# Slides to Accompany *Programming Languages and Methodologies*

**R. J. Schalkoff**

## Chapter 13, Part 2: wxWidgets

# Cross and Multi-Platform Software Development

**Cross-platform development of software involves developing software on one type of machine to run on another type of machine.**

1. It is both necessary and common.
2. Often it is necessary to develop applications which run on multiple platforms.
3. It is somewhat inefficient to develop separate applications for each platform.
4. Numerous tool exist to facilitate multi-platform and cross-platform software development.

## Cross-Platform Development: wxWidgets

1. Like the MFC, a tool based upon a class library for developing GUI-oriented C++ programs on a variety of different platforms.
2. Defines a common API across platforms, but uses the native graphical user interface (GUI) on each platform.
3. Programs developed with wxWidgets exhibit the native 'look and feel' for each platform, yet have common functionality.
4. Distributions of wxWidgets and documentation are available at:

   `http://wxidgets.org/`

# Platform-Specific Libraries

Once a program is written to the wxWidgets API, the platform-specific libraries are invoked by the wxWidgets API. This is shown in Figure 1.

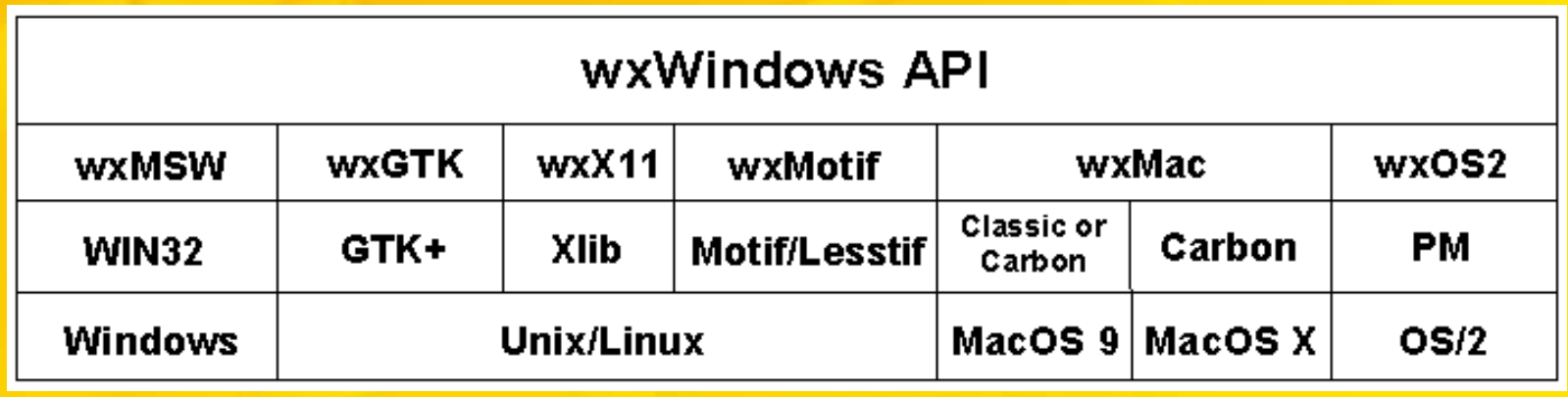


Figure 1: Relationship of the wxWidgets API to Platform-Specific Graphics Libraries

## `wxWidgets` Fundamentals: General Notes

1. All wxWidgets applications need need a derived wxApp class and to override the constructor `wxApp::OnInit`.
2. Every application must have a top-level wxFrame or wxDialog window, derived from the respective class.
   - Each frame may contain one or more instances of classes such as wxPanel, wxSplitterWindow or other windows and controls.
   - A frame can have a wxMenuBar, a wxToolBar, a status line, and a wxIcon (used when the frame is 'iconized'). wxPanel is used to place controls (classes derived from wxControl which are used for user interaction.
   - Examples of controls are wxButton, wxCheckBox, wxChoice , wxListBox and wxSlider.

3. An instance of wxDialog can be used for implementing controls. This strategy has the advantage of not requiring a separate frame. This is shown in the 'button' example.

4. For dialogs, the use of wxBoxSizer, for simple windows, may produce acceptable layouts with a minimum of design effort and coding.

