The XRCWidgets Package

Ryan Kelly (ryan@rfk.id.au)

July 13, 2006

1 Introduction

The XRCWidgets package is a Python extension to the popular wxWidgets library. It is designed to allow the rapid development of graphical applications by leveraging the dynamic run-type capabilities of Python and the XML-based resource specification scheme XRC.

1.1 Underlying Technologies

- wxWidgets is a cross-platform GUI toolkit written in C++
  http://www.wxwidgets.org/

- wxPython is a wxWidgets binding for the Python language
  http://www.wxPython.org

- XRC is a wxWidgets standard for describing the layout and content of a GUI using an XML file
  http://www.wxwidgets.org/manuals/2.4.2/wx478.htm

1.2 Purpose

The XRCWidgets framework has been designed with the primary goal of streamlining the rapid development of GUI applications using Python. The secondary goal is flexibility, so that everything that can be done in a normal wxPython application can be done using XRCWidgets. Other goals such as efficiency take lower precedence.

It is envisaged that XRCWidgets would make an excellent application-prototyping platform. An initial version of the application can be constructed using Python and XRCWidgets, and if final versions require greater efficiency than the toolkit can provide it is a simple matter to convert the XRC files into Python or C++ code.

1.3 Advantages

1.3.1 Rapid GUI Development

Using freely-available XRC editing programs, it is possible to develop a quality interface in a fraction of the time it would take to code it by hand. This interface can be saved into an XRC file and easily integrated with the rest of the application.

1.3.2 Declarative Event Handling

The XRCWidgets framework allows event handlers to be automatically connected by defining them as specially-named methods of the XRCWidget class. This saves the tedium of writing an event handling method and connecting it by hand, and allows a number of cross-platform issues to be resolved in a single location.

1.3.3 Separation of Layout from Code

Rapid GUI development is also possible using design applications that directly output Python or C++ code. However, it can be difficult to incorporate this code in an existing application and almost impossible to reverse-engineer the code for further editing.

By contrast, the use of an XRC file means that any design tool can be used as long as it understands the standard format. There is no tie-in to a particular development toolset.

1.3.4 Easy Conversion to Native Code

If extra efficiency is required, is is trivial to transform an XRC file into Python or C++ code that constructs the GUI natively.

1.3.5 Compatibility with Standard wxPython Code

All XRCWidget objects are subclasses of the standard wxPython objects, and behave in a compatible way. For example, an XRCPanel is identical to a wxPanel except that it has a collection of child widgets created and event handlers connected as it is initialised.

This allows XRCWidgets to mix with hand-coded wxPython widgets, and behavior that is not implemented by the XRCWidgets framework can be added using standard wxPython techniques.

1.3.6 Simple Re-sizing of GUI

Coding GUIs that look good when resized can be very tedious if done by hand. Since XRC is based on sizers for layout, resizing of widgets works automatically in most cases.

2 Classes Provided

The classes provided by the XRCWidgets framework are detailed below.

2.1 XRCWidgetsError

This class inherits from the python built-in Exception class, and is the base class for all exceptions that are thrown by XRCWidgets code. It behaves in the same way as the standard Exception class.

2.2 XRCWidget

The main class provided by the package, XRCWidget is a mix-in that provides all of the generic GUI-building and event-connecting functionality. It provides the following methods:

- getChild(cName): return a reference to the widget named <cName> in the XRC file

- createInChild(cName,toCreate,*args): takes a wxPython widget factory (such as a class) and creates an instance of it inside the named child widget.

- showInChild(cName,widget): displays <widget> inside of the named child widget

- replaceInChild(cName,widget): displays <widget> inside the named child widget, destroying any of its previous children

An XRCWidget subclass may have any number of methods named in the form "on_<child>_<action>" which will automatically be connected as event handlers. <child> must be the name of a child from the XRC file, and <action> must be a valid action that may be performed on that widget.

Actions may associate with different events depending on the type of the child widget. Valid actions include:

- change: Called when the contents of the widget have changed (eg change a text box's contents)

- activate: Called when the widget is activated by the user (eg click on a button)

- content: Called at creation time to obtain the content for the child widget. This may be used as a shortcut to using "replaceInChild" in the constructor

2.3   XRCPanel

XRCPanel inherits from XRCWidget and wxPanel, implementing the necessary functionality to initialise a wxPanel from an XRC resource file. It provides no additional methods.

2.4   XRCDialog

XRCDialog inherits from XRCWidget and wxDialog, implementing the necessary functionality to initialise a wxDialog from an XRC resource file. It provides no additional methods.

