System Development with Python: DesktopGUIs: wxPython

Christopher Barker

UW Continuing Education

Nov 26, 2013

Table of Contents

Desktop GUIs: wxPython

Desktop GUIs

Traditional Graphical User Interface Applications

Run entirely on local machine – interactive, interface and logic code in one process

Advantages:

- Easier to write – all in one program
- Faster – data/interface direct communication
- Faster display: direct to screen (or even OpenGL, etc.)
- Runs without network
- Save/Manipulate local files
- Familiar install/start/stop/run, etc.

Python Options

Multiple GUI frameworks available:

- PyGTK
- PyQT / PySide
- TkInter
- wxPython
- PyGame
- Native GUIs: Cocoa (PyObjC), PythonWin
- Kivy for touchscreen (mobile) platforms
- Some more minor ones...

wxPython

Why wxPython?

- Python wrapper around C++ toolkit (wxWidget)
- wxWidgets is a wrapper around *native* toolkit:
    - Windows: Win32 (64)
    - OS-X: Cocoa
    - Linux: GTK
- Native look and feel
- License: (modified) LGPL

Legacy: it was the best option for me when I first needed something...

See http://www.wxpython.org for more information

Installing

wxPython is a big complicated build:
can't do `pip` or `easy_install`

Windows or OS-X:
use the binaries on
`http://wxpython.org/download.php`

Linux: use your system's package
NOTE: there are some issues with some packages:

- May be old version
- May use standard wx build – more crash prone! (some run-time checking turned off)

Versions

"Stable" version: 2.8.12.1
("stable" means stable API, not less likely to crash)

"Development" version: 2.9.4.0
(Under active development, API may change (but not much)

wx project very slow to do official releases – You probably want to use the development version: it's getting more attention

"Phoenix": next generation version: new bindings, Py3 support, etc.
– Still experimental

`http://wiki.wxpython.org/ProjectPhoenix`

Documentation

"Docs and Demos": download these!

"wxPython Demo" – run this!

Examples of every Widget available

Primary wx docs:

Written for C++, with Python notes...

`http://wxpython.org/onlinedocs.php`

This may help:

`http://wiki.wxpython.org/C%2B%2BGuideForwxPythoneers`

Semi-experimental Sphinx docs:

`http://xoomer.virgilio.it/infinity77/wxPython/`

The wxPython wiki: lots of good stuff here

`http://wiki.wxpython.org/`

Some starting points

How to learn wxPython

http://wiki.wxpython.org/How%20to%20Learn%20wxPython

wxPython Style Guide

http://wiki.wxpython.org/wxPython%20Style%20Guide

The wxpython-users mailing list is a great resource (and great community):

https://groups.google.com/forum/?fromgroups#!forum/wxpython-users

My own repository of samples:

https://github.com/PythonCHB/wxPythonDemos

Pythonic code:

Over the years, wxPython has grown a number of things to make it more "pythonic" – hide some of that C++ legacy

Properties:

The C++ classes are full of getters and setters:

```
wxTextCtrl::SetValue
wxTextCtrl::GetValue
```

These methods have been translated into properties for Python

```
MyTextCtrl.Value = some_string
another_string = wxTextCtrl.Value
```

(The Get/Set versions are still there, but it's klunkier code)

Pythonic code:

Other Python options: some specific wx types can be accessed with standard python types:

`wxPoint` — `(x,y)` (tuple)

`wx.List` — `[1,2,3]` (python list)

`wxSize` — `(w,h)` (tuple)

....

Using these makes your code cleaner and more pythonic

Event-Driven programming

On app startup, the .MainLoop() method is called.

The mainloop takes control – monitoring for events, then dispatching them

Events can come from the system, or user interaction: keyboard, mouse, etc.

