dynamically or loaded from a file can be selected into a memory device context (instance of **wxMemoryDC**). This enables the bitmap to be copied to a canvas or memory device context using **wxDC::Blit** (section 2.19) , or to be used as a drawing surface. The **wxToolBar** class was implemented using bitmaps, and the toolbar demo shows one of the toolbar bitmaps being used for drawing a miniature version of the graphic which appears on the main canvas.

wxWindows contains code to 'grey out' a bitmap when used in an insensitive panel item. Under X, this code is contained in the wxBitmap class. Under Windows, the user-contributed Fafa library is responsible for this.

See **wxMemoryDC** (section 2.34) for an example of drawing onto a bitmap.

The following shows the conditional compilation required to load a bitmap in X and in Windows 3. The alternative is to use the string version of the bitmap constructor, which loads a file under X and a resource under Windows 3, but has the disadvantage of requiring the X icon file to be available at run-time.

```
#ifdef wx_x
#include "aiai.xbm"
#endif
#ifdef wx_msw
  wxIcon *icon = new wxBitmap("aiai");
#endif
#ifdef wx_x
  wxIcon *icon = new wxBitmap(aiai_bits, aiai_width, aiai_height);
#endif
```

and documented in future versions.

wxApp::~wxApp

void ~**wxApp**(*void*)

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

wxApp::argc

int *argc*

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

wxApp::argv

char ** *argv*

Command line arguments (after environment-specific processing).

wxApp::wx_class

char * *wx_class*

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

wxApp::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.

GUI applications aren't all graphical wizardry. List and hash table needs are catered for by **wxList** (section 2.32) , **wxStringList** (section 2.54) and **wxHashTable** (section 2.28) . You will undoubtedly need some platform-independent **file functions** (section 3.1) , and you may find it handy to maintain and search a list of paths using **wxPathList** (section 2.45) . There's a **miscellany** (section 3.6) of operating system and other functions.

If you have several communicating applications, you can try out the DDE-like functions, by using the three classes **wxClient** (section 2.12) , **wxServer** (section 2.52) and **wxConnection** (section 2.17) . These use DDE under Windows, and a simulation using sockets under UNIX.

2.3 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. The alternative is to explicitly delete child frames in the top-level frame **OnClose** handler. See **wxFrame::OnClose** (section 2.25) .

In emergencies the *wxExit* function can be called to kill the application (see **wxExit** (section 3.6)).

An example of defining an application follows:

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

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

MyApp DerivedApp;
```

wxApp::wxApp

void **wxApp**(*int* **language = wxLANGUAGE_ENGLISH**)

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

The argument is a language identifier; this is an experimental feature and will be expanded

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.

2.2 A Rough Guide

To set a wxWindows application going, you'll need to derive a **wxApp** (section 2.3) class.

An application must have a top-level **wxFrame** (section 2.25) window (returned by **wxApp::OnInit** (section 2.3)), each frame containing one or more instances of **wxPanel** (section 2.44) , **wxTextWindow** (section 2.56) or **wxCanvas** (section 2.8) .

A frame can have a **wxMenuBar** (section 2.36) , a status line, and a **wxIcon** (section 2.27) for when the frame is iconized.

A **wxPanel** (section 2.44) is used to place items (classes derived from **wxItem** (section 2.30)) which are used for user interaction. Examples of items are **wxButton** (section 2.7) , **wxCheckBox** (section 2.10) , **wxChoice** (section 2.11) , **wxListBox** (section 2.33) , **wxSlider** (section 2.53) , **wxRadioBox** (section 2.51) , **wxSlider** (section 2.53) .

Instances of **wxDialogBox** (section 2.20) can also be used for panels, items and they have the advantage of not requiring a separate frame.

Instead of creating a dialog box and populating it with items, it is possible to choose one of the convenient **dialog functions** (section 3.3) , such as **wxMessageBox** (section 3.3) and **wxFileSelector** (section 3.3) .

If you want to draw arbitrary graphics, you'll need a **wxCanvas** (section 2.8) . In fact, you never draw directly onto a canvas—you use a *device context* (DC). **wxDC** (section 2.19) is the base for **wxCanvasDC** (section 2.9) , **wxMemoryDC** (section 2.34) , **wxPostScriptDC** (section 2.49) , **wxMemoryDC** (section 2.34) , **wxMetaFileDC** (section 2.39) and **wxPrinterDC** (section 2.50) . If your drawing functions have **wxDC** as a parameter, you can pass any of these DCs to the function, and thus use the same code to draw to several different devices. You can draw using the member functions of **wxDC**, such as **wxDC::DrawLine** (section 2.19) and **wxDC::DrawText** (section 2.19) . Control colour on a canvas (**wxColour** (section 2.13)) with brushes (**wxBrush** (section 2.5)) and pens (**wxPen** (section 2.46)).

On a canvas, you will probably need to intercept key events by overriding the **wxCanvas::OnChar** (section 2.8) member, and mouse events by overriding **wxCanvas::OnEvent** (section 2.8) .

Most modern applications will have an on-line, hypertext help system; for this, you need wxHelp and the **wxHelpInstance** (section 2.29) class to control wxHelp. To add sparkle, you might use the wxToolBar class (documented separately) which makes heavy use of the **wxBitmap** (section 2.4) .

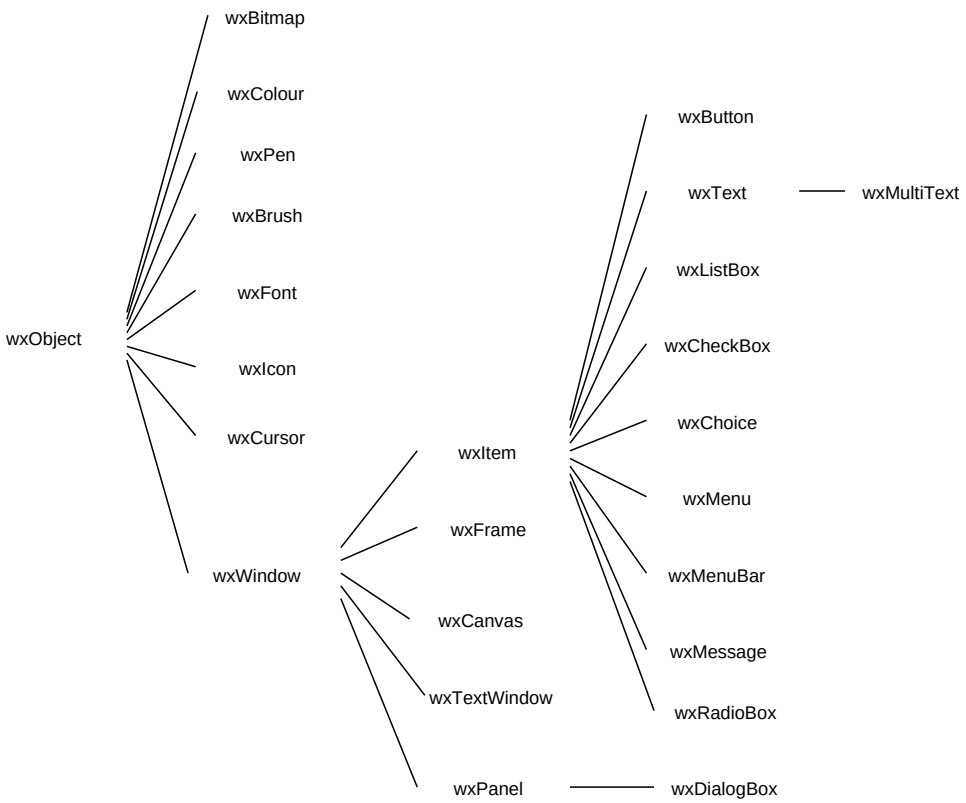
Figure 5.1: wxWindows class hierarchy