2.5   XRCFrame

XRCFrame inherits from XRCWidget and wxFrame, implementing the necessary functionality to initialise a wxFrame from an XRC resource file. It provides no additional methods.

2.6   XRCApp

XRCApp inherits from XRCFrame and is designed to provide a shortcut for specifying the main frame of an application. It provides the methods "MainLoop" and "EndMainLoop" mirroring those of the wxApp class. When created, it creates a private wxApp class and makes itself the top-level window for the application.

3   Tutorials

The following are a number of quick tutorials to get you started using the framework. The code for these tutorials can be found in the 'examles' directory of the source distribution.

3.1   The Basics

This section provides a quick tutorial for creating an appliction using the XRCWidgets framework. The application will consist of a single widget called 'SimpleApp', and will live in the file 'simple.py'. It will consist of a wxFrame with a text-box and a button, which prints a message to the terminal when the button is clicked. The frame will look something like this:
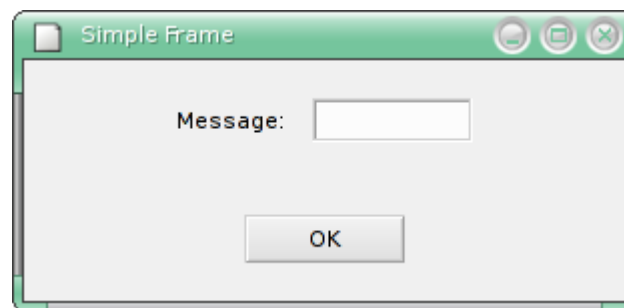
It will be necessary to create two files: 'simple.py' containing the python code, and 'simple.xrc' containing the XRC definitions.

3.1.1 Creating the XRC File

There are many ways to create an XRC file. The author recommends using wxGlade, a RAD GUI designer itself written in wxPython. It is available from http://wxglade.sourceforge.net/.

Launching wxGlade should result it an empty Application being displayed. First, set up the properties of the application to produce the desired output. In the 'Properties' window, select XRC as the output language and enter 'simple.xrc' as the output path.

Now to create the widget. From the main toolbar window, select the "Add a Frame" button. Make sure that the class of the frame is 'wxFrame' and click OK. In the Properties window set the name of the widget to "SimpleApp" - this is to correspond to the name of the class that is to be created.

Populate the frame with whatever contents you like, using sizers to lay them out appropriately. Consult the wxGlade tutorial (http://wxglade.sourceforge.net/tutorial.php) for more details. Make sure that you include a text control named "message" and a button named "ok".

When the frame is finished, selecte Generate Code from the File menu to produce the XRC file. You may also like to save the wxGlade Application so that it can be edited later. Alternately, wxGlade provides the tool "xrc2wxg" which can convert from the XRC file to a wxGlade project file.

3.1.2 Creating the Python Code

You should now have the file 'simple.xrc'. If you like, open it up in a text editor to see how the code is produced. If you are familiar with HTML or other forms of XML, you should be able to get an idea of what the contents mean.

Next, create the python file 'simple.py' using the following code:

```
from XRCWidgets import XRCApp
class SimpleApp(XRCApp):
    def on_message_change(self,msg):
        print "MESSAGE IS NOW:", msg.GetValue()
    def on_ok_activate(self,bttn):
        print self.getChild("message").GetValue()
```

This code is all that is required to make a functioning application. Notice that the defined methods meet the general format of "on_<child>_<action>" and so will be automatically connected as event handlers. The "on_message_change" method will be called whenever the text in the message box is changed, and "on_ok_activate" will be called whenever the button is clicked.

3.1.3 Testing the Widget

Once you have the files 'simple.py' and 'simple.xrc' ready, it is possible to put the widget into action. Launch a python shell and execute the following commands:

```
from simple import SimpleApp
app = SimpleApp()
app.MainLoop()
```

This code imports the widget's definition, creates the application and runs the event loop. The frame should appear and allow you to interact with it, printing messages to the console as the button is clicked or the message text is changed.

3.2 A more complicated Frame

This tutorial is yet to be completed. See the files 'menus.py' and menus.xrc' in the examples directory.

Quick Guide:

- Create a frame as usual, and select the 'Has MenuBar' option in its properties. Do *not* create a seperate MenuBar, this wont work.

- Edit the menus of the MenuBar to your liking. Ensure that you fill in the "name" field or the XML will not be generated correctly.