wxWindows 2

Programming cross-platform GUI applications in C++

1 Contents

2 Introduction

2.1 Why wxWindows?

Did you ever wanted to write a program in C++ that runs on Windows, Linux or Unix? How is this possible when every platform has it’s own framework, look and feel, or SDK? Of course, you don’t want to rewrite your application for each platform. This would create a maintenance nightmare.

wxWindows is the solution. wxWindows hides all the platform specific code for you. There are many platform independent frameworks available. So why should you choose for wxWindows?

- It is very complete. There are many utility classes.
- It is still heavily developed.
- Many compilers and platforms are supported: Windows, Linux, Mac, Unix
- There’s a lot of documentation.
- It’s free for personal and commercial use.
- Whenever possible, wxWindows uses the platform SDK. This means that a program compiled on Windows will have the look and feel of a Windows program, and when compiled on a Linux machine, it will get the look and feel of a Linux program.

2.2 History of wxWindows

Julian Smart started wxWindows in 1992 at the Artificial Intelligence Applications Institute, University of Edinburgh. In 1995, a port to Xt was released by Markus Holzem. In May 1997, the Windows and GTK+ ports were merged and put into a CVS repository.

2.3 Supported platforms

wxWindows supports many different platforms. wxWindows is developed as platform-independent as possible. Few classes (e.g. *wxTaskBar*) are platform-specific. Because these classes are wxWindows-compatible, it’s possible that you add support to some platform specific functionality in your application without sacrificing the cross-platform goal. Whenever possible, the native controls that are available on the platform are used. Unavailable controls are emulated.

In order to avoid architecture dependencies, wxWindows offers various architecture independent types and macros such as wxInt32, wxInt16,… wxWindows has functions for multi-platform file handling. wxPathList, for example, avoids the difficulty with the file separation character.

2.3.1 Windows 3.1

wxWindows is designed to make use of WIN32 features and controls. However most features work on the 16-bit windows. Don’t expect that the Windows95-specific classes such as *wxTaskBarIcon* work on Windows 3.1. 16-bit compilation is supported for Visual C++ 1.5 and Borland BC++ 4 to 5.

2.3.2 Windows 95/98/NT/2000

wxWindows is supported on Windows 95/98/NT and 2000.

The following compilers are supported:
- Visual C++ 1.5, 4.0., 5.0 and 6.0
- Borland C++ 4.5 and 5.0
- Borland C++ Builder 1.0 and 3.0
- Watcom C++ 10.6 (Win32)
- Cygwin b20
- Mingw32
- Metrowerks CodeWarrior 4

2.3.3 Unix with Motif

You need Motif version 1.2 or above. The following classes are not yet implemented: *wxSpinButton*, *wxCheckListBox*, *wxJoyStick* and *wxGLCanvas*. Some Windows-specific classes like *wxMiniFrame* and *wxTaskBar* are not likely to be implemented because there’s no sensible equivalent on Motif. Keep this in mind when developing a cross-platform GUI. Check the FAQ of wxWindows for Motif for more details.

2.3.4 Linux/Unix with GTK+

wxWindows 2 for GTK is a port of wxWindows to the free GTK+ toolkit. GTK+ is available for most flavors of Unix with X.

2.3.5 Mac

2.4 Installing wxWindows

Download the version specific for your platform at www.wxwindows.org. Extract the archive into a directory on your system, for example c:\wx. Don't use a pathname with spaces. Some compilers doesn't like that.

When you're going to use makefiles to build wxWindows, make sure you set the WXWIN environment variable. It should point to the installation directory. An example:

```
SET WXWIN=c:\wx
```

2.4.1 Compile wxWindows

wxWindows is a source-distribution, which means that you need to compile it before you can use it.

- ❑ Visual C++ 5.0/6.0

  Use the project file wxvc.dsp which you can find in the /src directory. Use Batch build to build both a Debug and a Release version of wxWindows.

  Previous versions of Visual C++ can use the makefile makefile.vc. For more information about how to do this read the install.txt file which you can find in the /doc/msw directory.

- ❑ Mingw32

  MinGW is a collection of header files and import libraries that allow you to use GCC and produce native Windows32 programs that do not rely on any 3rd-party DLLs. The current set of tools includes GNU Compiler Collection (GCC), GNU Debugger (GDB), GNU make, and other utilities.

  First, don't forget to set the WXWIN environment variable. Before you can compile wxWindows you have to perform the following tasks:
  - Download extra.zip from the remstar ftp site: ftp://www.remstar.com/pub/. (see folder \wxwin\ports\mingw32)
  - Extract cp.exe and rm.exe from extra.zip and put them in the bin-directory of your Mingw compiler.
  - When the PATH environment variable doesn't include the bin directory of the Mingw compiler, then you have to update mingw32.bat (in the src directory) to make sure the compiler can be found.
  - Run mingw32.bat
  - When necessary, update the makeg95.env file (see below). This file contains all the common settings for wxWindow programs.
  - Only for version lower then 2.2.9: The INT32 problem: Mingw32 already defines INT32. The problem is that it is defined as int, which is not the same as defined in wxWindows. Add the –DINT32_DEFINED compiler option in makeg95.env. Change the jinclude.h header file, which you can find in the jpeg directory, and add the following line at the top of the source code.
    ```
          typedef int INT32;
    ```
  - Change the working directory to src/msw and run "make –f makefile.g95".

Use forward slashes in the path, not backslashes!

- When you want to debug your applications with GDB then add –g to the debugflags. Your executables will be enormous, because debug information is created in each object file.

When the make program complains about some include files it cannot find, you can change the makeg95.env file as follows:

```
#ifeq ($(MINGW32),1)
#INC = -I$(WXINC) -I$(WXDIR)/contrib/include -I$(WXDIR)/src/png -I$(WXDIR)/src/jpeg -
I$(WXDIR)/src/zlib -I$(WXDIR)/src/xpm -I$(WXDIR)/src/tiff $(EXTRAINC) $(COMPPATHS)
#else
INC = -I$(WXINC) -I$(WXDIR)/contrib/include -I$(WXDIR)/src/png -I$(WXDIR)/src/jpeg -
I$(WXDIR)/src/zlib -I$(WXDIR)/src/xpm -I$(WXDIR)/src/tiff $(EXTRAINC) $(COMPPATHS) -
I$(WXDIR)/include/wx/msw/gnuwin32
#endif
```

Change it into:

```
#ifeq ($(MINGW32),1)
#WINC = -I$(WXINC) -I$(WXDIR)/contrib/include -I$(WXDIR)/src/png -I$(WXDIR)/src/jpeg -
I$(WXDIR)/src/zlib -I$(WXDIR)/src/xpm -I$(WXDIR)/src/tiff $(EXTRAINC) $(COMPPATHS)
#else
WINC = -I$(WXINC) -I$(WXDIR)/contrib/include -I$(WXDIR)/src/png -I$(WXDIR)/src/jpeg -
I$(WXDIR)/src/zlib -I$(WXDIR)/src/xpm -I$(WXDIR)/src/tiff $(EXTRAINC) $(COMPPATHS) -
I$(WXDIR)/include/wx/msw/gnuwin32
#endif

INC = $(WINC) -I C:/Development/programs/dev-c++/include/g++
```

You probably have to do this when you use the Dev-C++ IDE that contains the Mingw32 compiler.

2.5 Setting up your compiler

2.5.1 Visual C++

When you use wxWindows with Visual C++, you select new Windows 32 Application in the application wizard. Visual C++ provides a debug configuration and a release configuration for your project. To build your application with the wxWindow library you have to apply the following project settings.

❑ Debug configuration

C/C++ Tab

The **preprocessor definitions** field should contain the following symbols:

WIN32, _DEBUG, _WINDOWS, **__WINDOWS__, __WXMSW__, __WXDEBUG__, WXDEBUG=1, __WIN95__, __WIN32__**, WINVER=0x0400, STRICT

The **Use run-time library** should be set to **Debug Multithreaded DLL**.

Link Tab

The **Object/library modules** field should contain:

kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell.lib ole32.lib oleauto32.lib uuid.lib odbc32.lib odbccp32.lib comctl32.lib rpcrt4.lib winmm.lib **wxd.lib xpmd.lib pngd.lib zlibd.lib jpegd.lib tiffd.lib**

The **Ignore libraries** field should contain:

```
libcd.lib, libcid.lib, msvcrtd.lib
```

- Release configuration

  C/C++ Tab

  The **preprocessor definitions** field should contain the following symbols:

  ```
  WIN32, NDEBUG, _WINDOWS, __WINDOWS__, __WXMSW__, __WIN95__, __WIN32__,
  WINVER=0x0400, STRICT
  ```

  The **Use run-time library** should be set to **Multithreaded DLL**.

  Link Tab

  The **Object/library modules** field should contain:

  ```
  kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell.lib ole32.lib oleauto32.lib
  uuid.lib odbc32.lib odbccp32.lib comctl32.lib rpcrt4.lib winmm.lib wx.lib xpm.lib png.lib zlib.lib
  jpeg.lib tiff.lib
  ```

  The **Ignore libraries** field should contain:

  ```
  libc.lib, libci.lib, msvcrt.lib
  ```

- All configurations

  C/C++ Tab

  The Additional include directories field should contain:

  ```
  c:\<wx-path>\include;c:\<wx-path>\contrib\include
  ```

  Of course you can add other directories you wish to use in your project. You can skip this setting when you add those directories to the options ‘Directories’.

  Link Tab

  The **Additional library path** field should contain:

  ```
  c:\<wx-path>\lib, c:\<wx-path>\contrib\lib
  ```

  Replace the <wx-path> with the path where wxWindows is installed. You can skip this setting when you add those directories to the options ‘Directories’.

2.5.2 Mingw32

You have to create a resource file when you run your application on Windows. The only requirement is that you include wx.rc.

```
#include “wx/msw/wx.rc”
```

The name of the resource file should be the same as the target in your makefile. Create a makefile as shown in Example 1.

Example 1. A makefile for MingW.

```
WXDIR = $(WXWIN)

EXTRAINC=
EXTRACPPFLAGS=
EXTRALIBS=
EXTRALDFLAGS=
TARGET=YourApplicationName

OBJECTS=YourApplication.o YourFrame.o

include $(WXDIR)/src/makeprog.g95
```

Use the EXTRAINC to specify additional directories for header files. EXTRALIBS can be used to specify some extra libraries to link with and use EXTRACPPFLAGS to define additional preprocessor definitions. EXTRALDFLAGS is used to specify additional link flags and directories for searching libraries. OBJECTS is a list of the object files which are related to your source files.

Example 2 shows you how a makefile is created to use expat, the XML parser.

Example 2. Adding additional libraries to a makefile.

```
EXTRALIBS=-lexpat
EXTRALDFLAGS=-Lc:/development/c++/expat-1.95.2/libs
EXTRAINC=-Ic:/development/c++/expat-1.95.2/source/lib
```

Because the linker of MingW (ld) searches for libraries with a ".a" suffix, you probably have to rename some libraries, or you have to specify the full path when using the –l option flag.

2.6 Hello World

The first example is the well-known Hello World application. The application will show a window displaying 'Hello World' in its statusbar.

Each wxWindow application needs an object derived from *wxApp*. Each application overrides the *OnInit*() method for initializing the application. You can, for example, create your main window here. Example 3 shows the definition of HelloWorldApp.

Example 3. HelloWorldApp.h - The HelloWorldApp definition

```
#ifndef _HELLOWORLDAPP_H
#define _HELLOWORLDAPP_H

/**
 * The HelloWorldApp class
 * This class shows a window containing a statusbar with the text 'Hello World'
 */
class HelloWorldApp : public wxApp
{
public:
       virtual bool OnInit();
};

DECLARE_APP(HelloWorldApp)

#endif _HELLOWORLDAPP_H
```

For the main window, you use the *wxFrame* class. This class provides a window whose size and position can be changed by the user. It has thick borders and a title bar. In addition, you can provide it a menu bar, a statusbar and a toolbar. Example 4 shows the implementation of HelloWorldApp.

Example 4. HelloWorldApp.cpp - The implementation of HelloWorldApp

```
// For compilers that supports precompilation , includes “wx/wx.h”
#include “wx/wxprec.h”

#ifndef WX_PRECOMP
	#include “wx/wx.h”
#endif

#include “HelloWorldApp.h”

IMPLEMENT_APP(HelloWorldApp)

bool HelloWorldApp::OnInit()
{
	wxFrame *frame = new wxFrame((wxFrame*) NULL, -1, “Hello World”);
	frame->CreateStatusBar();
	frame->SetStatusText(“Hello World”);
	frame->Show(TRUE);
	SetTopWindow(frame);
	return true;
}
```

When you’re compiler supports precompiled headers, you can use the wxprec header file. When it doesn’t, you should include wx.h, which includes all necessary header files for wxWindow. You can also include each header file separately for each control.

The macros *DECLARE_APP* and *IMPLEMENT_APP* does the following for us:

- When the platform needs one, it creates a main() or WinMain() method.
- It creates the global method *wxGetApp*(). You can use this function to retrieve a reference to the one and only application object.

  Example:
  ```
  HelloWorldApp &app = (HelloWorldApp&) wxGetApp();
  ```

You could be wondering why the frame variable isn’t deleted somewhere. By setting the frame as the top window of the application, the application will delete the frame for us.

The cast of NULL to *wxFrame** is done because some compilers define NULL as 0L so that no conversion to pointers is allowed. This cast makes the code a little bit more portable.

After the frame is constructed, a statusbar is created with the *CreateStatusBar* method. The text of the statusbar is set to “Hello World”. Calling the *Show* method shows the frame. *Show* is a method of the wxWindow class which *wxFrame* derives from.

When the *OnInit*()-method returns false the application immediately stops. This way you can stop the application when something went wrong during the initialization phase.

3 Using wxFrame

The *wxFrame* class provides us a frame window. A frame window is a window whose size can be changed. It has a thick border, a title bar. Optionally it can have a menubar, a toolbar or a statusbar. A frame can act as a container for other controls except for another frame or a dialog. *wxFrame* is derived from *wxWindow*.

During this chapter we will create a small text editor.

3.1 Creating a frame

Normally you create your own class that derives from *wxFrame*. This way you can add functionality to your own frame class. In Example 5 you can see how this is done. The implementation is shown in Example 6.

Example 5. TextFrame.h - The TextFrame definition

```
#ifndef _TEXTFRAME_H
#define _TEXTFRAME_H

class TextFrame : public wxFrame
{
  public:
  /**
   * Constructor. Creates a new TextFrame
   */
   TextFrame(const wxChar title, int xpos, int ypos, int width, int height);

   /**
    * Destructor
    */
   ~TextFrame();
};

#endif _TEXTFRAME_H
```

The constructor of *wxFrame* is called from the TextFrame constructor. The constructor of *wxFrame* is defined as follows:

```
wxFrame(wxWindow* parent, wxWindowID id, const wxString& title,
        const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,
        long style = wxDEFAULT_FRAME_STYLE, const wxString& name = “frame”)
```

- **parent** is a pointer to the parent window. In this example, the frame doesn’t have any parent, so NULL is passed.
- **id** is the window identifier. –1 indicates the default.
- **title** is the name of the window. This will be shown in the frame’s title bar.
- **pos** is the window position. A value of (-1, -1) indicates the default (= wxDefaultPosition). Depending on the platform this default is chosen by the windowing system or wxWindow.
- **size** is the window size. A value of (-1, -1) indicates the default (= wxDefaultSize). Depending on the platform this default is chosen by the windowing system or wxWindow.
- **style** is the window style. wxFrame defines the following additional styles:

wxDEFAULT_FRAME_STYLE	A combination of the styles wxMINIMIZE_BOX, wxMAXIMIZE_BOX, wxRESIZE_BOX, wxSYSTEM_MENU and wxCAPTION.
wxICONIZE	The frame will be displayed minimized. (Only on windows)
wxCAPTION	A caption is shown.
wxMINIMIZE	The same as wxICONIZE
wxMINIMIZE_BOX	A minimize box will be shown.

wxMAXIMIZE	The frame will be displayed maximized. (Only on windows)
wxMAXIMIZE_BOX	A maximize box will be shown.
wxSTAY_ON_TOP	The frame will stay on top of other windows. (Only on windows)
wxSYSTEM_MENU	A system menu is displayed.
wxSIMPLE_BORDER	No border is displayed. (Only on GTK and windows)
wxRESIZE_BORDER	A resizable border is drawn (Unix only)
wxFRAME_FLOAT_ON_PARENT	The frame will be above the parent window in the z-order and is not shown in the task bar. (Windows and GTK only)
wxFRAME_TOOL_WINDOW	Shows a small caption bar. (Windows only)

- **name** is the name of the window. This parameter is used to associate a name with an item. This allows the application user to set Motif resource values for individual windows.

Example 6. TextFrame.cpp – The TextFrame implementation

```
// For compilers that supports precompilation , includes “wx/wx.h”
#include “wx/wxprec.h”

#ifndef WX_PRECOMP
       #include “wx/wx.h”
#endif
#include “TextFrame.h”

TextFrame::TextFrame(const wxChar *title, int xpos, int ypos, int width, int height)
       : wxFrame((wxFrame *) NULL, -1, title, wxPoint(xpos, ypos), wxSize(width, height))
{
}

TextFrame::~TextFrame()
{
}
```

Example 7. TextEditorApp.h – The TextEditorApp definition

```
#ifndef TEXTEDITORAPP_H
#define TEXTEDITORAPP_H
class TextEditorApp : public wxApp
{
public:
       /**
        * Initialize the application
        */
       virtual bool OnInit();
};

DECLARE_APP(TextEditorApp)

#endif // TEXTEDITORAPP_H
```

Example 8. TextEditorApp.cpp – The TextEditorApp implementation

```
// For compilers that supports precompilation , includes “wx/wx.h”
#include “wx/wxprec.h”

#ifndef WX_PRECOMP
       #include “wx/wx.h”
#endif
#include “TextEditorApp.h”
#include “TextFrame.h”

IMPLEMENT_APP(TextEditorApp)

bool TextEditorApp::OnInit()
{
       TextFrame *frame = new TextFrame(“Simple Text Editor”, 100, 100, 400, 300);
       frame->Show(TRUE);
       SetTopWindow(frame);
       return true;
}
```

3.2 Adding a control

Now that the frame window is created, you need to add a control for processing text. *wxTextCtrl* is the class you need. The frame class contains this control.

Example 9 shows the new definition of the TextFrame class. The only thing that's changed is that you add a member of the type *wxTextCtrl*. This member is initialized in the constructor.

Example 9. TextFrame.h – The TextFrame definition with wxTextCtrl

```
#ifndef _TEXTFRAME_H
#define _TEXTFRAME_H

class TextFrame : public wxFrame
{
  public:
  /**
   * Constructor. Creates a new TextFrame
   */
   TextFrame(const wxChar *title, int xpos, int ypos, int width, int height);

   /**
    * Destructor
    */
   ~TextFrame();
private:
   wxTextCtrl *m_pTextCtrl;
};

#endif _TEXTFRAME_H
```

Example 10 shows the new constructor of the TextFrame class. The parent of the *wxTextCtrl* member is the TextFrame, so we pass the this pointer. The text 'Type some text…' will be shown as the default text. Notice that the *wxTextCtrl's* constructor looks similar to the wxFrame's constructor. Each class derived from *wxWindow* follows the same constructor pattern.

Again, you don't see code for deleting the pointer of *wxTextCtrl*. You don't need to, because TextFrame is the parent that will delete all it's children when it is destroyed.

wxDefaultPosition is defined as *wxPoint*(-1, -1) and *wxDefaultSize* is defined as wxSize(-1, -1).

wxTE_MULTILINE is a specific window-style for a text control. This style indicates that *wxTextCtrl* allows multiple lines. See chapter 7.10 for more information about different styles.

Example 10. TextFrame.cpp – The TextFrame implementation with wxTextCtrl

```
// For compilers that supports precompilation , includes "wx/wx.h"
#include "wx/wxprec.h"

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

TextFrame::TextFrame(const wxChar *title, int xpos, int ypos, int width, int height)
       : wxFrame((wxFrame *) NULL, -1, title, wxPoint(xpos, ypos), wxSize(width, height))
{
       m_pTextCtrl = new wxTextCtrl(this, -1, wxString("Type some text..."),
                                    wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
}
```

```
TextFrame::~TextFrame()
{
}
```

When you build this project, you'll have an application that shows a window where you can enter some text. Try cut, paste, and you'll see that this small piece of code does a lot for you.

3.3 Adding a menubar

Naturally, you want to add menus to your application to aid the user in writing text. wxWindows provides the following classes for menus: *wxMenuBar* and *wxMenu*.

Each menu needs a unique ID. In Example 11 you see this is done by using an enum. #define for defining constants (e.g. #define MENU_FILE_MENU 1) isn't used because this can't guarantee you that you have unique ID's. It's quit easily to overlook some values and it's difficult to maintain when you have to insert new ID's.

Example 11. TextFrame.h - The TextFrame definition with menus

```
#ifndef _TEXTFRAME_H
#define _TEXTFRAME_H

class TextFrame : public wxFrame
{
  public:
  /**
   * Constructor. Creates a new TextFrame
   */
   TextFrame(const wxChar *title, int xpos, int ypos, int width, int height);

   /**
    * Destructor
    */
   ~TextFrame();
private:
   wxTextCtrl *m_pTextCtrl;
   wxMenuBar *m_pMenuBar;
   wxMenu *m_pFileMenu;
   wxMenu *m_pInfoMenu;

   enum
   {
     MENU_FILE_OPEN,
     MENU_FILE_SAVE,
     MENU_FILE_QUIT,
     MENU_INFO_ABOUT
   };
};

#endif _TEXTFRAME_H
```

In the implementation of TextFrame in Example 12 you see how the menubar is created in the constructor of TextFrame. A menu-item is added to the menu using the *Append* method of *wxMenu*. Note how the ampersand is used to indicate which character is used as the hotkey. When the menu is created you use *Append* method from *wxMenuBar* to append it to the menubar.

