C++, GUI Programming, and You

Wayne Manselle

Introduction

C++ is a powerful Object Orientated Language that was created by Bjarne Stroustrup in 1983 at Bell Labs as an enhancement to the earlier programming language, C. As an Object Orientated Language, C++ lends itself nicely to Graphical User Interface design. However, unlike languages like Java, or the not so related Tcl, C++ does not have a pre-packaged graphical library like Swing, AWT, or TK. Because of this "limitation" numerous graphical libraries have been created for the C++ language. Three such libraries are QT, GTK+, and WxWidget which will each be covered in turn. As another solution, on the Microsoft Windows platform, Microsoft developed the Visual C++ language, which is a User Interface Development Environment that adds a tool to manipulate default Windows widgets and map C++ code to their operation.

This paper is divided into four sections, with each section devoted to one of the GUI building approaches for C++. For each approach, I will discuss its approach to the main event loop, available events, widgets, and their structure. Further, for each I will include the code for a simple "Hello World" graphical program.

Qt Toolkit

Qt is a cross-platform application product of Trolltech predominantly used for GUI program development, possibly made most famous as being the driving force behind the GUI presentation of the KDE XWindows front end. Qt, pronounced "cute" uses standard C++, but also includes pre-processing methods that generate the C++ code necessary for the implementation of its extensions. Qt is a beautifully designed OO library to use with C++, with the entire library being an extension of one class, QObject, much in the way that AWT/Swing extends Object. QObject is unique in how it adds language functionality to C++, such as a form of garbage collection.

Main Event Loop

The QEventLoop class is the base of Qt's entire event looping system, and it allows for the creation of event loops arbitrarily throughout the program. [1] The main event loop in Qt is an instance of this class contained inside of an instance of the QApplication class, which all GUI applications in Qt must extend. QEventLoops can be configured with one of six filters, which configure it to either accept all events, filter all user input out, filter all socket notifiers out, enter wait mode, defer the deletion of objects, or to ignore X11 timer events.
The main event loop in Qt functions by fetching native windowing system events, and adding them to the event queue. Events on the event queue are dealt with by the QEventLoop currently accessing the queue until there are no events left to deal with, in which case the event loop(s) enter a waiting state.

Events

Every event in Qt is an extension of of the QEvent class. [1] Most QEvents are generated by the windowing system, but it is possible to send them manually using the methods sendEvent() and postEvent() from the class QCoreApplication (or any ancestor thereof). QEvent has multiple ancestor classes, each of which in turn have their own ancestors, and to list a tree of them all would easily drag this paper beyond the ten page limit. The full extent of QEvent's functionality may be best seen in the Type enumeration coupled with it. The Type enumeration describes, appropriately, the type of event one is dealing with. Qt defines 107 different event types, each of which given a unique int identifier, and containing things as common as key presses and mouse moves, to things as specific as "whatsthis", which indicates a "What's This?" QHelpEvent. Further, the int values between 1000 and 65535 in the enumeration are reserved specifically for user defined events.

Widgets

Qt comes with a substantial array of built in widgets. [2] The parent of all these classes is the QWidget class, and its basic children are outlined in the following tree:

- QAbstractButton
  - Q3Button
  - QCheckBox
  - QPushButton
  - QRadioButton
  - QToolButton

- QAbstractSlider
  - QDial
  - QScrollBar
  - QSlider
- QAbstractSpinBox
  - QDateTimeEdit
  - QDoubleSpinBox
  - QSpinBox
- QComboBox
  - QFontComboBox
- QDialog
  - ...
  - QAbstractPrintDialog
  - QColorDialog
  - QErrorMessage
  - QFileDialog
  - QInputDialog
  - QMessageBox
  - QPageSetupDialog
  - QProgressDialog
- QFrame
  - QAbstractScrollArea
  - QLabel
  - QLCDNumber
  - QSplitter
  - QStachedWidget
  - QToolBox
- QMainWindow
- QMenu
  - Q3PopupMenu
- QMenuBar
- QProgressBar
- QStatusBar
- QTabBar
- QToolBar

You may wonder why I put this tree in here when I wouldn't put the event tree, and my answer to that is: There weren't 107 predefined widgets to list. Naturally, these are all widgets that readers of this paper are assuredly familiar with from the other GUI libraries such as Tk and Swing, especially given the very straightforward naming scheme. However, Qt adds some widgets that I found to be interestingly unique, such as:

- QDesignerWidgetBoxInterface – A widget specifically designed to manipulate the contents of *Qt Designer's* widget box.
- QDockWidget – A widget that can be docked in a QMainWindow, or floated on the desktop as a top-level window.
- QGLWidget – A widget specifically for the rendering of OpenGL graphics.
- QSplashScreen – A widget specifically for the display of an opening splash screen.
- ...