All the work of your app is done in response to events

You only need to response to (Bind) the events you care about

Not so different than a web app, except events are finer-grained
(every mouse move, etc.)

wx.Window

Pretty much everything you see on the screen is a `wx.Window`

It is the superclass for all the “widgets”, “controls”, or whatever you want to call them

It is essentially a rectangle on the screen that catches events

You generally don’t use it by iteself, though you may derive from it to make a new widget

(Historical Note: `wxWidgets` was called `wxWindows` – until Microsoft threatened to sue them.)

wx.Window

Since everything is a `wx.Window`, it's good to know its methods and signature:

```
def __init__(parent,
             id=wx.ID_ANY,
             pos=wx.DefaultPosition,
             size=wx.DefaultSize,
             style=0,
             name=wx.PanelNameStr)
parent (wx.Window)
id (int)
pos (wx.Point)
size (wx.Size)
style (long)
name (string)
```

wx.Window

Methods types:

- Appearance: Colors, Fonts, Labels, Styles
- Geometry: Size, Position, IsShown, Move, etc
- Layout: Sizers, etc.
- Many others!

`http://xoomer.virgilio.it/infinity77/wxPython/Widgets/wx.Window.html`

Event-Driven programming

On app startup, the .MainLoop() method is called.

The mainloop takes control – monitoring for events, then dispatching them

Events can come from the system, or user interaction: keyboard, mouse, etc.

All the work of your app is done in response to events

You only need to response to (Bind) the events you care about

Not so different than a web app, except events are finer-grained
(every mouse move, etc.)

wx.App

Every wx app has a single wx.App instance:

```
app = wx.App(False)
frame = DemoFrame(None, title="Micro App")
frame.Show()
app.MainLoop()
```

(the `False` means: “don’t re-direct stdout to a Window”)
And you almost always start the ‘MainLoop‘

wx.Frame

wx.Frame is a “top level” Window: One with a title bar, min-max buttons,etc.

Most apps have a single wx.Frame – central interaction with the app.

This is where menu bars, etc are placed, and often the core GUI logic of the app.

```
class TestFrame(wx.Frame):
    def __init__(self, *args, **kwargs):
        kwargs.setdefault('title', "Simple test App")
        wx.Frame.__init__(self, *args, **kwargs)
```

demo: code\basic_app_1.py

Menus

A `wx.Frame` has a menu bar you can add items to:

```
# create the menu bar object
menuBar = wx.MenuBar()

# add a menu to it
fileMenu = wx.Menu()

# add an item to the menu
openMenuItem = fileMenu.Append(wx.ID_ANY, "&Open", "Ope
#bind a handler to the menu event
self.Bind(wx.EVT_MENU, self.onOpen, openMenuItem)

self.SetMenuBar(menuBar)
```

demo: `code\basic_app_2.py`

Event Handlers

Event handlers have a common signature:

```
def onOpen(self, evt=None):
    print "open menu selected"
    self.app_logic.file_open()
```

The second parameter is the `wx.Event` object that initiated the call – it holds information about the event that can be useful

I like to give the event parameter a default None, so the handler can be called from other parts of the code as well.

demo: `code\basic_app_2.py`

Common Dialogs

wxPython provides a number of common Dialogs. These wrap the native ones where possible for a native look and feel.

- `wx.MessageDialog`
- `wx.ColourDialog`
- `wx.FileDialog`
- `wx.PageSetupDialog`
- `wx.FontDialog`
- `wx.DirDialog`
- `wx.SingleChoiceDialog`
- `wx.TextEntryDialog`
- ...

These do pretty much what you'd expect...

wx.FileDialog

Example use of a common dialog: `wx.FileDialog`

```
dlg = wx.FileDialog(self,
                    message="Save file as ...",
                    defaultDir=os.getcwd(),
                    defaultFile="",
                    wildcard=wildcard,
                    style=wx.SAVE )
if dlg.ShowModal() == wx.ID_OK:
    path = dlg.GetPath()
else:
    print "The file dialog was canceled before anything was
dlg.Destroy()
```

example: `code/basic_app_3.py`

Basic Widgets

All the basic widgets (controls) you'd expect are there:

- Buttons
- TextCtrl (Text Control)
- Check Boxes
- List Box
- Combo Box
- Slider
- Spin Control
- ....