Example 12. TextFrame.cpp – The TextFrame implementation with menus

```
// For compilers that supports precompilation , includes "wx/wx.h"
#include "wx/wxprec.h"

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

TextFrame::TextFrame(const wxChar *title, int xpos, int ypos, int width, int height)
       : wxFrame((wxFrame *) NULL, -1, title, wxPoint(xpos, ypos), wxSize(width, height))
{
       m_pTextCtrl = new wxTextCtrl(this, -1, wxString("Type some text..."),
                                   wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);

       m_pMenuBar = new wxMenuBar();
       // File Menu
       m_pFileMenu = new wxMenu();
       m_pFileMenu->Append(MENU_FILE_OPEN, “&Open”);
       m_pFileMenu->Append(MENU_FILE_SAVE, “&Save”);
       m_pFileMenu->AppendSeparator();
       m_pFileMenu->Append(MENU_FILE_QUIT, “&Quit”);
       m_pMenuBar->Append(m_pFileMenu, “&File”);
       // About menu
       m_pInfoMenu = new wxMenu();
       m_pInfoMenu->Append(MENU_INFO_ABOUT, “&About”);
       m_pMenuBar->Append(m_pInfoMenu, “&Info”);

       SetMenuBar(m_pMenuBar);
}

TextFrame::~TextFrame()
{
}
```

3.4 Adding a Statusbar

Adding a statusbar to your frame is even easier. Call the *wxFrame*-method *CreateStatusBar* and you’ll have a statusbar. As parameter, you can pass a number that is used to create separate fields in the statusbar. With the SetStatusText method of wxFrame you set the text of the field in the statusbar.

Example 13. Creating a statusbar

```
CreateStatusBar(3);
SetStatusText(“Ready”, 0);
```

Example 13 creates a statusbar that contains 3 fields. The text “Ready” is displayed in the first field.

The statusbar can also be used to display a description of the selected menu. Example 14 shows you how to change Example 12. When you select a menu, the description is displayed in the first field of the statusbar.

Example 14. Menus with a description-text.

```
// File menu
m_pFileMenu = new wxMenu();
m_pFileMenu->Append(MENU_FILE_OPEN, “&Open”, “Opens an existing file”);
m_pFileMenu->Append(MENU_FILE_SAVE, “&Save”, “Save the content”);
m_pFileMenu->AppendSeparator();
m_pFileMenu->Append(MENU_FILE_QUIT, “&Quit”, “Quit the application”);
m_pMenuBar->Append(m_pFileMenu, “&File”);

// About menu
m_pInfoMenu = new wxMenu();
m_pInfoMenu->Append(MENU_INFO_ABOUT, “&About”, “Shows information about the application”);
m_pMenuBar->Append(m_pInfoMenu, “&Info”);
```

3.5 Processing Menu Events

Now that you have menus on your frame, you want to implement actions when the user selects a menu. You have to find a way to process the events that are triggered by the menus. The trick is that you attach a method of your class to the event you want to process. In previous versions of wxWindows, events were handled by supplying callback functions or by overloading virtual functions. From wxWindows 2.0 event tables are used instead. In this chapter, only the events generated by menus are explained. You’ll find more about event handling in chapter 4.

Each window that processes events needs to declare an event table. The *DECLARE_EVENT_TABLE* macro is used for this. You also need to define the methods for your actions. An event received from a menu is a wxCommandEvent. This is the type of the parameter of your methods. Example 15 contains the full definition of the TextFrame class.

Example 15. TextFrame.h – The TextFrame definition with an event table

```
#ifndef _TEXTFRAME_H
#define _TEXTFRAME_H


class TextFrame : public wxFrame
{
public:
  /**
   * Constructor. Creates a new TextFrame
   */
   TextFrame(const wxChar title, int xpos, int ypos, int width, int height);

   /**
    * Destructor
    */
   ~TextFrame();

   /**
    * Processes menu File|Open
    */
   void OnMenuFileOpen(wxCommandEvent &event);
   /**
    * Processes menu File|Save
    */
   void OnMenuFileSave(wxCommandEvent &event);
   /**
    * Processes menu File|Quit
    */
   void OnMenuFileQuit(wxCommandEvent &event);
   /**
    * Processes menu About|Info
    */
   void OnMenuInfoAbout(wxCommandEvent &event);

protected:
   DECLARE_EVENT_TABLE()
private:
   wxTextCtrl *m_pTextCtrl;
   wxMenuBar *m_pMenuBar;
   wxMenu *m_pFileMenu;
   wxMenu *m_pInfoMenu;

   enum
   {
     MENU_FILE_OPEN,
     MENU_FILE_SAVE,
     MENU_FILE_QUIT,
     MENU_INFO_ABOUT
   };
};

#endif _TEXTFRAME_H
```

The event table is placed in the implementation file. wxWindows helps you with a bunch of macros. The *BEGIN_EVENT_TABLE* macro is used to declare the beginning of the event table. Because you can have many event tables in your program you pass the name of the class that processes the events in the macro. To attach your method to the event, you use the *EVT_MENU* macro. The end of the event table is marked with the *END_EVENT_TABLE* macro.
Example 16 is the full implementation of the TextFrame class.

Example 16. TextFrame.cpp – The TextFrame implementation that processes menu events

```
// For compilers that supports precompilation , includes “wx/wx.h”
#include “wx/wxprec.h”

#ifndef WX_PRECOMP
       #include “wx/wx.h”
#endif
```

```
#include “TextFrame.h”

TextFrame::TextFrame(const wxChar *title, int xpos, int ypos, int width, int height)
       : wxFrame((wxFrame *) NULL, -1, title, wxPoint(xpos, ypos), wxSize(width, height))
{
       m_pTextCtrl = new wxTextCtrl(this, -1, wxString(“Type some text…”),
                                    wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);

       CreateStatusBar();
       m_pMenuBar = new wxMenuBar();
       // File Menu
       m_pFileMenu = new wxMenu();
       m_pFileMenu->Append(MENU_FILE_OPEN, “&Open”, “Opens an existing file”);
       m_pFileMenu->Append(MENU_FILE_SAVE, “&Save”, “Saves the file”);
       m_pFileMenu->AppendSeparator();
       m_pFileMenu->Append(MENU_FILE_QUIT, “&Quit, “Close the application”);
       m_pMenuBar->Append(m_pFileMenu, “&File”);
       // About menu
       m_pInfoMenu = new wxMenu();
       m_pInfoMenu->Append(MENU_INFO_ABOUT, “&About”, “Shows info about the application”);
       m_pMenuBar->Append(m_pInfoMenu, “&Info”);

       SetMenuBar(m_pMenuBar);
}

TextFrame::~TextFrame()
{
}

BEGIN_EVENT_TABLE(TextFrame, wxFrame)
   EVT_MENU(MENU_FILE_OPEN, TextFrame::OnMenuFileOpen)
   EVT_MENU(MENU_FILE_SAVE, TextFrame::OnMenuFileSave)
   EVT_MENU(MENU_FILE_QUIT, TextFrame::OnMenuFileQuit)
   EVT_MENU(MENU_INFO_ABOUT, TextFrame::OnMenuInfoAbout)
END_EVENT_TABLE

void TextFrame::OnMenuFileOpen(wxCommandEvent &event)
{
  wxLogMessage(“File Open Menu Selected”);
}

void TextFrame::OnMenuFileSave(wxCommandEvent &event)
{
  wxLogMessage(“File Save Menu Selected”);
}

void TextFrame::OnMenuFileQuit(wxCommandEvent &event)
{
  Close(FALSE);
}

void TextFrame::OnMenuInfoAbout(wxCommandEvent &event)
{
  wxLogMessage(“File About Menu Selected”);
}
```

Try this program and see what the results are. The Open and Save menu items of the File menu are not yet implemented. How you can open, read and save a file is explained in chapter 24 and how you can use common dialogs for opening and saving files is explained in chapter 5.

4 Event Handling

In chapter 3.5 on page 17 you've already learned how to do the event handling for menus. In this chapter, you'll learn how event handling works and how to handle other events.

4.1 Introduction

An event is something that occurred in your application or outside your application. An event might be initiated by the user, another application, the operating system… You need a mechanism to react on the events you're interested in.

In previous versions of wxWindows, events were handled by the application either by supplying a callback function or by overriding virtual member functions. From wxWindows version 2.0, event tables are used instead. An event table, as seen in Example 17, tells wxWindows how to map events to member functions. An event table is declared in the implementation file (cpp).

Example 17. An event table

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
   EVT_MENU(wxID_EXIT, MyFrame::OnExit)
   EVT_SIZE(MyFrame::OnSize)
   EVT_BUTTON(BUTTON1, MyFrame::OnButton1)
END_EVENT_TABLE()
```

The above event-table tells wxWindows when it receives a WM_SIZE event for your frame to call the OnSize member function of the MyFrame class. The *BEGIN_EVENT_TABLE* macro declares that MyFrame, which is derived from wxFrame, owns this event table.

The member function that handles an event does not have to be virtual. Actually the event handler ignores the virtual declaration. These functions have a similar form: the return type is void and they receive an event parameter. The type of this parameter depends on the event. For size events, *wxSizeEvent* is used. For menu commands and most control commands (e.g. a button), *wxCommandEvent* is used. When controls get more complicated, they use their own event class.

In the class definition, a *DECLARE_EVENT_TABLE*-macro must be placed. See Example 15 on page 18 for an example.

All these macros are used to hide the complexity of the event-handling system. For a complete list of these event-macros, look at Appendix A: Event Macros on page 115.

4.2 How it works

When an event is received, wxWindows calls the ProcessEvent method of wxEventHandler on the first handler object belonging to the window that generates the event. wxWindow (and therefore all window classes) is derived from *wxEventHandler* . *ProcessEvent* searches the event in the event tables and calls zero or more event handler function(s). These are the steps in processing an event:

1. When the object is disabled (with *SetEvtHandlerEnabled* of *wxEvtHandler*) the function skips to step 6.
2. When the object is a *wxWindow*, *ProcessEvent* is recursively called on the window's *wxValidator*. If this returns true, the function exists.
3. *SearchEventTable* is called for this event handler. When this fails, the base class is tried, and so on. When no more tables exists or an appropriate function is found the function exits. When a function is found, it's executed.

4. The search is applied down the entire chain of event handlers. When this succeeds (meaning the event is processed) the function exits.
5. When the object is a wxWindow and the event is a *wxCommandEvent*, *ProcessEvent* is recursively applied to the parent window's event handler. When this returns true, the function exits. This way a button click will be processed by the parent of the button and not by the button itself.
6. Finally, *ProcessEvent* is called on the *wxApp* object.

4.3 Event Skipping

ProcessEvent exits when it finds a function to handle the event. This means that when your class reacts on an event, the base class does not get the event. Sometimes this is not desired. This problem can be avoided with the *Skip* method of the *wxEvent* class, which is the base class of each event type. This way the search for event handlers is continued.

Example 18 defines a text window, which only accepts numeric characters.

Example 18. NumTextCtrl.h – The definition of a numeric text control

```
class NumTextCtrl : public wxTextCtrl
{
public:
   NumTextCtrl(wxWindow *parent);
   void OnChar(wxKeyEvent& event);
protected:
   DECLARE_EVENT_TABLE()
};
```

Example 19. NumTextCtrl.cpp – The implementation of a numeric text control

```
// For compilers that supports precompilation , includes "wx/wx.h"
#include "wx/wxprec.h"

#ifndef WX_PRECOMP
       #include "wx/wx.h"
#endif
#include "NumTextCtrl.h"
#include <ctype.h>

NumTextCtrl::NumTextCtrl(wxWindow *parent) : wxTextCtrl(parent, -1)
{
}

void NumTextCtrl::OnChar(wxKeyEvent& event)
{
   if ( isdigit(event.GetKeyCode()) )
   {
     // Numeric characters are allowed, so the base class wxTextCtrl
     // is allowed to process the event. This is done using the Skip() method.
     event.Skip();
   }
   else
   {
     // Character is not allowed.
     wxBell();
   }
}

BEGIN_EVENT_TABLE(NumTextCtrl, wxTextCtrl)
   EVT_CHAR(NumTextCtrl::OnChar)
END_EVENT_TABLE()
```

When NumericTextCtrl receives a key event, the keycode is checked. When the entered key is numeric, the base class wxTextCtrl may process the event. That's why the Skip method is called on the event. You must call the Skip method here, because otherwise no key would be processed.

4.4 Vetoing an event

Some events can be vetoed. When you veto an event, the event is not further processed. Example 20 shows you how the close event of the Simple Text Editor is vetoed when the text is changed in a text control. The user can't close the frame until the text is saved.

Example 20. Vetoing an event.

```
void TextFrame::OnClose(wxCloseEvent& event)
{
  bool destroy = true;

  if ( event.CanVeto() )
  {
    if ( m_pTextCtrl->IsModified() )
    {
      wxMessageDialog *dlg =
               new wxMessageDialog(this, "Text is changed!\nAre you sure you want to exit?",
                                   "Text changed!!!", wxYES_NO | wxNO_DEFAULT);
      int result = dlg->ShowModal();
      if ( result == wxID_NO )
      {
        event.Veto();
        destroy = false;
      }
    }
  }
  if ( destroy )
  {
    Destroy();
  }
```

When does *CanVeto* return false? You can't veto an event when the task manager ends your program, when the user logs off, …

4.5 Plug an event handler

Consider the following problem: you want to log each menu command. You could create a function that is called in every command event handler function. The problem with this solution is that this will be hard to maintain. When you add a new menu item and you forget to call the function the menu command is not logged.

The solution for this problem is to create a new event handler and plug it into a wxWindow class. To do this, you create a new class derived from *wxEvtHandler*. Within this new class, you do the same as you would when you do event handling for a normal window. The code is shown in Example 21 and Example 22.

Example 21. LogEventHandler.h - The definition of the LogEventHandler.

```
#ifndef _LogEventHandler_H
#define _LogEventHandler_H

class LogEventHandler : public wxEvtHandler
{
public:
  LogEventHandler() : wxEvtHandler()
  {
  }
  virtual ~LogEventHandler()
  {
  }
protected:
  DECLARE_EVENT_TABLE()
  void OnMenu(wxMenuEvent &event);

private:
};

#endif // _LogEventHandler_H
```

Example 22. LogEventHandler.cpp - The implementation of the LogEventHandler.

```
// For compilers that supports precompilation , includes "wx/wx.h"
#include "wx/wxprec.h"

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

#include "LogEventHandler.h"

void LogEventHandler::OnMenu(wxMenuEvent &event)
{
  wxLogMessage("Someone selected a menu item");
  event.Skip();
}

BEGIN_EVENT_TABLE(LogEventHandler, wxEvtHandler)
    EVT_MENU_RANGE(-1, -1, LogEventHandler::OnMenu)
END_EVENT_TABLE()
```

With the *EVT_MENU_RANGE* macro, you can handle a range of menus. The first two parameters specify the range of id's you want to handle. –1 means that you want to handle all the menu items.

The next step is to push the new event handler in the stack of handlers. This is done using the *PushEventHandler* method of the wxWindow class. Removing the event handler is done *PopEventHandler* method.

```
  PushEventHandler(new LogEventHandler());
```

The *PopEventHandler* method has a boolean parameter. When this parameter is true, wxWindows will delete the event handler. Note that when you don't pop the event handler, wxWindows will try to delete the event handler when the window is destroyed. This can generate access violation errors when you didn't create the event handler on the heap. Avoid these errors by calling *PopEventHandler* with parameter false in the destructor of your class.

5 Common Dialogs

5.1 Introduction

In chapter 3.5 on page 17 you provided menu commands for opening and saving a text-file. To allow the user to select or enter a filename you can use a common dialog: wxFileDialog. When common dialogs are used, you give the user a consistent interface. No matter which application the user uses, they can expect to see a dialog that looks familiar.

wxWindows provides the following common dialogs:

- *wxFileDialog*
  Shows a dialog for saving or opening a file.
- *wxColourDialog*
  Shows a dialog for selecting colors.
- *wxFontDialog*
  Shows a dialog for selecting a font.
- *wxPrintDialog*
  Shows a dialog for printing and setting up a printer.
- *wxDirDialog*
  Shows a dialog for selecting a directory.
- *wxTextEntryDialog*
  Shows a dialog that request the user to enter a one-line text.
- *wxMessageDialog*
  Shows a dialog with a single or multi-line message, with a choice of Ok, Yes, No and Cancel buttons.
- *wxSingleChoiceDialog*
  Shows a dialog with a list of strings, and allows the user to select one.
- *wxWizard*
  Shows a wizard dialog.

wxWindows is a cross-platform framework. When a common dialog is not available on the particular platform, wxWindow will present a generic dialog.

You'll see in the following examples that for every dialog the Destroy method is called instead of deleting the pointer. This is done for consistency purposes. wxWindows controls should always be created on the heap and they are safely removed when you call the Destroy method. This is because wxWindows sometimes delays deletion until all events are processed. This way wxWindows avoids that events are sent to non-existing windows.

5.2 wxFileDialog

wxFileDialog is used to present the user a dialog where he can select or enter a filename when he wants to open or save a file. The constructor of wxFileDialog looks like this:

```
wxFileDialog(wxWindow *parent, const wxString& message = “Choose a file”,
             const wxString& defaultDir = “”, const wxString& defaultFile = “”,
             const wxString& wildcard = “*.*”, long style = 0,
             const wxPoint& pos = wxDefaultPosition);
```

- **parent** is the window that owns this dialog.
- **message** is the title of the dialog.
- **defaultDir** is the default directory.
- **defaultFile** is a default filename.
- **wildcard** is something like “*.*” or “*.txt”. This is used for filtering files. “*.txt” will show files only when they have txt as extension. It's possible to list multiple extensions and to add a description for each extension. This list is displayed in the

dialog's "files of type" combo-box. Use the following syntax for this: description|extension. The following example gives the user the possibility to view all files, all text-files and all bitmap files: "All files(*.*)|*.*|Text files(*.txt)|*.txt|Bitmap files(*.bmp)|*.bmp"

- **style** is the type of the dialog.

wxOPEN	This is used for the open dialog.
wxSAVE	This is used for the save dialog.
wxHIDE_READONLY	Used to hide read-only files.
wxOVERWRITE_PROMPT	Used to ask for confirmation when a file will be overridden.
wxMULTIPLE	For open dialog only: Allows multiple file selections

- **pos** is not used.

Example 23 completes the text editor you've written in chapter 3. The ShowModal method displays the dialog. "Modal" means that no other window of the application can get the focus until the dialog is closed. ShowModal returns wxID_OK when the user presses OK and wxID_CANCEL otherwise. GetFilename is used to retrieve the name of the file that is selected. GetPath will return the full path (directory and filename) of the selected file. Loading and saving text is very simple when the wxTextCtrl class is used. You can use the methods LoadFile and SaveFile.

Example 23. Implementation of the File Open and File Save menu actions.

```
void TextFrame::OnMenuFileOpen(wxCommandEvent &event)
{
  wxFileDialog *dlg = new wxFileDialog(this, "Open a text file",
                                       "", "", "All files(*.*)|*.*|Text Files(*.txt)|*.txt",
                                       wxOPEN, wxDefaultPosition);
  if ( dlg->ShowModal() == wxID_OK )
  {
    m_pTextCtrl->LoadFile(dlg->GetFilename());
    SetStatusText(dlg->GetFilename(), 0);
  }
  dlg->Destroy();
}

void TextFrame::OnMenuFileSave(wxCommandEvent &event)
{
  wxFileDialog *dlg = new wxFileDialog(this, "Save a text file",
                                       "", "", "All files(*.*)|*.*|Text Files(*.txt)|*.txt",
                                       wxSAVE, wxDefaultPosition);
  if ( dlg->ShowModal() == wxID_OK )
  {
    m_pTextCtrl->SaveFile(dlg->GetPath());
    SetStatusText(dlg->GetFilename(), 0);
  }
  dlg->Destroy();
}
```

5.3 wxColourDialog

wxColourDialog implements a dialog where the user can select a color. wxColourDialog is used together with wxColourData. wxColourData is used to set the current color in the dialog and to get the selected color. The constructor of wxColourDialog looks like this:

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

- **parent** is the window that owns the dialog.
- **data** contains color information. It contains the default color, custom colors and under Windows you can tell it to show the full dialog with custom color selection controls.

Example 24 adds a menu-item to the application to allow the user to select another background color. The current background-color is assigned to colourData. When the application runs on a Windows-system the full dialog is displayed.

Example 24. Selecting a color with wxColourDialog

```
void TextFrame::OnMenuOptionBackgroundColor(wxCommandEvent &event)
{
  wxColourData colourData;
  wxColour colour = m_pTextCtrl->GetBackgroundColour();
  colourData.SetColour(colour);
  colourData.SetChooseFull(true);
  wxColourDialog *dlg = new wxColourDialog(this, &colourData);
  if ( dlg->ShowModal() == wxID_OK )
  {
    colourData = dlg->GetColourData();
    m_pTextCtrl->SetBackgroundColour(colourData.GetColour());
    m_pTextCtrl->Refresh();
  }
  dlg->Destroy();
}
```

5.4 wxFontDialog

wxFontDialog is used to show the user a dialog where he can select a font. This dialog is used together with wxFontData, wxFont and wxColour. wxFontData is used to set the current font in the dialog and to get the selected font. The constructor of the wxFontDialog looks like this:

```
wxFontDialog(wxWindow *parent, wxFontData *data = NULL);
```

- **parent** is the window that owns the dialog
- **data** contains information to show the current font and to retrieve information on the selected font.

Example 25 implements a new menu item for changing the font in the editor. First of all fontData is filled with the current font and color. When the application runs on a Windows-system, the user will see a help-button. When the user selects the font, fontData is used to set the new font and foregroundcolor.

Example 25.Selecting a font with wxFontDialog

```
void TextFrame::OnMenuOptionFont(wxCommandEvent& event)
{
  wxFontData fontData;
  wxFont font;
  wxColour colour;

  font = m_pTextCtrl->GetFont();
  fontData.SetInitialFont(font);
  colour = m_pTextCtrl->GetForegroundColour();
  fontData.SetColour(colour);
  fontData.SetShowHelp(true);

  wxFontDialog *dlg = new wxFontDialog(this, &fontData);
  if ( dlg->ShowModal() == wxID_OK )
  {
    fontData = dlg->GetFontData();
    font = fontData.GetChosenFont();
    m_pTextCtrl->SetFont(font);
    m_pTextCtrl->SetForegroundColour(fontData.GetColour());
    m_pTextCtrl->Refresh();
  }
  dlg->Destroy();
}
```

5.5 *wxPrintDialog*

wxPrintDialog is used to print a document and to set the printer options. It's used together with wxPrinterData and wxPrinterDC

5.6 *wxDirDialog*

wxDirDialog is used to select a directory. The constructor looks like this:

```
wxDirDialog(wxWindow *parent, const wxString& message = "Choose a directory",
            const wxString &defaultPath = "", long style = 0,
            const wxPoint& pos = wxDefaultPosition);
```

- **parent** is the window that owns this dialog.
- **message** is the title of this dialog.
- **defaultPath** is used to select the default in the dialog.
- **style** is not used.
- **pos** is not used.

The following example shows an implementation of a menu item to set a new working directory. wxGetCwd is used to retrieve the current working directory. When the user selected a directory you can retrieve the name of the directory with the GetPath method. wxSetWorkingDirectory is used to set the new working directory.