5. Argument default values may be omitted from a function call. In addition, size and position arguments may usually be given a value of -1 (the default), in which case wxWidgets will choose a suitable value.

6. Like the MFC, windows (frames and dialogs) and controls in `wxWidgets` programs are referenced by pointers to objects and thus created using `new`[a].

[a]Note `delete` is not used to free these resources because `wxWidgets` takes care of this.

7. An event table is used to map events to (handler) functions. Registering events with user-written functions is achieved using one or more

   `BEGIN_EVENT_TABLE ... END_EVENT_TABLE` macros. An example is:

```
BEGIN_EVENT_TABLE(ButtonDlg, wxDialog)
  EVT_BUTTON(BUTTON_SELECT, ButtonDlg::ButtonSelect)
  EVT_CLOSE(ButtonDlg::OnCloseWindow)
END_EVENT_TABLE()
```

   Note that the event table is *specific to a frame*. In the above example, the `BEGIN_EVENT_TABLE` macro indicates that events from the ButtonDlg frame (derived from the wxDialog class) are intercepted and handled. Also note that a `DECLARE_EVENT_TABLE` macro must be included in the corresponding class definition.

## Where's `main`?

Like the MFC approach for MS Windows, wxWidgets `c++` source files for applications *do not explicitly contain the* `main` *function.* In wxWidgets, the `IMPLEMENT_APP` macro creates an application instance and starts the wxWidgets program. The prototype is:

```
IMPLEMENT_APP(theApp)
```

where class `theApp` is derived from class `wxApp`. Recall `wxApp::OnInit()` is called upon class instance creation (startup) and is used to start the wxWidgets program.

# wxWidgets: Events and Handling

1. The mapping of events to functions is achieved using a `BEGIN_EVENT_TABLE ... END_EVENT_TABLE` macro block.
2. Between these macros, specific event macros corresponding to specific resources are defined. These map the event (e.g., a button press or a mouse click) to an event handling function.
3. For example, consider the use of a button. The `EVT_BUTTON` macro may be used. The basic prototype is:

   ```
   EVT_BUTTON(id, func)
   ```

   which indicates `func` is invoked when a user clicks the button with identifier `id`.

# A Dialog and Button-based wxWidgets Example

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

class ButtonApp : public wxApp
{
public:
// Initialize the application
        virtual bool OnInit();
};

//main window is from wxDialog
class ButtonDlg : public wxDialog
{
public:
// constructor
  ButtonDlg();

  void ButtonSelect(wxCommandEvent &event);
  void OnCloseWindow(wxCloseEvent &event);
```

```
private:
  DECLARE_EVENT_TABLE()
  // ID of only event
  enum
  {
    BUTTON_SELECT = 1000
  };
};

ButtonDlg::ButtonDlg() : wxDialog((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);
  dlgSizer->SetSizeHints(this);
}
```

```
void ButtonDlg::ButtonSelect(wxCommandEvent &command)
{
  wxMessageBox("The button was pressed!");
}

// here's the event table

BEGIN_EVENT_TABLE(ButtonDlg, wxDialog)
  EVT_BUTTON(BUTTON_SELECT, ButtonDlg::ButtonSelect)
// terminate the dialog when the 'x' window button is clicked
  EVT_CLOSE(ButtonDlg::OnCloseWindow)
END_EVENT_TABLE()

bool ButtonApp::OnInit()
{
// create instance of the dialog
  ButtonDlg *button = new ButtonDlg();
// Show it
  button->Show(TRUE);
// Tell the application that it's our main window
  SetTopWindow(button);
```

```
  return true;
}

void ButtonDlg::OnCloseWindow(wxCloseEvent &event)
{
// close using Destroy rather than Close.
this->Destroy();
}

IMPLEMENT_APP(ButtonApp)
```

Noteworthy aspects of the previous example include:

1. The use of 'sizers', which relieves the programmer of much of the detailed manual layout decisions (see the `wxWidgets` manual);
2. The creation of a button resource via:
   `new wxButton(this, BUTTON_SELECT, "Select", wxDefaultPosition);`
3. The association of the event resulting from clicking on this button and a user-defined method:
   `EVT_BUTTON(BUTTON_SELECT, ButtonDlg::ButtonSelect)`
4. An illustration of the proper way to terminate an application if the main window is closed.