An Introduction to wxPython

Robin Dunn

O’Reilly Open Source Convention
July 24–28, 2006
Presentation Overview

- Your Instructor
- Introduction
- GUI Basics
- wxPython Fundamentals
- wxPython Widgets
- Event Handling
- Window Layout
- Device Contexts
- Tools
Robin Dunn

• Creator and maintainer of wxPython
• Programming for 25+ years, started in Junior High
• Working on wxPython for about 9 years
  – More or less full-time for 3+ years
• Member of the wxWidgets core team
• Currently on contract with OSAF
• Coauthor of *wxPython in Action*
Why wxPython?

• wxPython is an **open source** GUI toolkit based on the wxWidgets (formerly wxWindows) library

• Designed to be cross-platform and supports most Unix/Linux platforms, MS Windows and Mac OS X

• Uses **native widgets** wherever possible to preserve native Look and Feel.

• Extensive sample programs, helpful and capable community

• Mature, well established projects.
  – wxWidgets: 1992
  – wxPython: 1996
Basic architecture

Proxy classes →

wxPython Extension Modules

wxWidgets Toolkit

Platform GUI

Operating System

wxPython: Cross Platform GUI Toolkit
Partial class hierarchy

wx.Object

wx.EvtHandler

wx.Window

wx.TopLevelWindow

wx.Frame

wx.Dialog

wx.ScrolledWindow

wx.Panel

wx.Control
Getting started with wxPython

• Choose an installer
  – Windows *.exe installers, Linux RPMs or OSX *.dmg
  – Can be built from source with a few prerequisites

• Which version of Python do you use?
  – 2.3, 2.4, 2.5

• Unicode or ANSI?
  – Unicode builds available on all platforms, but be careful with Win9x/ME
  – ANSI available for platforms, but may be phased out soon.
Getting started with wxPython

• Choose an editor or development environment:
  – Boa Constructor
  – WingIDE
  – SPE
  – SCiTE
  – Emacs, vi, etc.

• It’s just plain text, so any ordinary editor and command line will do.
Getting started with wxPython

• Ready, set, go!
• The wxPython Demo is a great way to learn about the capabilities of the toolkit.
Getting started with wxPython

wxPython: Cross Platform GUI Toolkit
Getting started with wxPython
Getting started with wxPython

wxPython: Cross Platform GUI Toolkit
Getting started with wxPython

wxPython: Cross Platform GUI Toolkit
Demo time...
GUI Basics

• GUI’s are composed of a collection of windows, or widgets
  – Some widgets are top-level windows that are managed by the OS
  – Some are contained in other widgets

• You can think of a window as a
  \textit{tree of graphical components}

• Before you can display a window you must
  – Create the component tree, and (optionally) ...
  – Associate events with particular objects and actions
GUI Basics

• GUI applications are *event driven*
  – The application spends most of its time waiting for something to happen, such as a keystroke, or mouse movement.
  – When that something (*the event*) happens, information about it is collected and sent to a handler.
  – Events are dispatched asynchronously
    • meaning they can happen in any order
GUI Basics

• Many events are a direct result of user actions
  – Left-click on a button
  – Select a menu item
  – Drag an item from one panel to another
    • That would actually be a sequence of events

• Other events are raised by the system
  – Timer countdown expires
  – An obscured part of a window is exposed
GUI Basics
Hello World!

# ex01.py
import wx

class App(wx.App):
    def OnInit(self):
        frame = wx.Frame(parent=None, title="Hello World! 1")
        frame.Show()
        return True

app = App()
app.MainLoop()
wxPython Fundamentals

• Every application needs an instance of the wx.App class
  – Some parts of the C++ library are not initialized until the app is created, so it must be done before most other things.
  – APIs for starting and stopping the application
  – Provides the central *event loop* and dispatches events to handlers
  – Other per-application functionality

• Traditionally, you subclass wx.App and override OnInit for creating the initial application widgets
  – Not strictly needed any longer
  – wx.App can be used without subclassing
  – But it often still makes sense for design purposes
Hello World!

# ex02.py
import wx

app = wx.App()
frame = wx.Frame(parent=None, title="Hello World! 2")
frame.Show()
app.MainLoop()
wxPython Fundamentals

• wx.App can redirect standard output
  – Sends print statements and writes to sys.stdout or sys.stderr to a window or a file
  – An easy way to view status messages or tracebacks
  – Controlled by parameters to wx.App.__init__
Hello World!