```
void TextFrame::OnMenuOptionDirectory(wxCommandEvent& event)
{
  wxDirDialog *dlg = new wxDirDialog(this, "Select a new working directory", wxGetCwd());
  if ( dlg->ShowModal() == wxID_OK )
  {
    wxSetWorkingDirectory(dlg->GetPath());
  }
  dlg->Destroy();
}
```

5.7 *wxTextEntryDialog*

wxTextEntryDialog is a dialog that requests a one-line text from the user. The constructor looks like this:

```
wxTextEntryDialog(wxWindow *parent, const wxString& message,
                  const wxString& caption = "Please enter text", const wxString& defaultValue,
                  long style = wxOK | wxCANCEL | wxCENTRE,
                  const wxPoint& pos = wxDefaultPosition);
```

- **parent** is the window that owns the dialog
- **message** is shown in the dialog.
- **caption** is the title of the dialog
- **defaultValue** is the default for the textfield
- **style** is the dialog style. All styles of wxTextCtrl can be specified here.

wxOK	An OK button is displayed.
wxCANCEL	A Cancel Button is displayed.
wxCENTRE	Centers the dialog.

- **pos** is the position of the default.

As you can see in the following example wxTextEntryDialog is ideal to prompt the user for a password.

```
wxTextEntryDialog *dlg = new wxTextEntryDialog(this, "Enter your password",
                                               "Enter your password", "",
                                               wxOK | wxCANCEL | wxCENTRE | wxTE_PASSWORD);
if ( dlg->ShowModal() == wxID_OK )
{
  // Check the password
}
else
{
  // Stop the application
}
dlg->Destroy();
```

5.8 wxMessageDialog

wxMessageDialog can be used to show a single or multi-line message, with a choice of OK, Yes, No or Cancel. The constructor looks like this:

```
wxMessageDialog(wxWindow *parent, const wxString& message,
                const wxString& caption = "Message box",
                long style = wxOK | wxCANCEL | wxCENTRE,
                const wxPoint& pos = wxDefaultPosition);
```

- **parent** is the parent window of the dialog
- **message** is the message to show. This can be a multi-line message.
- **caption** is the title of the dialog
- **style** is the dialog style

wxOK	Show an OK button.
wxCANCEL	Show a Cancel button.
wxYES_NO	Show Yes and No buttons.
wxYES_DEFAULT	Used with wxYES_NO and makes the Yes button the default. This is the default.
wxNO_DEFAULT	Used with wxYES_NO and makes the No button the default.
wxCENTRE	Centres the message.
wxICON_EXCLAMATION	Shows an exclamation mark icon
wxICON_HAND	Shows an error icon.
wxICON_ERROR	Shows an error icon.
wxICON_QUESTION	Shows a question mark icon.
wxICON_INFORMATION	Shows an information icon.

- **pos** is the dialog position.

On page 22 you see Example 20 that uses wxMessageDialog to ask the user if he is sure to exit the application.

5.9 wxSingleChoiceDialog

To Do

5.10 wxWizard

To Do

6 Dialogs

A dialog box is a window with a title bar. The window can be moved and it can contain controls and other windows. A dialog can be modal or modeless. A modal dialog blocks the flow of the program until the dialog is closed.

6.1 Programming dialogs

wxDialog is the base class for all dialogs. Your custom dialog derives from *wxDialog*. The constructor of wxDialog looks like this:

```
wxDialog(wxWindow* parent, wxWindowID id, const wxString& title,
         const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,
         long style = wxDEFAULT_DIALOG_STYLE, const wxString& name = "dialogBox")
```

- **parent** is the window that owns the dialog.
- **id** is a unique window id.
- **title** is the caption of the dialog.
- **pos** is the position of the dialog
- **size** is the size of the dialog.
- **style** is the window style of the dialog.

wxDIALOG_MODAL	Specifies that the dialog will be modal
wxCAPTION	Puts a caption on the dialog (default)
wxDEFAULT_DIALOG_STYLE	Equivalent to a combination of wxCAPTION, wxSYTEM_MENU, wxTHICK_FRAME
wxRESIZE_BORDER	Displays a resizable frame around the dialog
wxSYSTEM_MENU	Displays a system menu
wxTHICK_FRAME	Displays a thick frame around the dialog
wxSTAY_ON_TOP	The dialog stays on top of all other windows. (Windows only)
wxNO_3D	Specifies that by default no child controls should have 3D borders.

- **name** is the control name.

When you use dialogs in your program, wxWindows needs a resource file. When you don't need extra icons, it can be as simple as you can see in Example 26.

Example 26. TextEditorApp.rc - A simple resource file.

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

Example 27 shows you the definition of a dialog that shows the about-information of the simple text editor you have developed in the previous chapters. Example 28 shows the implementation.

Example 27. AboutDialog.h – The AboutDialog definition

```
#ifndef _ABOUTDIALOG_H
#define _ABOUTDIALOG_H

class AboutDialog : public wxDialog
{
public:
  /**
   * Constructor
   */
  AboutDialog(wxWindow *parent);

  /**
```

```
   * Destructor
   */
  virtual ~AboutDialog() { }

  /**
   * Sets the text on the dialog
   */
  void SetText(const wxString& text);

private:

  wxStaticText *m_pInfoText;
  wxButton *m_pOkButton;

};

#endif
```

Example 28. AboutDialog.cpp – The AboutDialog implementation

```
// For compilers that supports precompilation , includes "wx/wx.h"
#include "wx/wxprec.h"

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

#include "AboutDialog.h"

AboutDialog:: AboutDialog(wxWindow *parent)
    : wxDialog(parent, -1, "About Simple Text Editor", wxDefaultPosition,
               wxSize(200, 200), wxDEFAULT_DIALOG_STYLE)
{
  m_pInfoText = new wxStaticText(this, -1, "", wxPoint(5, 5),
                                 wxSize(100, 100), wxALIGN_CENTRE);
  m_pOkButton = new wxButton(this, wxID_OK, "Ok", wxPoint(5, 40));
}

void AboutDialog::SetText(const wxString& text)
{
  m_pInfoText->SetLabel(text);
}
```

You can see in the examples that there's no event handling. You don't need to provide it because you can use the default event handling of wxWindows. A dialog is automatically closed when the user clicks on a button that has the *wxID_OK* or *wxID_CANCEL* identifier. Example 29 shows you how you can show the dialog when the user clicks on the about menu item.

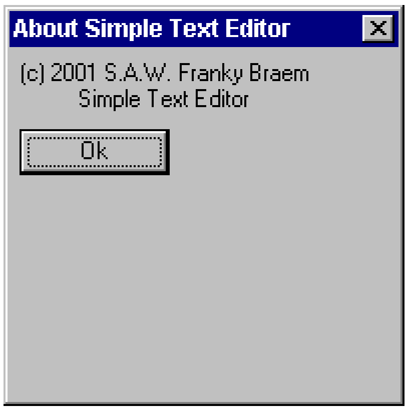
When you don't use the default handling, you don't use an Ok button for example, then you need to destroy the dialog yourself (by calling Destroy()) in the EVT_CLOSE event! Your application keeps running when you don't destroy the dialog.

Example 29. How to show a dialog?

```
void TextFrame::OnMenuInfoAbout(wxCommandEvent &event)
{
  AboutDialog *dlg = new AboutDialog(this);
  dlg->SetText("(c) 2001 S.A.W. Franky Braem\nSimple Text Editor\n");
  dlg->ShowModal();
}
```

When you execute the text editor and select the Info/About menu, you must see a dialog as shown in Figure 1.

Figure 1. The about dialog.



(c) 2001 S.A.W. Franky Braem
Simple Text Editor

6.2 Layout Dialogs

You're probably not happy with the layout of the dialog in Figure 1. It would be nice to center the text and the button on the dialog. Moreover, what happens when the size of the text changes? The button should always be positioned below the text.

6.2.1 wxLayoutConstraints

Instead of calculating the positions of the controls when you get an EVT_SIZE event, wxWindows helps you with the *wxLayoutConstraints* class to layout the controls on a parent window. Each window can be associated with a *wxLayoutConstraints* object to define the way it is laid out, with respect to its sibling or the parent.

wxLayoutConstraints contains 8 different constraints:

left	Represents the left hand edge of the window.
right	Represents the right hand edge of the window.
top	Represents the top hand edge of the window.
bottom	Represents the bottom edge of the window.
width	Represents the width of the window.
height	Represents the height of the window.
centreX	Represents the horizontal center point of the window.
centreY	Represents the vertical center point of the window.

The constraints are initially set to *wxUnconstrained*, which means that their values should be calculated by looking at known constraints. The layout algorithm needs to know exactly 4 constraints, to calculate the position and size of the control.

Therefore, you should always set exactly 4 constraints of the above table.

The algorithm is executed when the *Layout* method of wxWindow is called. You can use the *SetAutoLayout* method to automatically call *Layout* when the window, frame, dialog or panel is resized. Alternatively, you can call *Layout* yourself.

Each constraint is an object of the wxIndividualLayoutConstraint class. Each of the following methods defines how the window is constrained relative to either its siblings or its parent.

Above(wxWindow *w, int margin = 0)
Constrains the edge to be above of the given window, with an optional margin.

Absolute(int value)
Constrains the edge to be the given absolute value.

AsIs()
Sets the edge or constraint to the current value of the window's value. If either the width or the height constraints are as is, the window is not resized but moved. This is often used when the size of a control is not allowed to change (e.g. a button)

Below(wxWindow *w, int margin = 0)
Constrains the edge to be below the given window, with an optional margin.

LeftOf(wxWindow *w, int margin = 0)
Constrains the edge to be left of the given window, with an optional margin.

PercentOf(wxWindow *w, wxEdge edge, int per)
Constrains the edge or dimension to be a percentage of the given window and his edge.

RightOf(wxWindow *w, int margin = 0)
Constrains the edge to be right of the given window, with an optional margin.

SameAs(wxWindow *w, wxEdge edge, int margin = 0)
Constrains the edge to be the same as the edge of the window, with an optional margin.

Example 30 AboutDialog.cpp - Using wxLayoutConstraints.

```
// For compilers that supports precompilation , includes "wx/wx.h"
#include "wx/wxprec.h"

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

#include "AboutDialog.h"

AboutDialog:: AboutDialog(wxWindow *parent)
    : wxDialog(parent, -1, "About Simple Text Editor", wxDefaultPosition,
               wxSize(200, 200), wxDEFAULT_DIALOG_STYLE)
{
  SetAutoLayout(TRUE);

  wxLayoutConstraints *layout = new wxLayoutConstraints();

  layout->top.SameAs(this, wxTop, 10);
  layout->centreX.SameAs(this, wxCentreX);
  layout->width.AsIs();
  layout->height.AsIs();
  m_pInfoText = new wxStaticText(this, -1, "", wxPoint(-1, -1),
                                 wxDefaultSize, wxALIGN_CENTER);
  m_pInfoText->SetConstraints(layout);

  layout = new wxLayoutConstraints();
  layout->top.Below(m_pInfoText, 10);
  layout->centreX.SameAs(this, wxCentreX);
  layout->width.PercentOf(this, wxWidth, 80);
  layout->height.AsIs();

  m_pOkButton = new wxButton(this, wxID_OK, "Ok", wxPoint(-1, -1));
  m_pOkButton->SetConstraints(layout);

  Layout();
}

void AboutDialog::SetText(const wxString& text)
{
  m_pInfoText->SetLabel(text);
 // Call layout, because the static text could be resized.
  Layout();
}
```

Example 30 is the new implementation of the AboutDialog.
m_pInfoText is layout as follows:

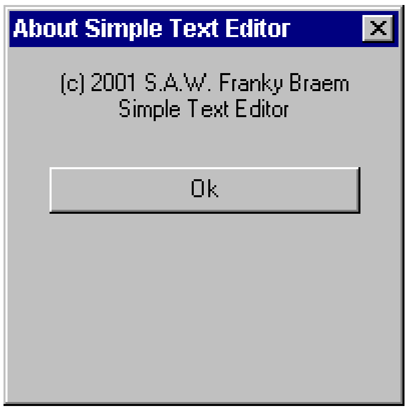
layout->top.SameAs(this, wxTop, 10);
The top is the same as the top of its parent with a margin of 10.
layout->centreX.SameAs(this, wxCentreX);
The horizontal center is the same as the horizontal center of its parent.
layout->width.AsIs();
The width is kept unchanged. The width could be changed by setting a new text.
layout->height.AsIs();
The height is kept unchanged. The height could be changed by setting a new text.

m_pOkButton is layout as follows:

layout->top.Below(m_pInfoText, 10);
The top of the button is below m_pInfoText with a margin of 10.
layout->centreX.SameAs(this, wxCentreX);
The horizontal center of the button is the same as the center of its parent.
layout->width.PercentOf(this, wxWidth, 80);
The width is 80% of the width of its parent.
layout->height.AsIs();
The height is kept unchanged.

As you can see in Figure 2 the layout of the dialog is much better now.

Figure 2 The About dialog with wxLayoutConstraint.



(c) 2001 S.A.W. Franky Braem
Simple Text Editor

6.2.2 wxSizer

The preferred way of wxWindows to layout controls on your dialogs are sizers. Sizers are classes derived from wxSizer. The most important method of wxSizer is Add. With this method, you add controls, other sizers or spacers as items to a sizer.

Adding controls

```
void Add(wxWindow* window, int option = 0,int flag = 0,
         int border = 0, wxObject* userData = NULL)
```

- **window** is the control to add to the sizer.
- **option** is used together with wxBoxSizer
  0 means that the size of the control is not allowed to change in the main orientation of the sizer.
  1 means that the size of the control may grow or shrink in the main orientation of the sizer.
- **flag** can be used to set a number of flags which can be combined with the OR operator.

Border flags:

wxTOP	Defines that the border is on the top.
wxBOTTOM	Defines that the border is on the bottom.
wxLEFT	Defines that the border is on the left.
wxRIGHT	Defines that the border is on the right.
wxALL	Defines that the border is on all sides.

Behavior flags:

wxGROW or wxEXPAND	The item may be resized.
wxSHAPED	The item may be proportionally resized.
wxALIGN_CENTER or wxALIGN_CENTRE	The item is centered.
wxALIGN_LEFT	The item is left aligned.
wxALIGN_TOP	The item is top aligned.
wxALIGN_RIGHT	The item is right aligned.
wxALIGN_CENTER_HORIZONTAL	The item is centered in the main orientation.
wxALIGN_CENTER_VERTICAL	The item is centered in the main orientation.

- **border** defines the border width when a border flag is set.
- **userData** allows an extra object to be attached to the sizer item, which can be used by derived classes.

Adding sizers

void Add(wxSizer* *sizer*, int *option* = *0*, int *flag* = *0*, int *border* = *0*, wxObject* *userData* = *NULL*)

Sizers can be nested. See adding controls for a detailed explanation of all the parameters.

Adding spacers

void Add(int width, int height, int option = 0, int flag = 0, int border = 0, wxObject* userData = NULL)

A spacer can be used to keep space between controls. You could for example add a spacer between two buttons on a dialog. When the spacer is allowed to grow, the result will be that the first button is kept left aligned and the other one is kept right aligned.

- wxBoxSizer

  A box sizer is used to layout your controls in a row/column hierarchy. A box sizer can grow in both directions. Depending on the flags you specify, a box sizer will grow vertically and/or horizontally. When you create a box sizer, you have to give it a main orientation.

  First, a sizer is created for the full dialog. Its main orientation is vertical. All the other sizers, who are children of the dialog sizer, will have a horizontal main orientation. They are allowed to grow horizontally, but not vertically. The second sizer is used to layout the Ok button. This is done to keep a fixed margin between the button and the edge of the dialog. When the button is added, a border of size 20 is specified.

  Example 31. Using the wxBoxSizer

```
AboutDialog:: AboutDialog(wxWindow *parent)
    : wxDialog(parent, -1, "About Simple Text Editor", wxDefaultPosition,
               wxSize(200, 200), wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER)
{
  wxBoxSizer *dialogSizer = new wxBoxSizer(wxVERTICAL);

  wxBoxSizer *textSizer = new wxBoxSizer(wxHORIZONTAL);
  m_pInfoText = new wxStaticText(this, -1, "", wxDefaultPosition,
                                 wxDefaultSize, wxALIGN_CENTER);
  // The text can grow horizontally.
  textSizer->Add(m_pInfoText, 1, 0);

  wxBoxSizer *buttonSizer = new wxBoxSizer(wxHORIZONTAL);
  m_pOkButton = new wxButton(this, wxID_OK, "Ok", wxPoint(-1, -1));
  // The button can grow horizontally, not vertically.
  // The button has a left and right border with size 20
  buttonSizer->Add(m_pOkButton, 1, wxLEFT | wxRIGHT, 20);

  // The sizers can't change their vertical size. They can only grow horizontally.
  dialogSizer->Add(textSizer, 0, wxGROW);
  dialogSizer->Add(buttonSizer, 0, wxGROW);

  SetSizer(dialogSizer);
  SetAutoLayout(TRUE);
  Layout();
}
```

- wxGridSizer

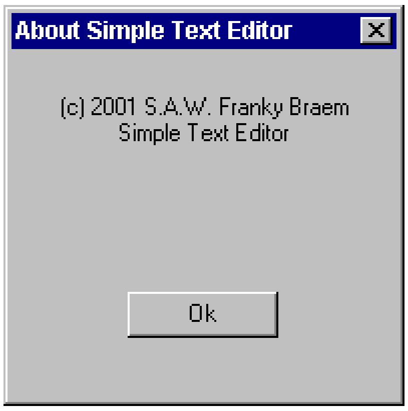
A grid sizer is a sizer which layouts the controls in a two-dimensional table that contains cells of the same size. The width of each cell is the width of the widest control and the height of each cell is the height of the tallest control. When you create a grid sizer you have to specify how many rows you want and how many columns. If one of these are zero the value is calculated by the grid sizer. You can also specify how much space there must be between columns and rows.

In Example 32 a grid sizer is created with two rows and one column. The space between columns and rows is 10.

Example 32. AboutDialog using wxGridSizer

```
AboutDialog:: AboutDialog(wxWindow *parent)
    : wxDialog(parent, -1, "About Simple Text Editor", wxDefaultPosition,
               wxSize(200, 200), wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER)
{
  wxGridSizer *dialogSizer = new wxGridSizer(2, 1, 10, 10);

  m_pInfoText = new wxStaticText(this, -1, "", wxDefaultPosition,
                                 wxDefaultSize, wxALIGN_CENTER);
  dialogSizer->Add(m_pInfoText, 0, wxALIGN_CENTER);

  m_pOkButton = new wxButton(this, wxID_OK, "Ok", wxPoint(-1, -1));
  dialogSizer->Add(m_pOkButton, 0, wxALIGN_CENTER);

  SetSizer(dialogSizer);
  SetAutoLayout(TRUE);
  Layout();
}
```

Figure 3. The about box using wxGridSizer



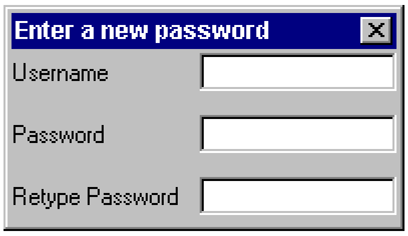
(c) 2001 S.A.W. Franky Braem
Simple Text Editor

- wxFlexGridSizer

  A flex grid sizer is a sizer which layouts the controls in a two-dimensional table with all cells in one row with the same width and all cells in one column with the same height.

  A grid sizer is very useful for dialogs where you need to attach labels to controls. Figure 4 shows an example of the use of wxFlexGridSizer. The width of the first column, containing the labels, is the maximum width of one of the labels. The code for creating such a dialog is shown in Example 33. You can see that 2 rows and 2 columns are created. When adding controls to the sizer, they are placed from left to right and from top to bottom.

Figure 4. An example of wxFlexGrid



Example 33. Using wxFlexGridSizer

```
wxFlexGridSizer *dialogSizer = new wxFlexGridSizer(2, 2, 10, 10);

dialogSizer->Add(new wxStaticText(this, -1, "Username"), 0, wxALIGN_CENTRE_VERTICAL);
dialogSizer->Add(new wxTextCtrl(this, ID_USER_NAME), 0, wxALIGN_CENTRE_VERTICAL);
dialogSizer->Add(new wxStaticText(this, -1, "Password"), 0, wxALIGN_CENTRE_VERTICAL);
dialogSizer->Add(new wxTextCtrl(this, ID_PASSWORD), 0, wxALIGN_CENTRE_VERTICAL);
dialogSizer->Add(new wxStaticText(this, -1, "Retype Password"), 0,
                                  wxALIGN_CENTRE_VERTICAL);
dialogSizer->Add(new wxTextCtrl(this, ID_PASSWORD2), 0, wxALIGN_CENTRE_VERTICAL);

SetSizer(dialogSizer);
SetAutoLayout(TRUE);
Layout();
```

- wxStaticBoxSizer
  wxStaticBoxSizer is a sizer derived from wxBoxSizer. It adds a static box around the sizer.
- wxNotebookSizer
  This sizer is a specialized sizer to make sizers work on wxNotebook controls.

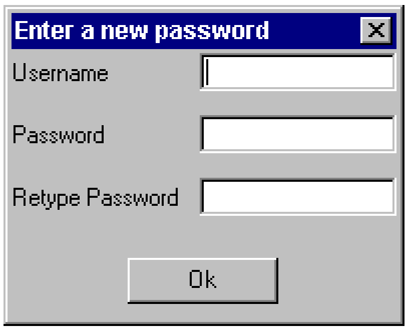
Sizers can be nested. You can use a box sizer together with a grid sizer. Example 34 shows you how a wxBoxsizer is combined with a wxFlexGridSizer.

Example 34. Nested sizers.