Layout Management

Layout management in QT is done in one of two ways, either through a Layout Widget, or through a subclass of the QLayout class. [3] There are three Layout Widgets, each of which is extremely simplistic. QHBox lays out its children in a horizontal row, QVBox lays out its children in a vertical row, and QGrid lays out its children in a grid of n rows, with a number of columns that you specify, with children being filled in row by row left to right.
QLayout is a much more powerful layout management system than the above widgets as they give the implementor much more control over the placement of widgets in their container. The downsides being, naturally, a higher complexity in implementation, and the fact that one must add their widget both to the QLayout ancestor being instantiated, and the containing widget. An example is the QGridLayout, which essentially is identical to Swing's GridLayout. A nice functionality, if one is so brave as to undertake it, is that QLayout can be extended to implement one's own custom layout. [3]

IDE/UIDE

I did not explore IDEs or UIDEs specifically made for Qt, and am unsure if they necessarily exist. The tutorials available on the TrollTech website are specifically written in such a way that one can easily enter the code by hand into text files, and this is the way I explored using Qt.

Example Code

The Following code snippet generates a window that contains a single button labeled "Hello World." [4]

```
#include <QApplication>
#include <QPushButton>

int main (int argc, char *argv[])
{
        QApplication app(argc, argv);
        QPushButton hello("Hello world!");
        hello.resize(100, 30);
        hello.show();
        return app.exec();
}
```

WxWidgets

WxWidgets is another cross platform C++ GUI library framework which was originally developed by the University of Edinburgh's Artificial Intelligence Applications Institute, and first made publically available in 1992. [6] Like Qt, all libraries are the ancestors of a single class in the libraries, wxObject, which resides in a library called WxBase upon which all other libraries also depend. [5]

Main Event Loop

Event handling in WxWidgets is approached in an interesting fashion. There is a class, wxEvtHandler, that handles events received from the Windowing System. What makes it interesting is that all Window classes in WxWidgets are ancestors of wxEvtHandler rather than using an instance of the class to handle the events it deals with. [6] wxEvtHandler works by invoking the method listed in the event table of an event pulled from the event queue on itself. This does introduce a bit of danger when multiple inheritance is used as the WxEvtHandler inheritance must come first for the this pointer to execute properly. The main event loop is essentially then the topmost Window class in the parent tree, and this is the wxApp, which is the class that is meant to represent the GUI application itself. It has methods to dispatch events in the queue and the ability to deal with any events not otherwise handled by other aspects of the application.

Events

Events in WxWidgets are any ancestor of the wxEvent class, [7] which in general can be divided between command and non-command events, and user and programmatically generated events. The former distinction is important as command events are passed recursively to the parents of the window which received the original event, whereas non-command events are not. The latter distinction is important as, save a few exceptions, only user generated events can generate a command event.
WxWidgets has a smaller general event set than Qt, which include:

- wxCommandEvent – the type of all control events
- wxEraseEvent – events that request Window erases
- wxKeyEvent – events that are keyboard presses, releases, etc
- wxMouseEvent – events that deal with either individual mouse events or all mouse events
- wxPaintEvent – events that request Window paintings
- wxUpdateUIEvent – the type of all user generated pseudo-events that require UI updates.

Like Qt, WxWidgets uses an int enumeration of types to identify events [7]. However, these ints are assigned to the event types at run-time, which allows for users to create custom events without risking an type id clash with either another event type.

Widgets

Widgets in wxWidget are interesting in their uniqueness as well. That uniqueness being that they are all ancestors of wxWindow [8], as wxWindow is specifically stated to be "the base class for all windows and represents any visible object on the screen." As a comparison, this would change the Java AWT/Swing inheritance tree such that everything would be a child of the AWT Frame class. However, wxWidgets provides the following basic widgets:

- wxButton
- wxCheckBox
- wxColourPickerCtrl
- wxComboBox
- wxDialog
- wxMenu
- wxMenuBar
- wxPanel

- wxRadioButton
- wxSashWindow
- wxScrollBar
- ...

It also provides the following more specialized widgets:

- wxDatePickerCtrl – a widget specifically for date picking.
- wxGLCanvas – a widget that contains a canvas for drawing OpenGL graphics.
- wxMediaCtrl – a widget that allows the display of media files.
- wxPrintDialog – a widget containing print and print setup dialogs.
- ...

Layout Management

Layouts in wxWidgets are represented by the wxSizer class [9], which in general communicates information on the minimal required size of individual subwindows, and are stretchable if the window is resized. One uses a Sizer by adding classes that inherit from wxWindow directly to the Sizer, and then the Sizer handles the layout of those entities that have become its children. WxSizer has five implementing predecessors that are available for use out of the box [9]:

- wxBoxSizer – a sizer that lays out its children either horizontally or vertically.
- wxStaticBoxSizer – a wxBoxSizer that is surrounded by a static framing.
- wxGridSizer – a sizer that lays out its children in a 2-dimensional grid, with each child being given the same size.
- wxFlexGridSizer – wxGridSizer with the additional power to individually calculate the space allocated to each child.
- CreateButtonSizer – a sizer that allows the creation of a standard button sizer in which buttons defined by passed flags are displayed.

While WxWidgets does not in the documentation reviewed describe the extension of wxSizer, any class that inherits from wxSizer should be a user defined Sizer class.