Way too many to list here!
See the docs and the Demo to find the one you need

Using a Control

A Button is about as simple as it gets

```
__init__(parent, id, label="", pos=wx.DefaultPosition, ...
```

Mostly the same as wx.Window, and other controls....

```
## add just a single button:
self.theButton = wx.Button(self, label="Push Me")
self.theButton.Bind(wx.EVT_BUTTON, self.onButton)

## and give it an event handler
def onButton(self, evt=None):
    print "You pushed the button!"
```

code: `code\basic_app_4.py`

wx.Panel

A `wx.Panel` is a `wx.Window` that you can put other controls on

It supplies nifty things like tab traversal, etc.

You *can* put controls right on a `wx.Frame` (we just did it), but a wx.Panel provided extra features, the "normal" look, and helps you organize and re-use your code

Mostly the same as wx.Window, and other controls....

wx.Panel

```
class ButtonPanel(wx.Panel):
    def __init__(self, *args, **kwargs):
        wx.Panel.__init__(self, *args, **kwargs)

        self.theButton = wx.Button(self, label="Push Me")
        self.theButton.Bind(wx.EVT_BUTTON, self.onButton)
    def onButton(self, evt=None):
        print "You pushed the button!"
```

And use it in the Frame:

```
        self.buttonPanel = ButtonPanel(self)
```

code: `code\basic_app_5.py`

Control Layout

With more than one control, you need to figure out how to place them and how big to make them

You may have noticed that `wx.Window` takes `pos` and `size` parameters

You may have also noticed that I didn't use them.

Why not?

Absolute Positioning

Absolute positioning:

Specifying the size and location of controls with pixel coordinates.

This is a serious pain to do!

Though it can be made a lot easier with GUI-building tools...

So why not?

Absolute Positioning

When you add or remove a control, the layout changes:
– recalculate all positions and sizes

When you change the text on a control the layout changes:
– recalculate all positions and sizes

When you try it on another platform the layout changes:
– recalculate all positions and sizes

When the user changes default font size, the layout changes:
– recalculate all positions and sizes

Sizers:

The alternative is "Sizers"

`wx.Sizer` is wx's system for automatically determining the size and location of controls

Instead of thinking in terms of what size and position a given control should be, you think in terms of how they relate to each other:

"I want a column of buttons all the same size along the left edge of the Panel"

Sizers capture that logic and compute the sizes for you

They will re-size things for you when anything changes – adding, removing, changing labels, re-sizing the Window, etc...

Sizers:

Sizers take a while to wrap your brain around...

But it's worth the learning curve.

Nice discussion here:
`http://wiki.wxpython.org/UsingSizers`

I have the graphic posted on the wall by my desk...

Sizer Example

The Basic BoxSizer
– Lays out a row or column of controls...

```
Sizer.Add( window, proportion, flag, border )
        ## do the layout
        S = wx.BoxSizer(wx.VERTICAL)

        S.Add(theButton1, 0, wx.GROW | wx.ALL, 4)
        S.Add(theButton2, 0, wx.GROW | wx.ALL, 4)

        self.SetSizerAndFit(S)
```

code: `code\basic_app_6.py`

Nested Sizers

How do I get them centered both ways?
– Nest a vertical inside a horizonal
– And add stretchable spacers...

```
buttonSizer = wx.BoxSizer(wx.VERTICAL)

buttonSizer.Add(theButton1, 0, wx.GROW | wx.ALL, 4)
buttonSizer.Add(theButton2, 0, wx.GROW | wx.ALL, 4)

mainSizer = wx.BoxSizer(wx.HORIZONTAL)
mainSizer.Add((1,1), 1)    # stretchable space
mainSizer.Add(buttonSizer, 0, wx.ALIGN_CENTER) # the sizer
mainSizer.Add((1,1), 1)    # stretchable space
```

Widget Inspection Tool

How do I keep all this straight?