```
  wxBoxSizer *dialogSizer = new wxBoxSizer(wxVERTICAL);

  wxFlexGridSizer *controlSizer = new wxFlexGridSizer(2, 2, 10, 10);

  controlSizer->Add(new wxStaticText(this, -1, "Username"), 0, wxALIGN_CENTRE_VERTICAL);
  controlSizer->Add(new wxTextCtrl(this, -1), 0, wxALIGN_CENTRE_VERTICAL);
  controlSizer->Add(new wxStaticText(this, -1, "Password"), 0, wxALIGN_CENTRE_VERTICAL);
  controlSizer->Add(new wxTextCtrl(this, -1), 0, wxALIGN_CENTRE_VERTICAL);
  controlSizer->Add(new wxStaticText(this, -1, "Retype Password"), 0,
                                   wxALIGN_CENTRE_VERTICAL);
  controlSizer->Add(new wxTextCtrl(this, -1), 0, wxALIGN_CENTRE_VERTICAL);

  wxBoxSizer *buttonSizer = new wxBoxSizer(wxVERTICAL);
  buttonSizer->Add(new wxButton(this, wxID_OK, "Ok"));

  dialogSizer->Add(controlSizer);
  dialogSizer->Add(20, 20);
  dialogSizer->Add(buttonSizer, 0, wxALIGN_CENTRE);

  SetSizer(dialogSizer);
  SetAutoLayout(TRUE);
  Layout();
```

The result of this code is shown in Figure 5.

Figure 5. A dialog with combined sizers.



6.3 wxPanel

A panel is a window that can contain controls. Its main purpose is to be similar as a dialog, with the exception that it can have any window as a parent. This way you can create a frame that contains several controls.

```
wxPanel(wxWindow *parent, wxWindowID id = -1, const wxPoint& pos = wxDefaultPosition,
        const wxSize& size = wxDefaultSize, long style = wxTAB_TRAVERSAL | wxNO_BORDER,
        const wxString& name = wxPanelNameStr)
```

- **parent** is the parent window of the panel.
- **id** is the unique identifier for the panel. Use –1 when you don’t need it.
- **pos** is the position of the panel.
- **size** is the size of the panel.
- **style** is the style of the panel. wxPanel doesn’t define any specific styles.

6.3.1 Methods

6.3.2 Event handling

6.3.3 Example

6.4 Dialog Resources

wxWindows provides a resource file facility, which allows the separation of dialog, menu, bitmap, and icon specifications from the application code, but there are some problems with it:

- The resource file format is a recent addition and subject to change. In later versions, the format will be replaced by XML.
- Not all controls are available.

7 Standard Controls

In the previous chapters, you already saw some standard controls. This chapter will show you how to use the standard controls and what you can do with them. A standard control always needs a parent window: a wxFrame, wxPanel or wxDialog.

A standard control should be created on the heap (using new) and added as a child to another window. The parent window will delete your control. So, don't delete it yourself, because this will result in access violations errors.

When using an identifier for a window you have to be sure that you use a valid number. Don't use zero because this can cause segmentation faults. The easiest way is to use an enumeration with the first element starting at 1000. When you don't need an identifier use -1.

Validators are not discussed in this chapter. Read chapter 8 to know how to use them.

7.1 wxButton

By clicking on a button, the user initiates a command.

The constructor looks like:

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

- **parent** is the parent of the button. Can't be NULL!
- **id** is the identifier of the button. Use –1 when you don't need an identifier.
- **label** is the text that is shown on the button.
- **pos** is the position of the button on its parent. When you use sizers you can use wxDefaultPosition.
- **size** is the size of the button.
- **style** is the style of the button. Besides the styles of wxWindow you can also use the following specific button styles:

wxBU_LEFT	Left-justifies the bitmap. (Win32 only)
wxBU_TOP	Aligns the bitmap to the top of the button. (Win32 only)
wxBU_RIGHT	Right-justifies the bitmap. (Win32 only)
wxBU_BOTTOM	Aligns the bitmap to the bottom of the button. (Win32 only)

- **validator** is a validator for the button.

7.1.1 Methods

wxString GetLabel() const
Returns the string label of the button.

wxSize GetDefaultSize()
Returns the default size for a button.

void SetDefault()
This sets the button as the default button for the panel or dialog.

void SetLabel(const wxString &label)
Sets a new string label for the button.

7.1.2 Event handling

How do you execute an action when the user clicks on a button? You use the *EVT_BUTTON* macro. The argument of the function is a *wxCommandEvent*. To attach a class method to your button, you have to use a unique id for the button. The easiest way to create a unique id is using an enumeration. For some buttons, you can use predefined identifiers. For example: use *wxID_OK* for an "ok" button on a dialog and wxWindows is able to do a default event handling for you.

EVT_BUTTON(id, func)	Fired when the user clicks the button.

7.1.3 Example

The following example shows a simple dialog with a centered button. When you click the button, a message box pops up with the message "You clicked my button".

Example 35. ButtonDlg.h - The definition of the wxButton example.

```
#ifndef _ButtonDlg_H
#define _ButtonDlg_H

class ButtonDlg : public wxDialog
{
public:
  ButtonDlg();

protected:

  void OnButtonSelect(wxCommandEvent &event);

private:
  DECLARE_EVENT_TABLE()

  // ID's
  enum
  {
    BUTTON_SELECT = 1000
  };

};

#endif
```

Example 36. ButtonDlg.cpp – The implementation of the wxButton example.

```
#include "wx/wx.h"

#include "ButtonDlg.h"

ButtonDlg::ButtonDlg() : wxDialog(NULL, -1, "ButtonDialog",
                                  wxDefaultPosition, wxSize(150, 150))
{
  wxButton *button = new wxButton(this, BUTTON_SELECT, "Select", wxDefaultPosition);

  // Setting the button in the middle of the dialog.
  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *buttonSizer = new wxBoxSizer(wxVERTICAL);
  buttonSizer->Add(button, 0, wxALIGN_CENTER);
  dlgSizer->Add(buttonSizer, 1, wxALIGN_CENTER);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();
}

void ButtonDlg::OnButtonSelect(wxCommandEvent& command)
{
  wxMessageBox("You clicked my button");
}

BEGIN_EVENT_TABLE(ButtonDlg, wxDialog)
```

```
  EVT_BUTTON(BUTTON_SELECT, ButtonDlg::OnButtonSelect)
END_EVENT_TABLE()
```

7.2 wxBitmapButton

A wxBitmapButton is a button that contains a bitmap. You can use it on a dialog, panel or almost any window.

The constructor:

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

- **parent** is the parent of this button. Can't be NULL!
- **id** is the unique identifier for the button. Use –1 when you don't need this.
- **bitmap** is the bitmap to display on the button.
- **pos** is the position of the button. When not set, *wxDefaultPosition* is used.
- **size** is the size of the button. When not set, *wxDefaultSize* is used.
- **style** is the style of the button. Besides the styles of *wxWindow*, the following styles can be used:

wxBU_AUTODRAW	The button will be drawn automatically using the label bitmap. When no style is specified, this will be used as default. (Win32 only)
wxBU_LEFT	Left-justifies the bitmap. (Win32 only)
wxBU_TOP	Aligns the bitmap to the top of the button. (Win32 only)
wxBU_RIGHT	Right-justifies the bitmap. (Win32 only)
wxBU_BOTTOM	Aligns the bitmap to the bottom of the button. (Win32 only)

- **validator** is a validator for the button.

7.2.1 Methods

wxBitmap& GetBitmapDisabled() const;
Returns the bitmap for the disabled button.

wxBitmap& GetBitmapFocus() const;
Returns the bitmap for the focused state

wxBitmap& GetBitmapLabel() const;
Returns the bitmap label (the one you passed in the constructor).

wxBitmap& GetBitmapSelected() const;
Returns the bitmap for the selected state.

void SetBitmapDisabled(const wxBitmap& disabled);
Sets the bitmap for the disabled state.

void SetBitmapFocus(const wxBitmap& focus);
Sets the bitmap for the focused state.

void SetLabel(const wxBitmap& bitmap);
void SetBitmapLabel(const wxBitmap& bitmap);
Sets the bitmap for the button.

void SetBitmapSelected(const wxBitmap& sel);
Sets the bitmap for the selected state.

7.2.2 Event handling

The events for wxBitmapButton are the same as *wxButton*.

7.2.3 Example

The example will show you how to create a *wxBitmapButton*. When you want more details on how to use bitmaps, you should read chapter 18.

Example 37. ButtonDlg.h - The definition of the wxBitmapButton example.

```
#ifndef _ButtonDlg_H
#define _ButtonDlg_H

class ButtonDlg : public wxDialog
{
public:
  ButtonDlg();

protected:

  void OnButtonSelect(wxCommandEvent &event);
  void OnClose(wxCloseEvent &event);
  DECLARE_EVENT_TABLE()

private:
  // ID's
  enum
  {
    ID_BUTTON_SELECT = 1000
  };

  wxBitmapButton *m_pButton;
};

#endif
```

Example 38. ButtonDlg.cpp - The implementation of the wxBitmapButton example.

```
#include "wx/wx.h"

#include "ButtonDlg.h"

ButtonDlg::ButtonDlg() : wxDialog(NULL, -1, "ButtonDialog", wxDefaultPosition, wxSize(150,
150))
{
  m_pButton = new wxBitmapButton(this, ID_BUTTON_SELECT, wxBitmap("button"));

  // Setting the button in the middle of the dialog.
  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *buttonSizer = new wxBoxSizer(wxVERTICAL);
  buttonSizer->Add(m_pButton, 0, wxALIGN_CENTER);
  dlgSizer->Add(buttonSizer, 1, wxALIGN_CENTER);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();

}

void ButtonDlg::OnButtonSelect(wxCommandEvent& command)
{
  wxMessageBox("You clicked my button");
}

// We have to destroy this dialog, because we don't use a default Ok button!!!
void ButtonDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}
```

```
BEGIN_EVENT_TABLE(ButtonDlg, wxDialog)
  EVT_BUTTON(ID_BUTTON_SELECT, ButtonDlg::OnButtonSelect)
  EVT_CLOSE(ButtonDlg::OnClose)
END_EVENT_TABLE()
```

In this example, you have to add the bitmap definition in the resource file. Example 39 shows you how to do that.

Example 39. ButtonApp.rc - The resource file of the wxBitmapButton example.

```
button BITMAP "books.bmp"

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

7.3 wxCalendarCtrl

The calendar control allows the user to select a date. You can customize the display of the control in several ways: colors and fonts can be globally changed and each day can have another display style. Read chapter 20 Date & Time on page101 when you need more information about handling dates in wxWindows.

You have to explicitly include "wx/calctrl.h" before you can use the calendar control.

The constructor looks like this:

```
wxCalendarCtrl(wxWindow* parent, wxWindowID id, const wxDateTime& date = wxDefaultDateTime,
               const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,
               long style = wxCAL_SHOW_HOLIDAYS, const wxString& name = wxCalendarNameStr)
```

- **parent** is the parent window of the calendar control. This can't be NULL!
- **id** is the unique identifier for this control. You can use –1 when you don't need one.
- **date** is the default date.
- **pos** is the position of the control.
- **size** is the size of the control.
- **style** is the style of the control.

wxCAL_SUNDAY_FIRST	Shows Sunday as the first day of the week.
wxCAL_MONDAY_FIRST	Shows Monday as the first day of the week.
wxCAL_SHOW_HOLIDAYS	Highlights the holidays in the calendar. (Default)
wxCAL_NO_YEAR_CHANGE	Disable the year selection.
wxCAL_NO_MONTH_CHANGE	Disable the month selection.

The display of each day can be changed using the wxCalendarDateAttr class.

7.3.1 Methods

void EnableHolidayDisplay(bool display = TRUE)
When enabled the special highlighting of holidays is used. Use this instead of changing the wxCAL_SHOW_HOLIDAYS style bit. The style can only be set on creation.

void EnableMonthChange(bool enable = TRUE)
When disabled the user can't select another month and year. Use this instead of changing the wxCAL_NO_MONTH_CHANGE style bit. The style can only be set on creation.

void EnableYearChange(bool enable = TRUE)
When disabled the user can't select another year. Use this instead of changing the wxCAL_NO_YEAR_CHANGE style bit. The style can only be set on creation.

wxCalendarDateAttr* GetAttr(size_t day) const
The attributes for the given day are returned. This can be NULL.

const wxDateTime& GetDate() const
The selected date is returned.

const wxColour& GetHeaderColourBg() const
The background color from the weekdays header is returned.

const wxColour& GetHeaderColourFg() const
The foreground color from the weekdays header is returned.

const wxColour& GetHighlightColourBg() const
Returns the background color used for highlighting the selected date.

const wxColour& GetHighlightColourFg() const
Returns the foreground color used for highlighting the selected date.

const wxColour& GetHolidayColourFg() const
Returns the foreground color used for highlighting holidays.

const wxColour& GetHolidayColourBg() const
Returns the background color used for highlighting holidays.

void ResetAttr(size_t day)
Reset the attributes for the given day.

void SetAttr(size_t day, wxCalendarDateAttr *attr)
Set the attributes for the given day. You can use NULL to reset all attributes for that day.

void SetDate(const wxDateTime& date)
Set the current date.

void SetHeaderColours(const wxColour &colFg, const wxColour &colBg)
Set the colors used for the weekday header. colFg is the foreground color and colBg is the background color.

void SetHighlightColours(const wxColour& colFg, const wxColour& colBg)
Set the foreground color (colFg) and the background color (colBg) for highlighting the selected date.

void SetHoliday(size_t day)
Mark the given day as a holiday.

void SetHolidayColours(const wxColour& colFg, const wxColour& colBg)
Set the foreground color (colFg) and background color (colBg) used for highlighting holidays. This is only used when the style wxCAL_SHOW_HOLIDAYS is set.

7.3.2 wxCalendarDateAttr

wxCalendarDateAttr is used for setting attributes for a specific day.

Constructors:

```
wxCalendarDateAttr()

wxCalendarDateAttr(const wxColour& colText, const wxColour& colBack = wxNullColour,
                   const wxColour& colBorder = wxNullColour, const wxFont& font = wxNullFont,
                   wxCalendarDateBorder border = wxCAL_BORDER_NONE)
```

`wxCalendarDateAttr(wxCalendarDateBorder` *`border`*`, const wxColour&` *`colBorder`* `=` *`wxNullColour`*`)`

- **colText** is the foreground color of the day number.
- **colBack** is the background color of the day number.
- **colBorder** is the color of the border.
- **font** is the font used for the day number.
- **border** specifies which border is drawn. *wxCalendarDateBorder* is an enumeration with the following values:

wxCAL_BORDER_NONE	No border.
wxCAL_BORDER_SQUARE	A square border.
wxCAL_BORDER_ROUND	A round border.

Methods:

const wxColour& GetBackgroundColour() const
Returns the background color to use for the day with this attribute.

wxCalendarDateBorder GetBorder() const
Returns the border type to use for the day with this attribute.

const wxColour& GetBorderColour() const
Returns the border color to use for the day with this attribute.

const wxFont& GetFont() const
Returns the font to use for the day with this attribute.

bool HasBackgroundColour() const
Returns true when the attribute has a non-default background color.

bool HasBorder() const
Returns true when the attribute has a border.

bool HasBorderColour() const
Returns true when the attribute has a non-default border color.

bool HasFont() const
Returns true when the attribute has a non-default font.

bool HasTextColour() const
Returns true when the attribute has a non-default text foreground color.

bool IsHoliday() const
Returns true when the attribute specifies that the day should be displayed as a holiday.

void SetBackgroundColour(const wxColour& colBack)
Set the background color of the day number.

void SetBorder(wxCalendareDateBorder border)
Set the kind of border. Look to the constructors for the possible value.

void SetBorderColour(const wxColour& col)
Set the color of the border.

void SetFont(const wxFont& font)
Set the font to use.

void SetHoliday(bool holiday)
Display the date with this attribute as a holiday.

void SetTextColour(const wxColour& colText)
Set the foreground color of the day number.

7.3.3 Event handling

wxCalendarCtrl has several event handler macros. The handler function takes a *wxCalendarEvent*.

EVT_CALENDAR(id, func)	A day was double clicked or the user pressed the enter key.
EVT_CALENDAR_SEL_CHANGED(id, func)	The selected date is changed.
EVT_CALENDAR_DAY(id, func)	The selected day is changed.
EVT_CALENDAR_MONTH(id, func)	The selected month is changed.
EVT_CALENDAR_YEAR(id, func)	The selected year is changed.
EVT_CALENDAR_WEEKDAY_CLICKED	User clicked on the weekday header.

A *wxCalendarEvent* has the following methods:

const wxDateTime& GetDate() const
The selected date is returned. This method can be used for all event types except EVT_CALENDAR_WEEKDAY_CLICKED.

wxDateTime::WeekDay GetWeekDay() const
The selected weekday is returned. This method is only useful when handling the EVT_CALENDAR_WEEKDAY_CLICKED event.

7.3.4 Example

The example shows a dialog with a calendar control. When you double click or press the enter key on a day, the day is marked with a red background color.

Example 40. CalendarDlg.h - The definition of the Calendar example.

```
#ifndef _CalendarDlg_H
#define _CalendarDlg_H

#include <wx/calctrl.h>

class CalendarDlg : public wxDialog
{
public:
  CalendarDlg();

protected:
  void OnClose(wxCloseEvent &event);
  void OnCalendar(wxCalendarEvent &event);
  DECLARE_EVENT_TABLE()

private:
  enum
  {
    ID_CALENDAR = 1000
  };

  wxCalendarCtrl *m_pCalendar;
};

#endif // _CalendarDlg_H
```

Example 41. CalendarDlg.cpp - The implementation of the calendar example.

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

#include "CalendarDlg.h"
```

```
CalendarDlg::CalendarDlg() : wxDialog(NULL, -1, "CalendarDialog", wxDefaultPosition,
wxSize(185, 185))
{
  m_pCalendar = new wxCalendarCtrl(this, ID_CALENDAR, wxDefaultDateTime,
                                   wxDefaultPosition, wxDefaultSize,
                                   wxCAL_MONDAY_FIRST |
                                   wxCAL_SHOW_HOLIDAYS |
                                   wxRAISED_BORDER);

  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *CalendarSizer = new wxBoxSizer(wxVERTICAL);
  CalendarSizer->Add(m_pCalendar, 1, wxGROW);
  dlgSizer->Add(CalendarSizer, 1, wxGROW);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();

}

void CalendarDlg::OnCalendar(wxCalendarEvent &event)
{
  int day = event.GetDate().GetDay();
  bool on = false;
  wxCalendarDateAttr *attr = m_pCalendar->GetAttr(day);
  if ( attr != NULL )
  {
    if ( attr->GetBackgroundColour() == *wxRED )
       on = true;
  }
  wxCalendarDateAttr *newAttr = new wxCalendarDateAttr(*wxBLACK, on ? *wxWHITE : *wxRED);
  m_pCalendar->SetAttr(day, newAttr);
}

void CalendarDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(CalendarDlg, wxDialog)
  EVT_CALENDAR(ID_CALENDAR, CalendarDlg::OnCalendar)
  EVT_CLOSE(CalendarDlg::OnClose)
END_EVENT_TABLE()
```

7.4 wxCheckBox

A checkbox is a labeled box which can be checked or unchecked.

The constructor:

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

- **parent** is the parent window of the checkbox. This can't be NULL!
- **id** is the unique identifier of the checkbox. Use –1 when you don't need it.
- **label** is the text that will be shown next to the box.
- **pos** is the position of the checkbox.
- **size** is the size of the checkbox.
- **style** is the style of the checkbox. The checkbox doesn't have any special styles.
- **val** is the validator of the checkbox.

7.4.1 Methods

bool GetValue() const
Returns true when the checkbox is checked, false otherwise.

void SetLabel(const wxString& label)
Sets a new label for the checkbox.

bool SetValue(const bool state)
Check or uncheck the checkbox. This does not generate an event!

7.4.2 Event handling

Besides the events from wxWindow, the only event generated by the checkbox is when the checkbox is checked or unchecked. The *EVT_CHECKBOX* handler-macro is defined as:

EVT_CHECKBOX(id, func)

The function receives a *wxCommandEvent* argument.

7.4.3 Example

Example 42 and Example 43 shows you how to create a checkbox and how to use the *EVT_CHECKBOX* handler. With the *IsChecked* method, you can see whether the checkbox is checked or not. You can also use *GetValue* of wxCheckBox for this.

Example 42. CheckBoxDlg.h - The definition of the Checkbox example.