Chapter 2

wxWindows Class Reference

2.1 Class hierarchy

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



Chapter 1

This manual

wxWindows is a class library for C++ providing GUI (Graphical User Interface) and other facilities on more than one platform. This document gives detailed information about the classes and functions that make up the wxWindows API (Application Programming Interface). Please refer to the wxWindows user manual for a more general description of wxWindows.

Copyright notice

Copyright (c) 1993 Artificial Intelligence Applications Institute, The University of Edinburgh

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

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

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

2.45 wxPathList: wxList . . . 98
2.46 wxPen: wxObject . . . 99
2.47 wxPenList: wxList . . . 100
2.48 wxPoint: wxObject . . . 101
2.49 wxPostScriptDC: wxDC . . . 101
2.50 wxPrinterDC: wxDC . . . 102
2.51 wxRadioBox: wxItem . . . 102
2.52 wxServer: wxIPCObject . . . 105
2.53 wxSlider: wxItem . . . 106
2.54 wxStringList: wxList . . . 107
2.55 wxText: wxWindow . . . 108
2.56 wxTextWindow: wxWindow . . . 109
2.57 wxTimer: wxObject . . . 111
2.58 wxTypeTree: wxList . . . 112
2.59 wxWindow: wxObject . . . 115

3 Miscellaneous functions **121**

3.1 File functions . . . 121
3.2 String functions . . . 123
3.3 Dialog functions . . . 123
3.4 GDI functions . . . 125
3.5 System event functions . . . 126
3.6 Miscellaneous . . . 128

Index **133**

2.21 wxEvent: wxObject 50
2.22 wxFont: wxObject 52
2.23 wxFontList: wxList 53
2.24 wxForm: wxObject 54
2.24.1 The purpose of the form class 54
2.24.2 Constraints on form items 54
2.24.3 Form appearance 55
2.24.4 Example 55
2.24.5 Functions for making form items and constraints 58
2.25 wxFrame: wxWindow 60
2.26 wxFunction 66
2.27 wxIcon: wxObject 66
2.28 wxHashTable: wxObject 67
2.29 wxHelpInstance: wxClient 69
2.30 wxItem: wxWindow 71
2.31 wxKeyEvent: wxEvent 72
2.32 wxList: wxObject 73
2.33 wxListBox: wxItem 77
2.34 wxMemoryDC: wxObject 80
2.35 wxMenu: wxItem 81
2.36 wxMenuBar: wxItem 83
2.37 wxMessage: wxItem 86
2.38 wxMetaFile: wxObject 87
2.39 wxMetaFileDC: wxDC 87
2.40 wxMouseEvent: wxEvent 88
2.41 wxMultiText: wxText 93
2.42 wxNode: wxObject 93
2.43 wxObject 94
2.44 wxPanel: wxWindow 94

Contents

1 This manual **6**

2 wxWindows Class Reference **7**

2.1 Class hierarchy 7

2.2 A Rough Guide 9

2.3 wxApp: wxObject 10

2.4 wxBitmap: wxObject 12

2.5 wxBrush: wxObject 14

2.6 wxBrushList: wxList 15

2.7 wxButton: wxItem 16

2.8 wxCanvas: wxWindow 17

2.9 wxCanvasDC: wxDC 25

2.10 wxCheckBox: wxItem 25

2.11 wxChoice: wxItem 27

2.12 wxClient: wxIPCObject 29

2.13 wxColour: wxObject 31

2.14 wxColourDatabase: wxObject 32

2.15 wxColourMap: wxObject 33

2.16 wxCommandEvent: wxEvent 33

2.17 wxConnection: wxObject 35

2.18 wxCursor: wxObject 38

2.19 wxDC: wxObject 39

2.20 wxDialogBox: wxPanel 48

Reference Manual for wxWindows 1.50: A portable C++ GUI toolkit

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

July 1993