IDE/UIDE

As in the Qt section, all code and examples that were explored for wxWidgets were conducted and explored entirely using the nano editor and my command line.

Example Code

The following code snippet generates a window containing a label that announces "Hello World" along with a menu bar with some basic functionality [10].

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

class MyApp: public wxApp
{
    virtual bool OnInit();
};

class MyFrame: public wxFrame
{
public:

    MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size);

    void OnQuit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);

    DECLARE_EVENT_TABLE()
};

enum
{
    ID_Quit = 1,
    ID_About,
};

BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(ID_Quit, MyFrame::OnQuit)
    EVT_MENU(ID_About, MyFrame::OnAbout)
END_EVENT_TABLE()

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    MyFrame *frame = new MyFrame( "Hello World", wxPoint(50,50), wxSize(450,340) )
    frame->Show(TRUE);
    SetTopWindow(frame);
```

```
    return TRUE;
}

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
: wxFrame((wxFrame *)NULL, -1, title, pos, size)
{
    wxMenu *menuFile = new wxMenu;

    menuFile->Append( ID_About, "&About..." );
    menuFile->AppendSeparator();
    menuFile->Append( ID_Quit, "E&xit" );

    wxMenuBar *menuBar = new wxMenuBar;
    menuBar->Append( menuFile, "&File" );

    SetMenuBar( menuBar );

    CreateStatusBar();
    SetStatusText( "Welcome to wxWindows!" );
}

void MyFrame::OnQuit(wxCommandEvent& WXUNUSED(event))
{
    Close(TRUE);
}

void MyFrame::OnAbout(wxCommandEvent& WXUNUSED(event))
{
    wxMessageBox("This is a wxWindows Hello world sample",
        "About Hello World", wxOK | wxICON_INFORMATION, this);
}
```

Visual C++ UIDE

Visual C++ is a tool designed by Microsoft to facilitate the quick usage of their own special implementation of the C++ with a UIDE built in for good measure. The version of Visual C++ explored for the purposes of this document was Microsoft's Visual C++ Express 2005, which is a freely available download from Microsoft focused entirely on working on the .Net 2.0 platform through the C++ language. The Visual C++ system expressly uses access to the Microsoft Win-

dows windowing system, and the same event management and widgets that it has access to. Essentially what makes the Visual C++ UIDE unique from the above is that there isn't direct access to graphic libraries, and instead one uses the tools available to one to draw and alter widgets pulled from those libraries.
Naturally, this makes the creation of GUIs incredibly easy. All one needs to do is add a UI to their project, and then use a tab that brings up a list of all the widgets supported by the program. One then selects the widget they wish to draw, draw it on the UI, and then set the particular properties of that object. What this does under the hood from experimentation is add code to the GUI's associated header file that does the actual creation of those widgets, and then sets up an event handling method that one is able to edit.

Event Handling

Visual C++ uses what Microsoft calls the Unified Event Model [11] which seems to be designed around enabling all of the Visual Studio languages of using events with a minimal dependent knowledge. In the Unified Event Model, everything starts with an Event Source, which is an object that contains and defines events. Events are then methods within an Event Source that generate events. Events are recieved by Event Recievers, which is an object that recieves Events, which contain methods called Event Handlers, who have a sufficiently self-descriptive name.

References

[1] TrollTech Inc. , *QEvent Class Reference. Available At: http://doc.trolltech.com/4.2/qevent.html*

[2] TrollTech Inc. , *QWidget Class Reference. Available At: http://doc.trolltech.com/4.0/qwidget.html*

[3] TrollTech Inc. , *Layout Classes. Available At: http://doc.trolltech.com/qtopia2.1/html/layout.html*

[4] TrollTech Inc. , *Qt Tutorial 1 - Hello World!. Available At: http://doc.trolltech.com/4.2/tutorial-t1.html*

[5] Julian Smart, Robert Roebling, Vadim Zeitlin, Robin Dunn, et al. , *Libraries List. http://www.wxwidgets.org/manuals/stable/wx_librarieslist.html*

[6] Julian Smart, Robert Roebling, Vadim Zeitlin, Robin Dunn, et al. , *What is wxWidgets?. Available At: http://www.wxwidgets.org/manuals/stable/wx_whatis.html*

[7] Julian Smart, Robert Roebling, Vadim Zeitlin, Robin Dunn, et al. , *Event Handling Overview. Available At: http://www.wxwidgets.org/manuals/stablewx_eventhandlingoverview.html*

[8] Julian Smart, Robert Roebling, Vadim Zeitlin, Robin Dunn, et al. , *wxApp. Available At: http://www.wxwidgets.org/manuals/stable/wx_wxwindow.html*

[9] Julian Smart, Robert Roebling, Vadim Zeitlin, Robin Dunn, et al. , *Sizer overview. Available At: http://www.wxwidgets.org/manauls/2.8/wx_sizeroverview.html*

[10] Robert Roebling, *Hello World. Available At: http://www.wxwidgets.org/docs/tutorials/hello.htm*

[11] Microsoft Inc. , *Introduction to the Unified Event Model. Available At: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore/html/vcconintroductiontounifiedeventmodel.asp*