```
#ifndef _CheckBoxDlg_H
#define _CheckBoxDlg_H

class CheckBoxDlg : public wxDialog
{
public:
  CheckBoxDlg();

protected:
  void OnCheck(wxCommandEvent &event);
  void OnClose(wxCloseEvent &event);
  DECLARE_EVENT_TABLE()

private:
  enum
  {
    ID_CHECKBOX = 1000
  };

  wxCheckBox *m_pCheckBox;
};

#endif // _CheckBoxDlg_H
```

Example 43. CheckBoxDlg.cpp - The implementation of the Checkbox example.

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

#include "CheckBoxDlg.h"

CheckBoxDlg::CheckBoxDlg() : wxDialog(NULL, -1, "CheckBoxDialog",
                                      wxDefaultPosition, wxSize(150, 150))
{
  m_pCheckBox = new wxCheckBox(this, ID_CHECKBOX, "Check Me");

  // Set the checkBox in the middle of the dialog.
  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *checkBoxSizer = new wxBoxSizer(wxVERTICAL);
  checkBoxSizer->Add(m_pCheckBox, 0, wxALIGN_CENTER);
  dlgSizer->Add(checkBoxSizer, 1, wxALIGN_CENTER);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();

}

void CheckBoxDlg::OnCheck(wxCommandEvent& command)
{
    if ( command.IsChecked() )
      wxMessageBox("You checked me!");
    else
      wxMessageBox("You unchecked me!");
}

void CheckBoxDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(CheckBoxDlg, wxDialog)
  EVT_CHECKBOX(ID_CHECKBOX, CheckBoxDlg::OnCheck)
  EVT_CLOSE(CheckBoxDlg::OnClose)
END_EVENT_TABLE()
```

7.5 wxCheckListBox

wxCheckListBox is like a listbox, except the items can be checked or unchecked. This control is only available on the GTK and Windows port. When you want to use it on a Windows platform, wxWindows should be compiled with the *USE_OWNER_DRAWN* set to 1. When you didn't change any definitions, this should be ok.

The header file for *wxCheckListBox* is not automatically included with wx.h. You have to include "wx/checklst.h" yourself.

wxCheckListBox is derived from *wxListBox*. See 7.9 for more information on using *wxListBox*.

The constructor looks like this:

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

- **parent** is the parent window of *wxCheckListBox*. This can't be NULL!
- **id** is the unique identifier. Use –1 when you don't need an identifier.
- **pos** is the position of the window.
- **size** is the size of the window.
- **n** is the number of items in the choice list.
- **choices** is an array of wxString elements. Make sure it has *n* elements.
- **style** is the style of the window. You can use the styles of *wxListBox*.
- **validator** is the validator for this control.

7.5.1 Methods

void Check(int item, bool check = true)
Use this method to check or uncheck an item.

bool IsChecked(int item)
Returns true when the item is checked, false when unchecked.

7.5.2 Event handling

You can use all events of *wxListBox*. *wxCheckListBox* has one specific event for notifying you that an item in the list is checked or unchecked.

EVT_CHECKLISTBOX(id, func)

The argument for the handler function is a wxCommandEvent. You can use *GetSelection* to retrieve which item is checked or unchecked.

7.5.3 Example

The following example shows a checklistbox that contains three items. When an item is checked or unchecked a message shows which item is selected and whether it is checked or unchecked.

Example 44. CheckListBoxDlg.h - The definition of a checklistbox example.

```
#ifndef _CheckListBoxDlg_H
#define _CheckListBoxDlg_H

class CheckListBoxDlg : public wxDialog
{
public:
  CheckListBoxDlg();

protected:
  void OnCheck(wxCommandEvent &event);
  void OnClose(wxCloseEvent &event);
  DECLARE_EVENT_TABLE()

private:
  enum
  {
    ID_CHECKLISTBOX = 1000
  };

  wxCheckListBox *m_pCheckListBox;
};

#endif // _CheckListBoxDlg_H
```

Example 45. CheckListBox.cpp - The implementation of a checklistbox example.

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

#include "CheckListBoxDlg.h"

CheckListBoxDlg::CheckListBoxDlg() : wxDialog((wxWindow*) NULL, -1, "Test",
                                              wxDefaultPosition, wxSize(150, 150))
{
  wxString list[] = { "ABS", "Airbag", "Air conditioning" } ;
  // We can use wxDefaultPosition and wxDefaultSize,
  // because the sizers will take care of this.
  m_pCheckListBox = new wxCheckListBox(this, ID_CHECKLISTBOX,
                                       wxDefaultPosition, wxDefaultSize, 3, list);

  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *CheckListBoxSizer = new wxBoxSizer(wxVERTICAL);
  CheckListBoxSizer->Add(m_pCheckListBox, 1, wxGROW);
  dlgSizer->Add(CheckListBoxSizer, 1, wxGROW);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();
}

void CheckListBoxDlg::OnCheck(wxCommandEvent& command)
{
  int item = command.GetSelection();
  wxString msg;
  msg.Printf("You clicked on item %d. The item is now %s", item,
             m_pCheckListBox->IsChecked(item) ? "checked" : "unchecked");
  wxMessageBox(msg);
}

void CheckListBoxDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(CheckListBoxDlg, wxDialog)
  EVT_CHECKLISTBOX(ID_CHECKLISTBOX, CheckListBoxDlg::OnCheck)
  EVT_CLOSE(CheckListBoxDlg::OnClose)
END_EVENT_TABLE()
```

To see if the item is checked or not, you have to use the *IsChecked* method from *wxCheckListBox*. The *IsChecked* method from *wxCommandEvent* can only be used for checkboxes and menus.

7.6 wxChoice

wxChoice is used to select one of a list of strings. The user sees the list when he clicks on the control. The index of the list is zero-based.

The constructor of *wxChoice*:

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

- **parent** is the parent of the *wxChoice* control. This can't be NULL!
- **id** is the unique identifier. When you don't need this use -1.
- **pos** is the position of the control.
- **size** is the size of the control.
- **n** is the number of items in the choices argument
- **choices** is a wxString array with the items.
- **style** is the style of the control. There are no special styles for *wxChoice*.
- **validator** is a validator for this control.

7.6.1 Methods

void Append(const wxString& item)
void Append(const wxString& item, void *clientData)
Item is appended to the list. clientData can be used to associate other data with the item. This can be useful to associate an item with an index in an array.

void Clear()
The list is cleared. All items are removed.

void Delete(int n)
Deletes the item from the list. The index of the list is zero-based.

int FindString(const &wxString string) const
Finds a string in the list. When the string is not found, -1 is returned.

void* GetClientData(int n) const
Returns a pointer to the client data associated to the item. The index is zero-based. When the item is not found, NULL is returned.

int GetCount() const
Returns the number of items.

int GetSelection() const
Returns the index of the selected item or –1 when nothing is selected. The index is zero-based.

wxString GetString(int n) const
Returns the string at the given position. The position is zero-based. When the position is invalid an empty string is returned.

wxString GetStringSelection() const
Returns the selected string or an empty string when nothing is selected.

void SetClientData(int n, void *data)
Associates the data pointer with the given item.

void SetSelection(int n)
Sets the text field with the string at the given position. This does not generate an *EVT_CHOICE* event.

void SetString(int n, const wxString& string)
Replaces the item at the given position with the new string.

void SetStringSelection(const wxString &string)
Sets the choice with the string. This does not generate an EVT_CHOICE event.

7.6.2 Event handling

wxChoice has one specific event handler:

EVT_CHOICE(id, func)

The handler function will be called when an item is selected. The argument of the handler is a *wxCommandEvent*.

7.6.3 Example

The example shows a dialog with a *wxChoice* control. When the user selects an item, the item will be shown immediately in the text field. This is done with the *SetSelection* method. When you don't do this the selected item will only be shown when the user clicks on the arrow or when the control looses the focus.

Example 46. ChoiceDlg.h - The definition of the wxChoice example.

```
#ifndef _ChoiceDlg_H
#define _ChoiceDlg_H

class ChoiceDlg : public wxDialog
{
public:
  ChoiceDlg();

protected:
  void OnClose(wxCloseEvent &event);
  void OnChoice(wxCommandEvent &event);

  DECLARE_EVENT_TABLE()

private:
  enum
  {
    ID_CHOICE = 1000
  };

  wxChoice *m_pChoice;
};

#endif // _ChoiceDlg_H
```

Example 47. ChoiceDlg.cpp - The implementation of the wxChoice example.

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

#include "ChoiceDlg.h"

ChoiceDlg::ChoiceDlg() : wxDialog(NULL, -1, "ChoiceDialog",
                                  wxDefaultPosition, wxSize(185, 185))
{
  wxString list[] = { "ABS", "Airbag", "Air Conditioning" };
  m_pChoice = new wxChoice(this, ID_CHOICE, wxDefaultPosition, wxDefaultSize, 3, list);
^
  // An item can also be added with the Append method
  m_pChoice->Append("Automatic gear");

  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *ChoiceSizer = new wxBoxSizer(wxVERTICAL);
  ChoiceSizer->Add(m_pChoice, 1, wxGROW);
  dlgSizer->Add(ChoiceSizer, 1, wxGROW);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();

}

void ChoiceDlg::OnChoice(wxCommandEvent &event)
{
  // When an item is selected it is not automatically shown in the choice box
  // That's why it is set explicitly
  m_pChoice->SetSelection(event.GetSelection());
}

void ChoiceDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(ChoiceDlg, wxDialog)
  EVT_CHOICE(ID_CHOICE, ChoiceDlg::OnChoice)
  EVT_CLOSE(ChoiceDlg::OnClose)
END_EVENT_TABLE()
```

7.7 wxComboBox

A combobox is a combination of an edit control and a listbox. The user can select a string from a list or can enter a value when the text field is not read-only. The control can be displayed as a drop-down list with or without a text field. The index of the list is zero-based.

The constructor looks like this:

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

- **parent** is the parent of the combobox. This can't be NULL!
- **id** is the unique identifier for the combobox. –1 can be used when you don't need it.
- **value** is the initial value for the text field.
- **pos** is the position of the combobox.
- **size** is the size of the combobox.
- **n** is the number of strings in the choices argument.
- **choices** is an array of strings to initialize the list.
- **style** is the window style of the combobox. The specific combobox styles are listed below.

wxCB_SIMPLE	A combobox with a permanently list. (Windows only) Should be used together with wxCB_READONLY
wxCB_DROPDOWN	A combobox with a drop-down list.
wxCB_READONLY	A combobox with a read-only text field.
wxCB_SORT	The entries are sorted alphabetically.

- **validator** is a validator for the combobox.

7.7.1 Methods

wxComboBox is derived from *wxChoice*. Therefore, all public methods of *wxChoice* are also available for *wxComboBox*.

void Copy()
Copies the selected text to the clipboard.

void Cut()
Copies the selected text to the clipboard and removes the selection.

long GetInsertionPoint() const
Returns the insertion point of the text field.

long GetLastPosition() const
Returns the last position of the text field.

int GetSelection() const
Returns the position of the selected string in the text field or –1 when nothing is selected.

wxString GetValue() const
Returns the value of the text field.

void Replace(long from, long to, const wxString& value);
Replaces the text between the two positions with the new string.

void Remove(long from, long to);
Removes the text between the two positions from the text field.

void SetEditable(bool editable);
Enables or disables the text field.

void SetInsertionPoint(long pos);
Sets the insertion point of the text field at the given position.

void SetInsertionPointEnd();
Sets the insertion point of the text field at the end.

void SetSelection(long from, long to);
Selects the text between the given positions.

7.7.2 Event handling

You can react on an item selection with the *EVT_COMBOBOX* event. When the content of the text field changes, an *EVT_TEXT* event is fired.

```
EVT_COMBOBOX(id, func)
EVT_TEXT(id, func)
```

7.7.3 Example

The example will show a combobox where the user can select an item from the list, or can enter a new value.

Example 48. ComboboxDlg.h - The definition of the wxComboBox example.

```
#ifndef _ComboboxDlg_H
#define _ComboboxDlg_H

class ComboboxDlg : public wxDialog
{
public:
  ComboboxDlg();

protected:
  void OnClose(wxCloseEvent &event);
  void OnCombobox(wxCommandEvent &event);

  DECLARE_EVENT_TABLE()

private:
  enum
  {
    ID_COMBOBOX = 1000
  };

  wxComboBox *m_pCombobox;
};

#endif // _ComboboxDlg_H
```

Example 49. ComboboxDlg.cpp - The implementation of the wxComboBox example.

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

#include "ComboboxDlg.h"

ComboboxDlg::ComboboxDlg() : wxDialog(NULL, -1, "ComboboxDialog", wxDefaultPosition,
                                      wxSize(185, 185))
{
  wxString list[] = { "ABS", "Airbag", "Air Conditioning" };
  m_pCombobox = new wxComboBox(this, ID_COMBOBOX, "",
                               wxDefaultPosition, wxDefaultSize, 3, list);

  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *ComboboxSizer = new wxBoxSizer(wxVERTICAL);
  ComboboxSizer->Add(m_pCombobox, 1, wxGROW);
  dlgSizer->Add(ComboboxSizer, 1, wxGROW);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();

}

void ComboboxDlg::OnCombobox(wxCommandEvent &event)
{
  wxLogMessage("You've selected item: " + event.GetString());
}

void ComboboxDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(ComboboxDlg, wxDialog)
  EVT_COMBOBOX(ID_COMBOBOX, ComboboxDlg::OnCombobox)
  EVT_CLOSE(ComboboxDlg::OnClose)
END_EVENT_TABLE()
```

7.8 wxGauge

A gauge is a horizontal or vertical bar that shows a quantity. You can use a gauge to show the user the progress of a lengthy process. In the MSW distribution *wxGauge* is defined as a macro. When __WIN95__ is defined, the actual class that implements the gauge is called wxGauge95. When __WIN95__ is not defined, *wxGaugeMSW* is used. The difference is that *wxGauge95* uses the gauge control of the common controls library from Microsoft. Don't use these classes directly and use wxGauge instead. Otherwise, you break the portability of your code.

The constructor of *wxGauge*:

```
wxGauge(wxWindow *parent, wxWindowID id, int range, const wxPoint& pos = wxDefaultPosition,
        const wxSize& size = wxDefaultSize, long style = wxGA_HORIZONTAL,
        const wxValidator& validator = wxDefaultValidator, const wxString& name = “gauge”)
```

- **parent** is the parent of the gauge. This can't be NULL!

- **id** is the unique identifier of the gauge. Use –1 when you don't need it.
- **range** is the maximum value of the gauge.
- **pos** is the position of the gauge.
- **size** is the size of the gauge.
- **style** is the window style of the gauge.

wxGA_HORIZONTAL	A horizontal gauge. (default)
wxGA_VERTICAL	A vertical guage.
wxGA_PROGRESSBAR	On win95, creates a progress bar.
wxGA_SMOOTH	On win95, creates a smooth progress bar.

- **validator** is the validator of the gauge.

7.8.1 Methods

int GetBezelFace(void) const
Returns the width of the 3D bezel face. This method can only be used when __WIN95__ is not defined. It returns 0 when __WIN95__ is defined.

int GetRange(void) const
Returns the range

int GetShadowWidth(void) const
Returns the 3D shadow margin width. This method can only be used when __WIN95__ is not defined. It returns 0 when __WIN95__ is defined.

int GetValue(void) const
Returns the current value of the gauge.

void SetBezelFace(int w)
Sets the width of the 3D bezel face. This method can only be used when __WIN95__ is not defined. It doesn't do anything when __WIN95__ is defined.

void SetRange(int r)
Sets a new maximum range of the gauge.

void SetShadowWidth(int w)
Sets the width of the 3D shadow margin width. This method can only be used when __WIN95__ is not defined. It doesn't do anything when __WIN95__ is defined.

void SetValue(int pos)
Sets the current value of the guage.

7.8.2 Event handling

wxGauge is a control that doesn't react on user actions. Therefore, there's no event handling available.

7.8.3 Example

The example shows a gauge with a maximum range of 100. To show the progress in the gauge a timer is used. After each 100 milliseconds, the value of the gauge will be added by one.

Example 50. GaugeDlg.h - The definition of the gauge example.

```
#ifndef _GaugeDlg_H
#define _GaugeDlg_H

class GaugeDlg : public wxDialog
{
public:
  GaugeDlg();

protected:
  void OnTimer(wxTimerEvent &event);
  void OnClose(wxCloseEvent &event);

  DECLARE_EVENT_TABLE()

private:

  wxGauge *m_pGauge;
  wxTimer *m_pTimer;

  enum
  {
    ID_TIMER = 1000
  };
};

#endif // _GaugeDlg_H
```

Example 51. GaugeDlg.cpp - The implementation of the gauge example

```
#include <wx/wx.h>
#include "GaugeDlg.h"

GaugeDlg::GaugeDlg() : wxDialog(NULL, -1, "GaugeDialog", wxDefaultPosition, wxSize(185, 185))
{
  m_pGauge = new wxGauge(this, -1, 100, wxDefaultPosition, wxSize(150, 20), wxGA_SMOOTH);

  // This example uses a timer to show the progress in the gauge
  m_pTimer = new wxTimer(this, ID_TIMER);
  m_pTimer->Start(100);

  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *GaugeSizer = new wxBoxSizer(wxVERTICAL);
  GaugeSizer->Add(m_pGauge, 1, wxALIGN_CENTER);
  dlgSizer->Add(GaugeSizer, 1, wxALIGN_CENTER);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();

}

void GaugeDlg::OnTimer(wxTimerEvent &event)
{
  int value = m_pGauge->GetValue();
  if ( value == m_pGauge->GetRange() )
  {
    m_pTimer->Stop();
  }
  else
  {
    m_pGauge->SetValue(value + 1);
  }
}

void GaugeDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(GaugeDlg, wxDialog)
  EVT_TIMER(ID_TIMER, GaugeDlg::OnTimer)
  EVT_CLOSE(GaugeDlg::OnClose)
END_EVENT_TABLE()
```

7.9 wxListBox

A listbox is used to show the user a list where he can select one or more items. The index of an item is zero-based. The constructor looks like this:

```
wxListBox(wxWindow *parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition,
          const wxSize& size = wxDefaultSize, int n = 0, const wxString choices[] = NULL,
          long style = 0, const wxValidator& validator = wxDefaultValidator,
          const wxString& name = “listbox”);
```

- **parent** is the parent window of the listbox. This can’t be NULL!
- **id** is the unique identifier for the listbox. Use –1 when you don’t need it.
- **pos** is the position of the listbox.
- **size** is the size of the listbox.
- **n** is the number of items in the choices array.
- **choices** is an array of strings to show in the listbox.
- **style** is the window style of the listbox. Besides the common wxWindow styles, wxListbox has also its own styles.

wxLB_SINGLE	Single selection list.
wxLB_MULTIPLE	Multiple selection list.
wxLB_EXTENDED	Extended selection list. The user can select multiple items using the SHIFT key, the mouse or special keys.
wxLB_HSCROLL	Creates a horizontal scrollbar.
wxLB_ALWAYS_SB	Always shows a vertical scrollbar.
wxLB_NEEDED_SB	Only show a vertical scrollbar when needed.
wxLB_SORT	Items are sorted alphabetically. (Not on GTK)

- **validator** is the window validator of the listbox.

7.9.1 Methods

void Clear()
Removes all the items from the list.

void Delete(int n)
Deletes the item at the given position.

void Deselect(int n)
Deselectes the item at the given position.

int FindString(const wxString& s) const
Searches the item with the given string. Returns –1 when no item is found.

int GetCount() const
Returns the number of items in the list.

int GetSelection() const
Returns the position of the selected item. This method is only useful when the listbox doesn’t allow multiple selections.

int GetSelections(wxArrayInt& aSelections) const
Fills the array with all positions of the selected items. The array size is returned.

wxString GetString(int n) const
Returns the item at the given position.

void Insert(const wxString& item, int pos)
void Insert(const wxString& item, int pos, void *clientData)
void Insert(const wxString& item, int pos, wxClientData *clientData)
A new item is inserted before the given position. Position 0 means inserting at the beginning of the list. ClientData can be useful to associate the item with a pointer to your data.

void InsertItems(int nItems, const wxString *items, int pos)
void InsertItems(const wxArrayString& items, int pos)
Inserts the items before the given position. Don't forget to deallocate when you use a pointer to an array of wxString elements.

bool IsSelected(int n) const
Returns true when the item at the given position is selected.

void Set(int n, const wxString* items, void **clientData = NULL)
void Set(const wxArrayString& items, void **clientData = NULL)
The listbox is cleared and filled with the new items. When using a pointer to an array of wxString elements, don't forget to deallocate it.

void SetClientData(int n, void *data)
The item at the given position is associated with a pointer to the client data.

void SetFirstItem(int n)
void SetFirstItem(const wxString& s)
The item at the given position or with the given string will be the first visible item in the list. (Windows only)

void SetSelection(int n, bool select = TRUE)
Selects or deselects the item at the given position. This doesn't fire an EVT_COMMAND_LISTBOX_SELECT event.

void SetString(int n, const wxString& s)
Sets the item at the given position.

bool SetStringSelection(const wxString& s, bool select = TRUE)
The item with the given string is selected or deselected. This doesn't fire an EVT_COMMAND_LISTBOX_SELECT event.

7.9.2 Event handling

Events are fired when the user selects an item (EVT_COMMAND_LISTBOX_SELECTED) or when he double clicks on an item (EVT_COMMAND_LISTBOX_DOUBLE_CLICKED). The argument type for the handler function of both events is a wxCommandEvent.