The Widget Inspection Tool (WIT) is very handy:

```
app = TestApp(False)
## set up the WIT -- to help debug sizers
import wx.lib.inspection
wx.lib.inspection.InspectionTool().Show()
app.MainLoop()
```

(you can also bring it up from a menu event, or...)

code: `code\basic_app_7.py`

Other Sizers

Sizers for laying out stuff in grids...

`wx.GridSizer`

`wx.FlexGridSizer`

`wx.GridBagSizer`

(you can do it all with a GridBagSizer)

See the docs for info.

Hierarchies...

wxPython has multiple independent hierarchies ...

The nested parent-child relationship:

- every `wx.Window` has a parent
- every `wx.Window` has zero or more children

The class Hierarchy

- sub classes of `wx.Window`
- classes with instances as attributes

The Layout Hierarchy

- Sizers within Sizers...
- Arbitrarily deep.

Each of these takes care of different concerns:
confusing but powerful

Accessing inputs

Much of the point of a GUI is to collect data from the user.

So you need to be able to access what s/he has input

```
## add a text control:
self.textControl = wx.TextCtrl(self)

def onGetData(self, evt=None):
    print "get data button pressed"
    contents = self.textControl.Value
    print "the contents are:", contents
```

Most controls have a `.Value` property

Setting Values

You also want to display data...

So you need to be able to set the values, too:

```
## and another text control:
self.outTextControl = wx.TextCtrl(self,
                                  style=wx.TE_READONLY)

def onGetData(self, evt=None):
    self.outTextControl.Value = self.inTextControl.Value
```

You can set the `.Value` property too...

example: `code\basic_app8.py`

Code-generated GUIs...

You shouldn't write the same repetitive code for a GUI..

You may need to build a GUI to match data at run time.

Lots of ways to do that with wxPython – Sizers help a lot.

Try to do it whenever you find yourself writing repetitive code...

The key is how to do the event Binding

```
def OnButton(self, evt):
    label = evt.GetEventObject().GetLabel()

    do_somethign_with_label(label)
```

example: `code/CalculatorDemo.py`

Code-generated GUIs...

The "lambda trick"

– a way to pass custom data to an event handler:

The key is how to do the event Binding

```
for name in ["first", "second", "third"]:
  btn = wx.Button(self, label=name)
  btn.Bind(wx.EVT_BUTTON,
          lambda evt, n=name: self.OnButton(evt, n) )
....
def OnButton(self, Event, name):
    print "In OnButton:", name
```

http://wiki.wxpython.org/Passing%20Arguments%20to%20Callbacks

Long Running Tasks

The UI is locked up while an event is being handled

So you want all event handlers to run fast.

But what if there is significant work to do?

Enter: threading and multi-processing

But: wxPython is not thread-safe: almost all wx methods must be called from within the same thread.

Thread-safe operations: Creating and Posting Events

CallAfter

Easiest way to communicate with threads: `wx.CallAfter`

Puts an event on the event stack, calls the designated function or method when the stack is cleared:

```
wx.CallAfter(function_to_call, *args, **kwargs)

# *args, **kwargs are passed on to FunctionToCall
```

(see also: `wx.CallLater()`)

`http://wiki.wxpython.org/LongRunningTasks`

BILS

Browser **I**nterface, **L**ocal **S**erver

Web app: Server runs on local machine

Browser is the interface – but all running local

Can wrap the Browser window in a desktop app: Chrome Embedded Framework, wxWebkit, etc.

Good way to get both a web app and desktop app with one codebase

Example: Cameo Chemicals

(PyCon 2009: Browser Interface, Local Server Application)

LAB

Make a very simple address book app:

1. Really basic data model is in `address_book_data.py`
2. Make a form to edit an entry – subclass a `wx.Panel` (`a_book_form.py`)
3. Put the form in a `wx.Frame` – the frame should have a way to switch between entries
4. Add file–save and file–open menus to the frame
5. Add some validation, better layout, etc....

`code\address_book\`