```
EVT_LISTBOX(id, func)
EVT_LISTBOX_DCLICK(id, func)
```

7.9.3 Example

The example shows a listbox and will show a message box when the user double clicks on the listbox.

Example 52. ListboxDlg.h - The definition of the wxListBox example.

```
#ifndef _ListboxDlg_H
#define _ListboxDlg_H

class ListboxDlg : public wxDialog
{
public:
  ListboxDlg();

protected:
  void OnClose(wxCloseEvent &event);
  void OnListboxDClick(wxCommandEvent &event);
  DECLARE_EVENT_TABLE()

private:
  enum
  {
    ID_LISTBOX = 1000
  };

  wxListBox *m_pListbox;
};

#endif // _ListboxDlg_H
```

Example 53. ListboxDlg.cpp - The implementation of the wxListBox example.

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

#include "ListboxDlg.h"

ListboxDlg::ListboxDlg() : wxDialog(NULL, -1, "ListboxDialog", wxDefaultPosition,
                           wxSize(185, 185))
{
  wxString list[] = { "ABS", "Airbag", "Air Conditioning" };
  m_pListbox = new wxListBox(this, ID_LISTBOX, wxDefaultPosition, wxDefaultSize, 3, list);

  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *ListboxSizer = new wxBoxSizer(wxVERTICAL);
  ListboxSizer->Add(m_pListbox, 1, wxGROW);
  dlgSizer->Add(ListboxSizer, 1, wxGROW);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();
}

void ListboxDlg::OnListboxDClick(wxCommandEvent &event)
{
  wxString selection = m_pListbox->GetStringSelection();
  wxMessageBox("You double clicked on " + selection);
}

void ListboxDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(ListboxDlg, wxDialog)
  EVT_LISTBOX_DCLICK(ID_LISTBOX, ListboxDlg::OnListboxDClick)
  EVT_CLOSE(ListboxDlg::OnClose)
END_EVENT_TABLE()
```

In the OnListboxDClick function, you see that the content of the selected item is retrieved using the GetStringSelection method of the wxListBox class. The GetString method of wxCommandEvent doesn't work here.

7.10 wxRadioBox

A radiobox is a control that shows a list of mutually exclusive choices. The radio buttons can be displayed in a vertical or horizontal column.

The constructor:

```
wxRadioBox(wxWindow *parent, wxWindowID id, const wxString& title,
           const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,
           int n = 0, const wxString choices[] = NULL, int majorDim = 0,
           long style = wxRA_SPECIFY_COLS, const wxValidator& val = wxDefaultValidator,
           const wxString& name = "radiobox");
```

- **parent** is the parent window of the radiobox. This can't be NULL!
- **id** is the unique identifier for this window. Use –1, when you don't need it.
- **pos** is the position of the radiobox.
- **size** is the size of the radiobox.
- **n** is the number of radio buttons.

- **choices** is a wxString array containing the names of the radio buttons.
- **majorDim** specifies the maximum number of rows (when the style wxRA_SPECIFY_ROWS is used) or columns (when the style wxRA_SPECIFY_COLS is used) for a two-dimensional radiobox.
- **style** is the style of the radiobox. wxRadioBox has two additional styles:

wxRA_SPECIFY_ROWS	The major dimension parameter refers to the maximum number of rows.
wxRA_SPECIFY_COLS	The major dimension paramater refers to the maximum number of columns.

- **validator** is a validator for the radiobox.

7.10.1 Methods

bool Enable(bool enable)
void Enable(int item, bool enable)
All items or the given item are enabled or disabled.

int FindString(const wxString& s) const
The item with the given string is search. –1 is returned when no item is found.

wxString GetLabel(int item) const
The label of the item is returned.

int GetNumHor() const
The number of rows is returned.

int GetNumVer() const
The number of columns is returned.

int GetSelection() const
The index of the current selected item is returned. –1 is returned when there's no item selected. The index is zero-based.

wxString GetString(int N) const
The name of the item at the given index is returned.

wxString GetStringSelection() const
The name of the item that is selected is returned or an empty string when there's no item selected.

int Number() const
The number of items is returned.

void SetLabel(int item, const wxString& label)
The name of the item at the given index is set to the new label.

void SetSelection(int N)
The item at the given index is selected. An EVT_COMMAND_RADIOBOX_SELECTED event is not fired.

bool SetStringSelection(const wxString& s)
The item with the given name is selected. An EVT_COMMAND_RADIOBOX_SELECTED event is not fired.

bool Show(bool show)
void Show(int item, bool show)
Shows or hides all items or the given item.

7.10.2 Event handling

The EVT_COMMAND_RADIOBOX_SELECTED event is fired when an item is selected. Use the following macro for handling this event.

EVT_RADIOBOX(id, func)

7.10.3 Example

The example shows a dialog that contains a radiobox with three items. The items are placed in one column. When the user selects an item, a message box is shown with the name of the selected item.

Example 54. RadioboxDlg.h - The definition of the wxRadioBox example.

```
#ifndef _RadioboxDlg_H
#define _RadioboxDlg_H

class RadioboxDlg : public wxDialog
{
public:
  RadioboxDlg();

protected:
  void OnClose(wxCloseEvent &event);
  void OnRadiobox(wxCommandEvent &event);
  DECLARE_EVENT_TABLE()

private:
  enum
  {
    ID_RADIOBOX = 1000
  };

  wxRadioBox *m_pRadiobox;
};

#endif // _RadioboxDlg_H
```

In Example 55 you see that wxRA_SPECIFY_COLS is used to place the items in a column. Because all the items must be placed in one column, the maximum dimension parameter is set to 1.

Example 55. RadioboxDlg.cpp - The implementation of the wxRadioBox example.

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

#include "RadioboxDlg.h"

RadioboxDlg::RadioboxDlg() : wxDialog(NULL, -1, "RadioboxDialog", wxDefaultPosition,
wxSize(185, 185))
{
  wxString list[] = { "ABS", "Airbag", "Air Conditioning" };
  m_pRadiobox = new wxRadioBox(this, ID_RADIOBOX, " Car options ", wxDefaultPosition,
                             wxDefaultSize, 3, list, 1, wxRA_SPECIFY_COLS);

  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *RadioboxSizer = new wxBoxSizer(wxVERTICAL);
  RadioboxSizer->Add(m_pRadiobox, 1, wxGROW);
  dlgSizer->Add(RadioboxSizer, 1, wxGROW);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();
}

void RadioboxDlg::OnRadiobox(wxCommandEvent &event)
{
  wxString selection = m_pRadiobox->GetStringSelection();
  wxMessageBox("You selected " + selection);
}

void RadioboxDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(RadioboxDlg, wxDialog)
  EVT_RADIOBOX(ID_RADIOBOX, RadioboxDlg::OnRadiobox)
  EVT_CLOSE(RadioboxDlg::OnClose)
END_EVENT_TABLE()
```

7.11 wxRadioButton

A radio button is a button that is used to show a mutually exclusive selection.

What's the difference between wxRadioButton and wxRadioBox? wxRadioBox already groups the items, while with a wxRadioButton you have to specify where a group starts and ends. wxRadioBox also helps you to retrieve the selected item. In addition, wxRadioBox is responsible for the layout of the buttons in rows or columns. While with a wxRadioButton, you have to specify where the button is shown. What control do you use then? Well, when you're satisfied with the layout of wxRadioBox, then use wxRadioBox.

The constructor:

```
wxRadioButton(wxWindow *parent, wxWindowID id, const wxString& label,
              const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,
              long style = 0, const wxValidator& validator = wxDefaultValidator,
              const wxString& name = wxRadioButtonNameStr)
```

- **parent** is the parent window of the radio button. This can't be NULL!
- **id** is the unique identifier for this window. Use –1, when you don't need it.
- **label** is the text displayed with the radio button.
- **pos** is the position of the radio button.
- **size** is the size of the radio button.
- **style** is the window style of the radio button. wxRadioButton defines one additional style.

wxRB_GROUP	Marks the beginning of a new group of radio buttons.

- **validator** is the validator for the radio button.

7.11.1 Methods

bool GetValue(void) const
When the radio button is selected true is returned. Otherwise false is returned.

void SetValue(bool val)
Selects or deselects the radio button. The EVT_COMMAND_RADIOBUTTON_SELECTED event is not fired.

7.11.2 Event handling

The EVT_COMMAND_RADIOBUTTON_SELECTED event is fired when a radio button is selected. The EVT_RADIOBOX macro helps you to handle this event.

EVT_RADIOBOX(id, func)

In the current version of wxWindows (2.2.9), there's a known bug in wxRadioButton. When you handle the selection of a button and the button loses the focus, because you show a message box for example, the button will not be refreshed correctly. This bug will be solved in version 2.3.3.

7.11.3 Example

The example will try to implement the same functionality as in wxRadioBox. You'll see that the use of wxRadioBox is much easier.

Example 56. RadioButtonDlg.h - The definition of the wxRadioButton example.

```
#ifndef _RadiobuttonDlg_H
#define _RadiobuttonDlg_H

class RadiobuttonDlg : public wxDialog
{
public:
  RadiobuttonDlg();

protected:
  void OnClose(wxCloseEvent &event);
  void OnRadioButton(wxCommandEvent &event);
  DECLARE_EVENT_TABLE()

private:
  enum
  {
    ID_RADIOBUTTON_1 = 1000,
    ID_RADIOBUTTON_2,
    ID_RADIOBUTTON_3
  };

  wxRadioButton *m_pRadiobutton1;
  wxRadioButton *m_pRadiobutton2;
  wxRadioButton *m_pRadiobutton3;
};

#endif // _RadiobuttonDlg_H
```

In the constructor of the dialog, you'll see that the first radio button has the wxRB_GROUP style. This style indicates the start of a new group of radio buttons.

Example 57. RadioButtonDlg.cpp - The implementation of the wxRadioButton example.

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

#include "RadiobuttonDlg.h"

RadiobuttonDlg::RadiobuttonDlg() : wxDialog(NULL, -1, "RadiobuttonDialog",
                                            wxDefaultPosition, wxSize(185, 185))
{
  m_pRadiobutton1 = new wxRadioButton(this, ID_RADIOBUTTON_1, "ABS", wxDefaultPosition,
                                      wxDefaultSize, wxRB_GROUP);
  m_pRadiobutton2 = new wxRadioButton(this, ID_RADIOBUTTON_2, "Airbag");
  m_pRadiobutton3 = new wxRadioButton(this, ID_RADIOBUTTON_3, "Air Conditioning");

  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *RadioButtonSizer = new wxBoxSizer(wxVERTICAL);
  RadioButtonSizer->Add(m_pRadiobutton1);
  RadioButtonSizer->Add(m_pRadiobutton2);
  RadioButtonSizer->Add(m_pRadiobutton3);
  dlgSizer->Add(RadioButtonSizer, 1, wxGROW);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();
}

void RadiobuttonDlg::OnRadioButton(wxCommandEvent &event)
{
  wxString selection;
  wxRadioButton *current = (wxRadioButton*) NULL;
  switch(event.GetId())
  {
    case ID_RADIOBUTTON_1:
         current = m_pRadiobutton1;
         break;
    case ID_RADIOBUTTON_2:
         current = m_pRadiobutton2;
         break;
    case ID_RADIOBUTTON_3:
         current = m_pRadiobutton3;
         break;
  }
  // Warning: wxRadioButton doesn't like loosing the focus
  // in this event handler. So to keep the example simple
  // the message is put in the title of the dialog.
  // This problem will be solved in version 2.3.3.
  if ( current != (wxRadioButton*) NULL )
  {
    SetTitle("You've selected " + current->GetLabel());
  }
}

void RadiobuttonDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(RadiobuttonDlg, wxDialog)
  EVT_RADIOBUTTON(ID_RADIOBUTTON_1, RadiobuttonDlg::OnRadioButton)
  EVT_RADIOBUTTON(ID_RADIOBUTTON_2, RadiobuttonDlg::OnRadioButton)
  EVT_RADIOBUTTON(ID_RADIOBUTTON_3, RadiobuttonDlg::OnRadioButton)
  EVT_CLOSE(RadiobuttonDlg::OnClose)
END_EVENT_TABLE()
```

7.12 wxSlider

A slider is a control with a handle that the user can slide to change its value. Before Win95, a scrollbar is used to simulate a slider. In Win95 the track bar control is used. The correct implementation is chosen by wxWindows depending on the definition of __WIN95__. wxSlider is translated into wxSlider95 when __WIN95__ is defined, wxSliderMSW is used when it is not defined. You should always use wxSlider, instead of the concrete classes. Otherwise, you break the portability of your code.

The constructor:

```
wxSlider(wxWindow *parent, wxWindowID id, int value, int minValue, int maxValue,
         const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,
         long style = wxSL_HORIZONTAL, const wxValidator& validator = wxDefaultValidator,
         const wxString& name = wxSliderNameStr)
```

- **parent** is the parent window of the slider control. This can't be NULL!
- **id** is the unique identifier for the slider control. Use –1 when you don't need it.
- **value** is the initial value of the slider control.
- **minValue** is the minimum value of the slider control.
- **maxValue** is the maximum value of the slider control.
- **pos** is the position of the slider control.
- **size** is the size of the slider control.
- **style** is the style of the slider control. wxSlider provides the following additional styles:

wxSL_AUTOTICKS	Tick marks are displayed.
wxSL_BOTH	Tick marks are displayed on both sides.
wxSL_BOTTOM	When wxSL_HORIZONTAL is set, the ticks are on bottom.
wxSL_HORIZONTAL	The slider is displayed horizontally.
wxSL_LABELS	The minimum, maximum and current value labels are shown.
wxSL_LEFT	When wxSL_VERTICAL is set, the ticks are on the left.
wxSL_RIGHT	When wxSL_VERTICAL is set, the ticks are on the right.
wxSL_SELRANGE	The user can select a range on the slider (__WIN95__ only)
wxSL_TOP	When wxSL_HORIZONTAL is set, the ticks are on top.
wxSL_VERTICAL	The slider is displayed vertically.

- **validator** is the validator of the slider control.

7.12.1 Methods

void ClearSel() *__WIN95__ Only*
Clears a selection. This is only useful when wxSL_SELRANGE is used.

void ClearTicks() *__WIN95__ Only*
Clears the ticks.

int GetLineSize() const
Returns the number of steps the slider moves when the user moves it up or down.

int GetMax() const
Returns the maximum value.

int GetMin() const
Returns the minimum value.

int GetPageSize() const
Returns the number of steps the slider moves when the user pages up or down.

int GetSelEnd() const *__WIN95__ Only*
Returns the selection end point. This is only useful when wxSL_SELRANGE is used.

int GetSelStart() const *__WIN95__ Only*
Returns the selection start point. This is only useful when wxSL_SELRANGE is used.

int GetTickFreq() const *__WIN95__ Only*
Returns the tick mark frequency.

int GetThumbLength() const *__WIN95__ Only*
Returns the thumb length.

int GetValue() const
The current value is returned.

void SetLineSize(int lineSize)
The number of steps the slider moves when the user moves it up or down.

void SetPageSize(int pageSize)
Sets the number of steps the slider moves when the user pages up or down.

void SetRange(int minValue, int maxValue)
A new minimum and maximum value is set.

void SetSelection(int minPos, int maxPos) *__WIN95__ Only*
Sets the selection. This is only useful when wxSL_SELRANGE is used.

void SetTick(int tickPos) *__WIN95__ Only*
Sets a tick position.

void SetTickFreq(int n, int pos) *__WIN95__ Only*
Sets the tick mark frequency and position.

void SetThumbLength(int len) *__WIN95__ Only*
Sets the slider thumb length.

void SetValue(int)
Sets a new value.

7.12.2 Event handling

The event handling of a slider is done in the same way as a scrollbar. Each function receives a wxScrollEvent as argument.

EVT_COMMAND_SCROLL(id, func)	Handle all the scroll events.
EVT_COMMAND_TOP(id, func)	The slider is put on the top.
EVT_COMMAND_BOTTOM(id, func)	The slider is put on the bottom.
EVT_COMMAND_LINEUP(id, func)	Handle a line up command.
EVT_COMMAND_LINEDOWN(id, func)	Handle a line down command.
EVT_COMMAND_PAGEUP(id, func)	Handle a page up command.
EVT_COMMAND_PAGEDOWN(id, func)	Handle a page down command.
EVT_COMMAND_THUMBTRACK(id, func)	Handles a thumb track command.

7.12.3 Example

The example shows a dialog with a slider that shows its ticks, minimum value, maximum value and the current value. Note that you have to set the line and page size, otherwise clicking on the slider won't work.

Example 58. SliderDlg.h - The definition of the wxSlider example.

```
#ifndef _SliderDlg_H
#define _SliderDlg_H

class SliderDlg : public wxDialog
{
public:
  SliderDlg();

protected:
  void OnClose(wxCloseEvent &event);
  DECLARE_EVENT_TABLE()
private:
};
```

```
#endif
```

Example 59. SliderDlg.cpp - The implementation of the wxSlider example.

```
#include "wx/wx.h"

#include "SliderDlg.h"

SliderDlg::SliderDlg() : wxDialog(NULL, -1, "SliderDialog", wxDefaultPosition,
                                  wxSize(200, 150))
{
  wxSlider *slider = new wxSlider(this, -1, 5, 1, 10,
                                  wxDefaultPosition, wxSize(80, 30),
                                  wxSL_HORIZONTAL | wxSL_AUTOTICKS | wxSL_LABELS);

  slider->SetLineSize(1);
  slider->SetPageSize(2);

  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *sliderSizer = new wxBoxSizer(wxVERTICAL);
  sliderSizer->Add(slider, 0, wxALIGN_CENTER);
  dlgSizer->Add(sliderSizer, 0, wxALIGN_CENTER);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();
}

void SliderDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(SliderDlg, wxDialog)
  EVT_CLOSE(SliderDlg::OnClose)
END_EVENT_TABLE()
```

7.13 wxSpinCtrl

wxSpinCtrl combines a text control with a spin button. wxWindows also provides a wxSpinButton, but this is only implemented on Windows and GTK platforms. Therefore, portable programs should use wxSpinCtrl.

wxSpinCtrl is not automatically included when you use <wx.h>. You have to include <wx/spinctrl.h> yourself.

```
wxSpinCtrl(wxWindow *parent, wxWindowID id = -1, const wxString& value = wxEmptyString,
           const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,
           long style = wxSP_ARROW_KEYS, int min = 0, int max = 100, int initial = 0,
           const wxString& name = _T("wxSpinCtrl"))
```

- **parent** is the parent window of the control. This can't be NULL!
- **id** is the unique identifier for the control. Use –1 when you don't need it.
- **value** is the default value shown in the text control.
- **pos** is the position of the control.
- **size** is the size of the control.
- **style** is the window style of the control. wxSpinCtrl defines the following additional styles:

wxSP_ARROW_KEYS	The user can use the arrow keys.
wxSP_HORIZONTAL	Specifies a horizontal spin button (not on GTK)
wxSP_VERTICAL	Specifies a vertical spin button.
wxSP_WRAP	The value wraps when the minimum or maximum value is reached.

- **min** is the minimal value
- **max** is the maximum value
- **initial** is the initial value of the control.

7.13.1 Methods

int GetMax() const;
Returns the maximum value.

int GetMin() const;
Returns the minimum value

int GetValue() const
Returns the current value.

void SetRange(int minVal, int maxVal);
Sets a new minimum and maximum value.

void SetValue(const wxString& text)
void SetValue(int val)
Sets the value of the spin button.

7.13.2 Event handling

The only specific event fired is when the user changes the value of the control. Your handler function receives a wxSpinEvent as the only argument.

EVT_SPINCTRL(id, func)	Generated whenever the control is updated.

7.13.3 Example

The example shows a spin control in the middle of a dialog. You can select values from 0 to 10. When the maximum value is reached, the value will be reset to the beginning of the range.

Example 60. SpinDlg.h - The definition of the wxSpinCtrl example.

```
#ifndef _SpinDlg_H
#define _SpinDlg_H

class SpinDlg : public wxDialog
{
public:
  SpinDlg();

protected:
  void OnClose(wxCloseEvent &event);
  DECLARE_EVENT_TABLE()
private:

  enum
  {
    ID_SPIN = 1000
  };
};
```

Example 61. SpinDlg.cpp - The implementation of the wxSpinCtrl example.

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

#include "SpinDlg.h"

SpinDlg::SpinDlg() : wxDialog(NULL, -1, "SpinDialog", wxDefaultPosition, wxSize(200, 150))
{
  wxSpinCtrl *spin = new wxSpinCtrl(this, ID_SPIN, "", wxDefaultPosition,
                                    wxDefaultSize, wxSP_ARROW_KEYS | wxSP_WRAP, 0, 10);

  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *spinSizer = new wxBoxSizer(wxVERTICAL);
  spinSizer->Add(spin, 0, wxALIGN_CENTER);
  dlgSizer->Add(spinSizer, 1, wxALIGN_CENTER);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();
}

void SpinDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(SpinDlg, wxDialog)
  EVT_CLOSE(SpinDlg::OnClose)
END_EVENT_TABLE()
```

7.14 wxStaticBitmap

wxStaticBitmap displays a bitmap.

```
wxStaticBitmap(wxWindow *parent, wxWindowID id, const wxGDIImage& label,
               const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,
               long style = 0, const wxString& name = wxStaticBitmapNameStr)
```

- **parent** is the parent of this window. This can't be NULL!
- **id** is the unique identifier for this control. –1 can be used, when you don't need it.
- **label** is the bitmap to show.
- **pos** is the position of the window.
- **size** is the size of the window.
- **style** is the style of the window. wxStaticBitmap doesn't have any own defined styles.

7.14.1 Methods

const wxBitmap& GetBitmap() const;
Returns the bitmap. This method will assert when you've used an icon instead of a bitmap.

const wxIcon& GetIcon() const;
Returns the icon. This method will assert when you've used a bitmap instead of an icon.

void SetIcon(const wxIcon& icon);
Sets the icon.

void SetBitmap(const wxBitmap& bitmap);
Sets the bitmap.

7.14.2 Example

The example will show a bitmap in the middle of a dialog.

Example 62. StaticBitmapDlg.h - The definition of the wxStaticBitmap example.

```
#ifndef _StaticBitmapDlg_H
#define _StaticBitmapDlg_H

class StaticBitmapDlg : public wxDialog
{
public:
  StaticBitmapDlg();

protected:
  void OnClose(wxCloseEvent &event);
  DECLARE_EVENT_TABLE()
private:

};

#endif
```

Example 63. StaticBitmapDlg.cpp - The implementation of the wxStaticBitmap example.

```
#include <wx/wx.h>
#include "StaticBitmapDlg.h"

StaticBitmapDlg::StaticBitmapDlg() : wxDialog(NULL, -1, "StaticBitmapDialog",
wxDefaultPosition, wxSize(200, 150))
{
  wxStaticBitmap *staticBitmap = new wxStaticBitmap(this, -1, wxBitmap("button"));

  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  dlgSizer->Add(staticBitmap, 1, wxALIGN_CENTER);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();
}

void StaticBitmapDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(StaticBitmapDlg, wxDialog)
  EVT_CLOSE(StaticBitmapDlg::OnClose)
END_EVENT_TABLE()
```

Example 64. StaticBitmapApp.rc - The resource file of the wxStaticBitmap example.

```
button BITMAP "books.bmp"
#include "wx/msw/wx.rc"
```

7.15 wxStaticBox

A static box is a rectangle drawn around other controls to denote a logical grouping of items.

```
wxStaticBox(wxWindow *parent, wxWindowID id, const wxString& label,
            const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,
            long style = 0, const wxString& name = wxStaticBoxNameStr)
```

- **parent** is the parent window. This can't be NULL!
- **id** is the unique identifier for the static box. Use –1 when you don't need it.
- **label** is the text shown on the top of the static box.
- **pos** is the position of the control.
- **size** is the size of the control.
- **style** is window style. The static box doesn't have any special styles.

What's the easiest way to put controls in the static box? Moreover, how can you make sure that the size of the static box is big enough to show all the controls? The solution is wxStaticBoxSizer. It works like a box sizer and draws a static box around its current size. Read chapter 6.2.2 when you don't know how use sizers.

7.15.1 Example

The example creates two controls. wxStaticBoxSizer is used two make sure that the controls are laid out fine: the first control encloses the second one.

Example 65. StaticBoxDlg.h - The definition of the wxStaticBox example.

```
#ifndef _StaticBoxDlg_H
#define _StaticBoxDlg_H

class StaticBoxDlg : public wxDialog
{
public:
  StaticBoxDlg();
protected:
  void OnClose(wxCloseEvent &event);
  DECLARE_EVENT_TABLE()
private:

};

#endif
```

Example 66. StaticBoxDlg.cpp - The implementation of the wxStaticBox example.

```
#include <wx/wx.h>
#include "StaticBoxDlg.h"

StaticBoxDlg::StaticBoxDlg() : wxDialog(NULL, -1, "StaticBoxDialog", wxDefaultPosition,
wxSize(200, 150))
{
  wxStaticBox *staticBox1 = new wxStaticBox(this, -1, "Box 1");//, wxDefaultPosition,
wxDefaultSize);
  wxStaticBox *staticBox2 = new wxStaticBox(this, -1, "Box 2");//, wxDefaultPosition,
wxDefaultSize);

  wxBoxSizer *dlgSizer = new wxStaticBoxSizer(staticBox1, wxHORIZONTAL);
  wxBoxSizer *staticBoxSizer = new wxStaticBoxSizer(staticBox2, wxVERTICAL);
  dlgSizer->Add(staticBoxSizer, 1, wxGROW);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();
}

void StaticBoxDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(StaticBoxDlg, wxDialog)
  EVT_CLOSE(StaticBoxDlg::OnClose)
END_EVENT_TABLE()
```

7.16 wxStaticLine

wxStaticLine can be used to draw a vertical or horizontal line to separate groups of controls.

wxStaticLine is not automatically included when you use <wx.h>. You have to include <wx/statline.h> yourself.

```
wxStaticLine( wxWindow *parent, wxWindowID id, const wxPoint &pos = wxDefaultPosition,
              const wxSize &size = wxDefaultSize, long style = wxLI_HORIZONTAL,
              const wxString &name = wxStaticTextNameStr )
```

- **parent** is the parent window of the static line. This can't be NULL!
- **id** is the unique identifier for the line. Use –1 when you don't need it.
- **pos** is the position of the line.
- **size** is the size of the line.
- **style** is the style of the line. wxStaticLine has two additional styles:

wxLI_HORIZONTAL	Draws a horizontal line
wxLI_VERTICAL	Draws a vertical line

7.16.1 Methods

bool isVertical() const
Returns true when the line is vertical.

7.16.2 Example

The example draws a static line in the center of the dialog.

Example 67. StaticLineDlg.h - The definition of the wxStaticLine example.

```
#ifndef _StaticLineDlg_H
#define _StaticLineDlg_H

class StaticLineDlg : public wxDialog
{
public:
  StaticLineDlg();

protected:
  void OnClose(wxCloseEvent &event);
  DECLARE_EVENT_TABLE()
private:

};

#endif
```

Example 68. StaticLineDlg.cpp - The implementation of the wxStaticLine example.

```
#include <wx/wx.h>
#include <wx/statline.h>
#include "StaticLineDlg.h"

StaticLineDlg::StaticLineDlg() : wxDialog(NULL, -1, "StaticLineDialog",
                                          wxDefaultPosition, wxSize(200, 150))
{
  wxStaticLine *staticLine = new wxStaticLine(this, -1);

  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  dlgSizer->Add(staticLine, 1, wxALIGN_CENTER);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();
}

void StaticLineDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(StaticLineDlg, wxDialog)
  EVT_CLOSE(StaticLineDlg::OnClose)
END_EVENT_TABLE()
```

7.17 wxStaticText

wxStaticText allows you to show one or more lines of read-only text.

```
wxStaticText(wxWindow *parent, wxWindowID id, const wxString& label,
             const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,
             long style = 0, const wxString& name = wxStaticTextNameStr)
```

- **parent** is the parent window of the control. This can't be NULL!
- **id** is the unique identifier for the control. –1 can be used, when you don't need it.
- **label** is the text to show.
- **pos** is the position of the control.
- **size** is the size of the control.
- **style** is the style of the control. wxStaticText has the following additional styles:

wxALIGN_LEFT	The text is left aligned.
wxALIGN_RIGHT	The text is right aligned.
wxALIGN_CENTER	The text is centered horizontally.
wxST_NO_AUTORESIZE	When this style is set, the control is not automatically resized to exactly fit to the size of the text.

7.17.1 Methods

wxString GetLabel() const
Returns the text.

void SetLabel(const wxString& label);
Sets the text.

7.17.2 Example

The example will show a dialog that contains a multiline text.

Example 69. StaticTextDlg.h - The definition of the wxStaticText example.

```
#ifndef _StaticTextDlg_H
#define _StaticTextDlg_H

class StaticTextDlg : public wxDialog
{
public:
  StaticTextDlg();

protected:
  void OnClose(wxCloseEvent &event);
  DECLARE_EVENT_TABLE()
private:

};

#endif
```

Example 70. StaticTextDlg.cpp - The implementation of the wxStaticText example.

```
#include <wx/wx.h>
#include "StaticTextDlg.h"

StaticTextDlg::StaticTextDlg() : wxDialog(NULL, -1, "StaticTextDialog",
                                          wxDefaultPosition, wxSize(200, 150))
{
  wxStaticText *staticText = new wxStaticText(this, -1, "This is line 1\nThis is line 2");

  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  dlgSizer->Add(staticText, 1, wxALIGN_CENTER);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();
}

void StaticTextDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(StaticTextDlg, wxDialog)
  EVT_CLOSE(StaticTextDlg::OnClose)
END_EVENT_TABLE()
```

7.18 wxTextCtrl

A wxTextCtrl gives the user the ability to enter some text. The text can be on a single line or multi-line.

```
wxTextCtrl(wxWindow *parent, wxWindowID id, const wxString& value = wxEmptyString,
           const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,
           long style = 0, const wxValidator& validator = wxDefaultValidator,
           const wxString& name = wxTextCtrlNameStr);
```

- **parent** is the parent of the text control.
- **id** is the unique identifier. –1 can be used, when you don't need it.

- **value** is the initial text to show.
- **pos** is the position of the control.
- **size** is the size of the control.
- **style** is the window style of the control. wxTextCtrl defines some additional styles:

wxTE_PROCESS_ENTER	You will receive an EVT_COMMAND_TEXT_ENTER event when the user presses the enter key.
wxTE_PROCESS_TAB	You will receive an EVT_CHAR event when the user presses the tab key.
wxTE_MULTILINE	Multiple lines are allowed.
wxTE_PASSWORD	The entered text will be replaced by asterisks.
wxTE_READONLY	The text can't be changed by the user.
wxTE_RICH	A Rich Edit Text control will be created on Windows. Only when wxUSE_RICHEDIT was defined when wxWindows was compiled. (Does nothing on other platforms)
wxHSCROLL	A horizontal scrollbar is drawn. (Not on GTK+)

- **validator** is a validator for the control.

7.18.1 Methods

void AppendText(const wxString& text);
Appends the text.

bool CanCopy() const;
Returns true when the selection can be copied to the clipboard.

bool CanCut() const;
Returns true when the selection can be cut.

bool CanPaste() const;
Returns true when the content of the clipboard can be pasted to the text control.

bool CanRedo() const;
Returns true when there's a redo facility available.

bool CanUndo() const;
Returns true when there's a undo facility available.

void Clear();
Removes the text.

void Copy();
Copies the selected text to the clipboard.

void Cut();
Copies the selected text to the clipboard and removes it from the control.

void DiscardEdits();
Clears the modified flag.

long GetInsertionPoint() const;
Returns the current insertion point.

long GetLastPosition() const;
Returns the last position of the text control.

int GetLineLength(long lineNo) const;
Returns the length of the given line. A line number is zero-based.

wxString GetLineText(long lineNo) const;
Returns the text of the given line. A line number is zero-based.

int GetNumberOfLines() const;
Returns the number of lines.

void GetSelection(long* from, long* to) const;
Puts the start and end position of a selection in the given arguments. The values are the same when there is no selection.

wxString GetValue() const;
Returns the text.

bool IsEditable() const;
Returns true when the user can edit the text.

bool IsModified() const;
Returns true when the user changed the text.

bool LoadFile(const wxString& file);
Loads the file with the given name and puts the contents in the control.

void Paste();
Copies the content of the clipboard to the selection of the text control.

bool PositionToXY(long pos, long *x, long *y) const;
Converts the given position into a column (x) and a row (y).

void Redo();
When the last operation can be redone, it redoes the last operation.

void Remove(long from, long to);
Removes the text between the given selection.

void Replace(long from, long to, const wxString& value);
Replaces the text between the given selection with the new text.

bool SaveFile(const wxString& file = wxEmptyString);
Saves the text to a file. When no argument is passed then the argument of the last LoadFile call will be used.

void SetEditable(bool editable);
Enables/Disables the text control.

void SetInsertionPoint(long pos);
Sets the insertion point at the given position.

void SetInsertionPointEnd();
Sets the insertion point at the end of the text.

void SetSelection(long from, long to);
Sets the selection to the given arguments.

void SetValue(const wxString& value);
Sets the text.

void ShowPosition(long pos);
Makes sure that the line containing the position is visible.

void Undo();
Tries to undo the last change.

void WriteText(const wxString& text);
Inserts the given text at the current position.

long XYToPosition(long x, long y) const;
Converts the given column (x) and the given row (y) in a position in the text control.

The << operator can also be used to set the value of the text control.

wxTextCtrl& operator<<(const wxString& s);
Writes a string value to the text control.

wxTextCtrl& operator<<(int i);
Writes an int value to the text control.

wxTextCtrl& operator<<(long i);
Writes a long value to the text control.

wxTextCtrl& operator<<(float f);
Writes a float value to the text control.

wxTextCtrl& operator<<(double d);
Writes a double value to the text control.

wxTextCtrl& operator<<(const wxChar c);
Writes a wxChar value to the text control.

7.18.2 Events

wxTextCtrl has two specific events:

EVT_TEXT(id, func)	Generated whenever the control is updated. Also when the SetValue method is used.
EVT_TEXT_ENTER(id, func)	Generated whenever the user presses the enter key in a single line text control.

7.18.3 Example

The example shows a dialog that contains a multi-line text control that fills the total area of the dialog.

Example 71. TextDlg.h - The definition of the wxTextCtrl example.

```
#ifndef _TextDlg_H
#define _TextDlg_H

class TextDlg : public wxDialog
{
public:
  TextDlg();

protected:
  void OnClose(wxCloseEvent &event);
  DECLARE_EVENT_TABLE()
private:

};

#endif
```

Example 72. TextDlg.cpp - The implementation of the wxTextCtrl example.

```
#include <wx/wx.h>
#include "TextDlg.h"

TextDlg::TextDlg() : wxDialog(NULL, -1, "TextDialog", wxDefaultPosition, wxSize(200, 150))
{
  wxTextCtrl *text = new wxTextCtrl(this, -1, wxEmptyString,
                                    wxDefaultPosition, wxDefaultSize,
                                    wxTE_MULTILINE);

  *text << "This is line 1";

  wxBoxSizer *dlgSizer = new wxBoxSizer(wxHORIZONTAL);
  dlgSizer->Add(text, 1, wxGROW);
  SetSizer(dlgSizer);
  SetAutoLayout(TRUE);
  Layout();
}

void TextDlg::OnClose(wxCloseEvent &event)
{
  Destroy();
}

BEGIN_EVENT_TABLE(TextDlg, wxDialog)
  EVT_CLOSE(TextDlg::OnClose)
END_EVENT_TABLE()
```

8 Validators

A validator is an object that mediates between data and a control (mediator pattern), transferring the data in either direction and validating it. A validator is also able to intercept events generated by the control, providing a filtering behavior without the need to create a new control class.

You can use *wxTextValidator* and *wxGenericValidator*, or you can write your own.

8.1 How it works

To make sure that a validator works correctly, you must call the validator functions at the right time during dialog initialization and dismissal. When a dialog is shown the *InitDialog* method of *wxDialog* is automatically called. This method sends an initialization event to the dialog. The default implementation for the *wxEVT_INIT_DIALOG* event is defined in the *wxWindow* class and calls the *wxWindow::TransferDataToWindow* method. This function finds all validators in the windows' children and calls the *TransferToWindow* function for each child.

When you're using a panel or a window, you need to call *InitDialog* yourself before showing the window.

When the user clicks a button, the *wxWindow::Validate* method should be called. When this return FALSE, the button handler should return immediately. When it returns TRUE, the *TransferDataFromWindow* should be called.

wxDialog contains a default command event handler for the *wxID_OK* button. So when you're using validators and a normal OK button, you don't need to write any code for validating the dialog.

8.2 wxTextValidator

wxTextValidator validates *wxTextCtrl* objects. It provides also a filtering mechanism to prevent that the user enters invalid characters. The constructor looks like this:

```
wxTextValidator(long style = wxFilterNone, wxString* valPtr = NULL)
```

- **style** is a bit list of flags that is used to filter characters.
- **valPtr** is a pointer to a wxString variable that contains the value.

Example 73 is a text control that allows only numeric characters. When the control is validated, the text is stored in the textValue variable which type is a wxString.

Example 73. A Numeric Text Control with a wxTextValidator

```
wxTextCtrl *text = new wxTextCtrl(this, -1, "", wxDefaultPostion, wxDefaultSize, 0,
                                  wxTextValidator(wxFILTER_NUMERIC, &textValue));
```

8.3 wxGenericValidator

wxGenericValidator performs data transfer (no validation or filtering) for some basic controls.

wxCheckBox wxRadioButton	Transfer data to/from a bool variable.
wxListBox wxCheckListBox	Transfer data to/from a wxArrayInt variable.

wxButton wxChoice wxComboBox wxStaticText wxTextCtrl	Transfer data to/from a wxString variable.
wxGauge wxRadioBox wxSlider wxScrollBar wxSpinButton	Transfer data to/from an int variable.

8.4 Your own Validator

A new validator should provide the following functionality:

- ❑ A constructor is responsible for setting the kind of validation and to set a pointer to a C++ variable that is used to store the data of the control.
- ❑ A Validate method that returns TRUE when the data of the control is valid.
- ❑ A TransferToWindow method that transfers the data of the C++ variable to the control.
- ❑ A TransferFromWindow method that transfers the data of the control to the C++ variable.
- ❑ A Copy constructor and a Clone method is needed because validators are passed by reference to window constructors, and must therefore be cloned internally.

Optionally you can provide event handling. These events will be captured before the control does. The following example implements a validator that checks that the value entered in a text control is in a valid range.

Example 74. RangeValidator.h - A validator that checks that the value of a wxTextctrl is numeric and in a valid range.

```
#ifndef _RANGE_VALIDATOR_H
#define _RANGE_VALIDATOR_H

/**
 * RangeValidator. Checks if an entered value is in a given range.
 */

class RangeValidator : public wxValidator
{
public:
  /**
   * Constructor.
   */
  RangeValidator(wxString *value, long min, long max)
     : m_min(min), m_max(max), m_value(value) {}

  /**
   * Copy constructor
   */
  RangeValidator(const RangeValidator &toCopy);

  /**
   * Returns a copy of this validator
   */
  virtual wxObject* Clone() const;

  /**
   * Validate. Returns true when the entered value is correct, otherwise false
   */
  bool Validate(wxWindow *parent);

  /**
   * Transfer the data to the window. Returns false when there's a problem.
   */
  bool TransferToWindow();
  /**
   * Transfer the data from the window. Returns false when there's a problem.
   */
  bool TransferFromWindow();

protected:

  /**
   * Copies the given RangeValidator to this validator
   */
  bool Copy(const RangeValidator& val);

private:
  long m_min;
  long m_max;
  wxString *m_value;
};

#endif // _RANGE_VALIDATOR_H
```

Of course, you could extend from wxTextValidator, but this example is to show you how to create your own validator.

Example 75. RangeValidator.cpp – The implementation of the RangeValidator.

```
// For compilers that supports precompilation , includes "wx/wx.h"
#include "wx/wxprec.h"

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

#include "RangeValidator.h"

RangeValidator::RangeValidator(const RangeValidator &toCopy)
{
  Copy(toCopy);
}

wxObject* RangeValidator::Clone() const
{
  return new RangeValidator(*this);
}

bool RangeValidator::Copy(const RangeValidator& val)
{
  wxValidator::Copy(val);

  m_min   = val.m_min;
  m_max   = val.m_max;
  m_value = val.m_value;

  return true;
}

bool RangeValidator::Validate(wxWindow *parent)
{
  wxWindow *win = GetWindow();
  if ( !win )
     return false;
  wxTextCtrl *control = (wxTextCtrl*) win;

  wxString message;
  bool result = false;

  wxString currentValue = control->GetValue();

    long value;
    if ( ! currentValue.ToLong(&value) )
    {
      message.Printf("%s is not a valid number", currentValue.c_str());
    }
    else
    {
      if (    value < m_min
           || value > m_max )
      {
        message.Printf("%ld is not between %ld and %ld", value, m_min, m_max);
      }
      else
      {
        result = true;
      }
    }

  if ( ! result )
  {
    wxMessageBox(message);
    control->SetFocus();
  }

  return result;
}

bool RangeValidator::TransferToWindow()
{
  wxWindow *win = GetWindow();
  if ( ! win )
     return false;
  if ( ! m_value )
     return false;
  if ( ! win->IsKindOf(CLASSINFO(wxTextCtrl)) )
      return false;

  wxTextCtrl *control = (wxTextCtrl*) win;
  control->SetValue(*m_value);

  return true;
}

bool RangeValidator::TransferFromWindow()
{
  wxWindow *win = GetWindow();
  if ( ! win )
     return false;
  if ( ! m_value )
     return false;
  if ( ! win->IsKindOf(CLASSINFO(wxTextCtrl)) )
      return false;

  wxTextCtrl *control = (wxTextCtrl*) win;
  *m_value = control->GetValue();

  return true;
}
```

To use this validator you have to create a text control and a class member that stores the value of the text control. Example 76 shows you how the RangeValidator is used to force the user to enter a valid line number. The variable m_lineNumber contains the value that is shown in the text control. When the user clicks the Ok button, wxWindows performs the default event handling for this button. This is validating the input and transferring it to your C++ variable. When you implement the EVT_INIT_DIALOG event, don't forget to call *TransferDataToWindow*, otherwise the controls are not initialized.

Example 76. Using the RangeValidator.

```
void GotoLineDialog::OnInitDialog(wxInitDialogEvent &event)
{
  RangeValidator rangeValidator(&m_lineNumber, 1, m_maxLineNumber);
  m_pLineNumberCtrl->SetValidator(rangeValidator);

  TransferDataToWindow();
}
```

9 Scrolling

10 Splitting windows

Sometimes it can be useful to be able to split a window. The *wxSplitterWindow* class helps you doing this. A window can split vertically or horizontally.

11 Common controls

11.1 Toolbar

A toolbar is a user interface component that contains bitmap buttons or toggles. A toolbar gives the user faster access to an application's facilities than menus.

wxWindows provides several implementations of the *wxToolBarBase* class. Depending on the platform used, *wxToolBar* is one of these implementations. The selection of the implementation is done in the toolbar.h header file.

A toolbar on a frame is created with the *CreateToolBar* method.

11.1.1 Events

The toolbar class handles menu commands in the same way that a menubar does. Therefore, you can use one EVT_MENU macro for both a menu item and a toolbar button. The event handler functions take a *wxCommandEvent* argument. For most event macros, the identifier of the tool is passed, except for EVT_TOOL_ENTER that passes the toolbar window and the tool id is retrieved from the *wxCommandEvent*.

Tool commands (and UI update events) are first sent to the focus window within the frame that contains the toolbar. If no window within the frame has the focus, then the events are sent directly to the toolbar.

Processing toolbar events can be done with the following macros:

- EVT_TOOL(id, func)
  Process a wxEVT_COMMAND_TOOL_CLICKED event.
- EVT_MENU(id, func)
  The same as EVT_TOOL.
- EVT_TOOL_RANGE(id1, id2, func)
  Process a wxEVT_COMMAND_TOOL_CLICKED event for a range of id identifiers.
- EVT_MENU_RANGE(id1, id2, func)
  The same as EVT_TOOL_RANGE.
- EVT_TOOL_RCLICKED(id, func)
  Processes the right button mouse click on a tool.
- EVT_TOOL_RCLICKED_RANGE(id1, id2, func)
  Processes the right button mouse click for a range of tools.
- EVT_TOOL_ENTER(id, func)
  Process a wxEVT_COMMAND_TOOL_ENTER event.

11.2 Tree Control

The tree control displays its items in a tree like structure. Each item has its own optional icon and a label. An item may be either collapsed (meaning its children are not visible) or expanded (meaning its children are visible. Each item in tree is identified with a unique id. *wxTreeCtrl* is the class that implements the tree control.

In Windows32 applications, wxTreeCtrl uses the standard common treeview control from the system library comctl32.dll. Some versions of this library are known to have bugs with handling the tree control colors. When you encounter these problems, the recommended solution is to upgrade the comctrl32.dll to a newer version.

The constructor of wxTreeCtrl looks like this:

```
wxTreeCtrl(wxWindow *parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition,
           const wxSize& size = wxDefaultSize, long style = wxTR_HAS_BUTTONS,
           const wxValidator& validator = wxDefaultValidator,
           const wxString& name = "treeCtrl");
```

- **parent** is the parent of the tree control. This must not be NULL.
- **id** is the window identifier. –1 indicates a default value.
- **pos** is the window position.
- **size** is the window size.
- **style** is the style of the window. Multiple styles can be specified with the or operator.
  wxTR_HAS_BUTTONS
  Use this style to show + and – buttons to the left of parent items.(Win32 only)
  wxTR_EDIT_LABELS
  Use this style when the user is allowed to change the text of the label.
  wxTR_MULTIPLE
  Use this style when the user is allowed to select more than one item.
- **validator** is a validator for this window.
- **name** is the window name.

11.2.1 Items

Each item has a unique id. The type of an id depends on the platform. That's why wxWindows wraps the id in the *wxTreeItemId* class. For example under Windows32, this class wraps around the HTREEITEM handle. This way, wxWindows makes sure that your code is portable.

Before you can add items to the tree, you need to add a root item. The *AddRoot* method is used for this. *AddRoot* returns the id of the item. Later you can use this id to add children to the root with the *AppendItem* method. When you want to insert an item at the same level, you use the *InsertItem* method. Inserting an item as the first child of his parent can be done with the *PrependItem* method.

To associate data with your items you can use the *wxTreeItemData* class. Your data class should derive from *wxTreeItemData*. The association is set with the *SetItemData* method. Because the tree itself controls the pointers to the wxTreeItemData instances, you must create the data classes on the heap and you are not allowed to delete them. Once a data class is set, the tree will delete it when the tree is destroyed. If you do delete a data class, make sure that you've cleared the association with the *SetItemData* (with NULL as the second argument) method to prevent that the tree deletes the pointer for the second time.

11.2.2 Enumerating items

There are three ways of enumerating items: for the children of a given item, for the sibling of the given item and for the visible items (those that are currently shown to the user, which means that the item is expanded and in the current view).

Enumerating the children of items is done with the GetFirstChild and GetNextChild methods. For these enumerating methods, you pass a 'cookie' parameter, which is necessary to make these functions reentrant. The cookie (type long) passed to these methods must be the same for each enumeration.

Enumerating the sibling item of a given item is done with the GetNextSibling and GetPrevSibling methods.

Enumerating the visible items is done with the GetFirstVisibleItem, GetNextVisible and GetPrevVisible methods.

All these enumerating methods return an invalid id when an item is not found. Use the *IsOk* method of *wxTreeItemId* to check this.

11.2.3 Using images

Setting and getting the image of an item is done respectively with *GetItemImage* and *SetItemImage*. Use the enumeration *wxTreeItemIcon* to get the right image. An item can have four different kinds of images:

- A normal image is shown in normal circumstances. (*wxTreeItemIcon_Normal*)
- A selected image is shown when the item is selected. (*wxTreeItemIcon_Selected*)
- An expanded image is shown when the item is expanded. (*wxTreeItemIcon_Expanded*)
- A selected expanded image is shown when the item is selected and expanded. (*wxTreeItemIcon_SelectedExpanded*)

You can also use the *wxImageList* class.

11.2.4 Sorting items

Sorting items is done with the *SortChildren* method. Items are compared in the *OnCompareItems* method, which you can override to provide your own comparison. The default is ascending alphabetical order.

11.2.5 Tree control events

To process events from a tree control you can use the following event handler macros.

- EVT_TREE_BEGIN_DRAG(id, func)
  Begin dragging with the left mouse button. Call the wxNotifyEvent Allow method to allow dragging.
- EVT_TREE_BEGIN_RDRAG(id, func)
  Begin dragging with the right mouse button. Call the wxNotifyEvent Allow method to allow dragging.
- EVT_TREE_END_DRAG(id, func)
  Drag ended (drop).
- EVT_TREE_BEGIN_LABEL_EDIT(id, func)
  Begin editing a label. Call the wxNotifyEvent Veto method to disallow editing.
- EVT_TREE_END_LABEL_EDIT(id, func)
  Finished editing a label. Call the wxNotifyEvent Veto method to disallow the change.
- EVT_TREE_DELETE_ITEM(id, func)
  Delete an item.
- EVT_TREE_GET_INFO(id, func)
  Request information from the application.
- EVT_TREE_SET_INFO(id, func)
  Information is supplied.
- EVT_TREE_ITEM_ACTIVATED(id, func)
  The item is activated.
- EVT_TREE_ITEM_COLLAPSED(id, func)
  The item is collapsed.
- EVT_TREE_ITEM_COLLAPSING(id, func)
  The item is about to be collapsed. Prevent this by calling the wxNotifyEvent Veto method.
- EVT_TREE_ITEM_EXPANDED(id, func)
  The item is expanded.
- EVT_TREE_ITEM_EXPANDING(id, func)
  The item is about to be expanded. Prevent this by calling the wxNotifyEvent Veto method.
- EVT_TREE_SEL_CHANGED(id, func)
  The selection has changed.
- EVT_TREE_SEL_CHANGING(id, func)
  The selection is about to be changed. Prevent this by calling the wxNotifyEvent Veto method.

- EVT_TREE_KEY_DOWN(id, func)
  A key is pressed.

The member functions take a *wxTreeEvent*, derived from *wxNotifyEvent*, as an argument.

11.2.6 Editing items

When you've created the tree control with the wxTR_EDIT_LABELS style, the user can click on a label to start editing. On Win32 systems, you can also start editing the label, by calling the *EditLabel* method. Whenever the user tries to start editing a label the tree control gets an EVT_TREE_BEGIN_LABEL_EDIT event which can be vetoed so that the user can't change the label. When the user ends editing, an EVT_TREE_END_LABEL_EDIT event is send to the tree control. This event can also be vetoed, which means that the label keeps unchanged.

11.3 List Control

A list control displays its items in a list in the following formats: list view, report view, icon view and small icon view. wxListCtrl is implemented in a generic way on all platforms except Windows32. The constructor of wxListCtrl looks like this:

```
wxListCtrl(wxWindow *parent, wxWindowID id = -1, const wxPoint& pos = wxDefaultPosition,
           const wxSize& size = wxDefaultSize, long style = wxLC_ICON,
           const wxValidator& validator = wxDefaultValidator,
           const wxString& name = _T("wxListCtrl"));
```

- **parent** is the parent window.
- **id** is the unique id of the list control.
- **pos** is the position of the control.
- **size** is the size of the control.
- **style** is the style of the control.

wxLC_LIST	Multicolumn list view, with optional small icons.
wxLC_REPORT	Single or multicolumn report view, with optional header.
wxLC_ICON	Large icon view, with optional labels.
wxLC_SMALL_ICON	Small icon view, with optional labels
wxLC_ALIGN_TOP	Icons align to the top. (Win32 only)
wxLC_ALIGN_LEFT	Icons align to the left.
wxLC_AUTOARRANGE	Icons arrange themselves. (Win32 only)
wxLC_USER_TEXT	The application provides label text on demand. (Win32 only)
wxLC_EDIT_LABELS	Labels are editable.
wxLC_NO_HEADER	No header in report mode. (Win32 only)
wxLC_SINGLE_SEL	Only one item can be selected.
wxLC_SORT_ASCENDING	Sort in ascending order.
wxLC_SORT_DESCENDING	Sort in descending order.

- **validator** is the validator.
- **name** is the window name.

11.3.1 Methods

11.3.2 wxListItem

An item is inserted with the *InsertItem* method that returns the index of the inserted item when it succeeds or -1 when it fails. When you want to set several attributes of an item, you can use the *wxListItem* class. Associating data with an item is done with the *SetItemData* method or you can use the *wxListItem* class again.

When you have a list control in report view, you can set sub-items with the *SetColumn* or the *SetItem* method. *SetItem* can also be used with the *wxListItem* class.

wxListItem is used together with wxListCtrl. Use this class to set/get several attributes of an item in a wxListCtrl at once.

void Clear()
Clears all the members.

void ClearAttributes()
Clears all the attributes (font, color, ...)

wxListColumnFormat GetAlign () const
Returns the format of a column. See SetAlign for more information.

wxColour GetBackgroundColour () const
Returns the background color of the item.

int GetColumn () const
Returns the column.

long GetData () const
Returns a pointer to application defined data.

wxFont GetFont () const
Returns the font of the item.

long GetId () const
Returns the index of the item.

int GetImage () const
Returns the image index in the ImageList of the item.

long GetMask() const
Returns the mask. See SetMask for more information.

long GetState () const
Returns the State of the item. See SetState for more information.

const wxString& GetText () const
Returns a const reference to the text of the item.

wxColour GetTextColour () const
Returns the color of the text.

int GetWidth () const
Returns the width of the item.

bool HasAttributes () const
Returns true when the item has associated attributes (font, color, ...)

void SetAlign (wxListColumnFormat align)
A format for a column.

wxLIST_FORMAT_LEFT	The text will be left aligned.
wxLIST_FORMAT_RIGHT	The text will be right aligned.

wxLIST_FORMAT_CENTRE, wxLIST_FORMAT_CENTER	The text will be centered.

void SetBackgroundColour (const wxColour& colBack)
Sets the background color of the item.

void SetColumn (int col)
Sets the index of the column.

void SetData (long data)
void SetData(void *data)
Sets a pointer to application-defined data.

void SetFont (const wxFont& font)
Sets the font of the item.

void SetId (long id)
Sets the index of the item.

void SetImage (int image)
Sets the index of the image in the associated ImageList.

void SetMask (long mask)
Sets a flag that indicates which fields are valid. You can combine these flags with the OR-operator.

wxLIST_MASK_STATE	The m_state field is valid.
wxLIST_MASK_TEXT	The m_text field is valid.
wxLIST_MASK_IMAGE	The m_image field is valid.
wxLIST_MASK_DATA	The m_data field is valid.
wxLIST_MASK_WIDTH	The m_width field is valid.
wxLIST_MASK_FORMAT	The m_format field is valid.

void SetState (long state)
Sets the state of the item.

wxLIST_STATE_DONTCARE	Don’t care what the state is. (Windows32 only)
wxLIST_STATE_DROPHILITED	The item is highlighted to receive a drop event. (Windows32 only)
wxLIST_STATE_FOCUSED	The item has the focus.
wxLIST_STATE_SELECTED	The item is selected.
wxLIST_STATE_CUT	The item is in cut state. (Windows32 only)

void SetStateMask (long stateMask)
A mask indicating which state flags are valid. See SetState for all flags.

void SetText (const wxString& text)
Sets the text of the item.

void SetTextColour (const wxColour& colText)
Sets the foreground color of the item.

void SetWidth (int width)
Sets the width of the column.

11.3.3 List control events

To process events from a list control you use the following event macros:

- EVT_LIST_BEGIN_DRAG(id, func)
  Begin dragging with the left mouse.
- EVT_LIST_BEGIN_RDRAG(id, func)
  Begin dragging with the right mouse button.
- EVT_LIST_BEGIN_LABEL_EDIT(id, func)
  Begin editing a label. Call the wxNotifyEvent Veto method to disallow editing.
- EVT_LIST_END_LABEL_EDIT(id, func)
  Editing a label is finished. Call the wxNotifyEvent Veto method to discard the change.
- EVT_LIST_DELETE_ITEM(id, func)
  An item is deleted.
- EVT_LIST_DELETE_ALL_ITEMS(id, func)
  All items are deleted.
- EVT_LIST_GET_INFO(id, func)
  Request information from the application, usually the item text.
- EVT_LIST_SET_INFO(id, func)
  Information is supplied. (not implemented)
- EVT_LIST_ITEM_SELECTED(id, func)
  The item is selected.
- EVT_LIST_ITEM_DESELECTED(id, func)
  The item is deselected.
- EVT_LIST_ITEM_ACTIVATED(id, func)

11.3.4 Editing items

When you've set the wxLC_EDIT_LABELS style, the user can edit the items. When the user clicks on a selected item, a text control is shown. When the user doesn't press the escape key, the item is changed. Before the editing starts, the application receives an EVT_LIST_BEGIN_LABEL_EDIT, which can be vetoed so that no text control will appear. When the user has changed the label, the application receives an EVT_LIST_END_LABEL_EDIT, which can also be vetoed so that the item is not changed.

11.4 Imagelist

A wxImageList contains a list of images and is used together with wxTreeCtrl, wxNotebook and wxListCtrl.

11.5 Tab

A Tab control is created with the *wxNotebook* control. The tabs can be placed on the left, right, top or bottom side of the control.

This is the constructor:

```
wxNotebook(wxWindow* parent, wxWindowID id, const wxPoint& pos = wxDefaultPosition,
           const wxSize& size, long style = 0, const wxString& name = "notebook");
```

- **parent** is the parent window
- **id** is the window identifier
- **pos** is the position of the window
- **size** is the size of the window
- **style**

wxNB_FIXEDWIDTH	All tabs have the same width. (Windows only)
wxNB_LEFT	Place tabs on the left side.
wxNB_RIGHT	Place tabs on the right side.
wxNB_BOTTOM	Place tabs on the bottom of the control.

- **name** is the name of the control

The default placement of the tabs is at the top of the control. To reduce flickering you can use the wxCLIP_CHILDREN window style.

11.5.1 Tab pages

Any window can be a page for a tab. *wxNotebookPage* is just a typedef for wxWindow. A page is added with the AddPage or InsertPage method. When you add a window as a page, you're not allowed to delete this window. This will be done by *wxNotebook* when the control is destroyed. When you don't want this behavior, you can use the RemovePage method. RemovePage deletes the specified page, without deleting the associated window.

11.5.2 Events

To process events from a wxNotebook control you can use the following macros:

- EVT_NOTEBOOK_PAGE_CHANGED(id, func)
  The page selection was changed.
- EVT_NOTEBOOK_PAGE_CHANGINGt(id, func)
  The page selection is about to be changed. This can be prevented by calling Veto.

11.6 Tooltip

12 Property Sheets

13 HTML

14 Document / View framework

15 The clipboard

16 Drag & drop

17 Printing

18 Drawing

19 Strings

20 Date & Time

21 Container classes

wxWindows provides some container classes such as dynamic arrays and hashtables.

wxWindows doesn't use the STL container classes for some reasons. First, wxWindows was written before STL was written. Second, there are still compilers today that cannot deal well with all of the STL classes. When the support for STL improves, future versions of wxWindows will use STL.

21.1.1 Dynamic Arrays

wxWindows provides three different kinds of arrays. All of them derive from the *wxBaseArray* class, which handles the untyped data. *wxBaseArray* can't be used directly. wxWindows has some macros that you can use to generate a new class that derives from *wxBaseArray*. Because of portability issues, wxWindows doesn't use templates.

wxBaseArray provides you dynamic arrays. When you add an element and there's not enough memory allocated, *wxBaseArray* will allocate more memory. In debug mode, *wxBaseArray* also performs range checking on the index values.

For simple types, the macro WX_DEFINE_ARRAY can be used. A simple type is a type which size is smaller then sizeof(long). Example 77 shows you how it works. The macro generates a class with the name IntArray that can only contain int's.

Example 77. A dynamic array of a simple type.

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

WX_DEFINE_ARRAY(int, IntArray) // Defines a dynamic unsorted array of int's.
```

For types with a larger size then sizeof(long) or for objects you need to use the WX_DECLARE_OBJARRAY macro. This macro only declares the array. After you've included the <wx/arrimpl.cpp> file in your implementation file, you need to use WX_DEFINE_OBJARRAY to define the array.

Example 78. Declaring a dynamic array for an object.

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

WX_DECLARE_OBJARRAY(wxFile, ArrayOfFiles);
```

Example 79. Defining a dynamic array for an object.

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

Example 78 shows you how to declare an array class for wxFile instances, while Example 79 shows you how to define the array class.

A sorted array is created with the WX_DEFINE_SORTED_ARRAY macro as you can see in Example 80.

Example 80. A sorted dynamic array.

```
WX_DEFINE_SORTED_ARRAY(wxFile*, SortedArrayOfFiles);
```

21.1.2 wxHashTable

The wxHashTable provides hash table functionality for wxWindows. wxHashTable is kept simple: you can only use integers or strings as the key. The hash table is implemented as an array of pointers to lists. The constructor of wxHashTable looks like this:

```
wxHashTable(unsigned int key_type = wxKEY_INTEGER, int size = wxHASH_SIZE_DEFAULT);
```

- **key_type** defines the type of the key. wxKEY_INTEGER for int, wxKEY_STRING for string.
- **size** is the default size of the hash table. The default contains space for 1000 elements.

You can use the WX_DECLARE_HASH macro to define a type safe hash table. This macro generates a class for you.

```
WX_DECLARE_HASH(wxFile, wxList, FileHashTable)
```

21.1.3 Use STL

Before you can use STL together with wxWindows you have to change the following macros in the “setup.h” header file and set them as shown below. You find this file in the “include/wx” directory.

```
#define wxUSE_GLOBAL_MEMORY_OPERATORS 0
#define wxUSE_DEBUG_NEW_ALWAYS 0
```

You’ve to do this because wxWindows redefines new for debugging purposes and this gives trouble with STL.

22 File handling

23 Multi-lingual applications

24 Logging

25 Databases

26 Multithreading

27 Debugging

28 Class reference

28.1 *wxAcceleratorEntry*

28.1.1 Overview

An object used by an application, which want to create an accelerator table.

- Derived from

  None

- Include files

  #include <wx/accel.h>

- See also

  wxAcceleratorTable

28.1.2 Example

28.2 wxAcceleratorTable

28.2.1 Overview

28.3 wxFrame

28.3.1 Overview

- See also

Appendix A: Event Macros

This is a list of all the classes that are derived from wxEvent and their associated macros for event handling.

Event	Macros	
wxActivateEvent	EVT_ACTIVATE(func)	An activate event is sent when a window or application is activated or deactivated.
	EVT_ACTIVATE_APP(func)	
wxCommandEvent	EVT_COMMAND(id, event, func)	
	EVT_COMMAND_RANGE(id1, id2, event, func)	
	EVT_BUTTON(id, func)	
	EVT_CHECKBOX(id, func)	
	EVT_CHOICE(id, func)	
	EVT_LISTBOX(id, func)	
	EVT_LISTBOX_DCLICK(id, func)	
	EVT_TEXT(id, func)	
	EVT_TEXT_ENTER(id, func)	
	EVT_MENU(id, func)	
	EVT_MENU_RANGE(id1, id2, func)	
	EVT_SLIDER(id, func)	
	EVT_RADIOBOX(id, func)	
	EVT_RADIOBUTTON(id, func)	
	EVT_SCROLLBAR(id, func)	
	EVT_COMBOBOX(id, func)	
	EVT_TOOL(id, func)	
	EVT_TOOL_RANGE(id1, id2, func)	

Appendix B: Example List