# ex03.py
import wx

class Frame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, parent=None, title="Hello World! 3")
        b1 = wx.Button(self, label="Hello", pos=(20,20))
        b2 = wx.Button(self, label="World", pos=(20,60))
        self.Bind(wx.EVT_BUTTON, self.OnHelloWorld)

    def OnHelloWorld(self, self, evt):
        print "Hello World!"

app = wx.App(redirec=True)
Frame().Show()
app.MainLoop()
Hello World!

# ex03.py
import wx

class Frame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, parent=None, title="Hello World! 3")
        b1 = wx.Button(self, label="Hello", pos=(20,20))
        b2 = wx.Button(self, label="World", pos=(20,60))
        self.Bind(wx.EVT_BUTTON, self.OnHelloWorld)

    def OnHelloWorld(self, evt):
        print "Hello World!"

app = wx.App(redirect=True)
Frame().Show()
app.MainLoop()
wxPython Fundamentals

• Every application needs a `wx.App` and one or more top level windows
• Window/Widget classes can be used directly, but you will often subclass them to add-in your application’s functionality
• Events are signals from the user or the system that your application is interested in.
• Events are delivered to event handler functions (usually members of the derived widget classes)
• Events can happen in any order
wxPython widgets: top level windows

- **wx.Frame**
  - A container for other windows.
  - Can automatically manage a MenuBar, ToolBar, and a StatusBar.

- **wx.Dialog**
  - For Modal or Modeless dialog boxes.

- **wx.MiniFrame**
  - Good for floating tool pallets, etc.

- **wx.MDIParentFrame, wx.MDIChildFrame**
  - Multiple Document Interface
wxPython widgets: common dialogs

• All standard Windows common dialogs:
  – Color, Directory, File,
  – Font, PageSetup, Print,
  – Message, Progress,
  – FindReplace, etc.

• For other platforms either native dialogs are used, or suitable recreations in wxWidgets are provided.
wxPython widgets: common dialogs
wxPython widgets: basic windows

• `wx.Window`
  - General purpose window.

• `wx.Panel`
  - Can do tab-traversal of controls.
  - Uses standard system color for the background.

• `wx.ScrolledWindow`
  - Manages its own scrollbars and scrolling of client area.
  - Transforms coordinates based on scrollbar positions.
wxPython widgets

- `wx.SplitterWindow`
  - Can be split vertically or horizontally.
  - Draggable sash for redistributing the space between sub-windows.
wxPython widgets

• wx.grid.Grid
  – Table or spreadsheet-like capabilities.
  – Editors, Renderers, Tables (the data provider) can all be customized and “plugged in”.
wxPython widgets

<table>
<thead>
<tr>
<th>Custom</th>
<th>column</th>
<th>labels</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>First cell</td>
<td>Another cell</td>
<td>Yet another cell</td>
<td>Limited text</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>123.34</td>
<td></td>
<td></td>
<td></td>
<td>You can veto editing this cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This default cell will overflow into neighboring cells, but not if you turn overflow off.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This cell is set to span 3 rows and 3 columns</td>
</tr>
</tbody>
</table>
wxPython widgets

- `wx.StatusBar`

- `wx.ToolBar`
wxPython widgets

• `wx.Notebook`
  – Manages multiple windows with tabs.
  – Tabs can be on any side of the notebook that the platform supports.
wxPython widgets

• **wx.html.HtmlWindow**
  - Capable of parsing and rendering most simple HTML tags.
  - Custom Tag Handlers can change or add to how HTML is rendered.

```html
<wxp class="wx.Button">
  <param name="label" value="Okay">
  <param name="id" value="wxID_OK">
</wxp>
```
wxPython widgets

- `wx.html.HtmlWindow`
wxPython widgets: controls

- `wx.Button`, `wx.BitmapButton`
- `wx.RadioBox`, `wx.RadioButton`
- `wx.CheckBox`
- `wx.Choice`
- `wx.ComboBox`
- `wx.SpinButton`
wxPython widgets: controls

- `wx.ToggleButton`
- `wx.gizmosEditableListBox`
- `wx.lib.masked.TextCtrl`
- `wx.calendar.CalendarCtrl`
- `wx.lib.masked.TimeCtrl`
wxPython widgets: controls

- wx.TextCtrl
  - Password masking, multi-line with or without word-wrap, simple attributes, etc.
wxPython widgets: controls

- wx.ListBox
- wx.CheckListBox
- wx.Gauge
- wx.Slider
- wx.StaticBox
wxPython widgets: controls

- **wx.ListCtrl**
  - Supports list, icon, small icon, report views.
  - Virtual mode, where data items are provided by overloaded methods.

```
<table>
<thead>
<tr>
<th>Artist</th>
<th>Title</th>
<th>Genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad English</td>
<td>The Price Of Love</td>
<td>Rock</td>
</tr>
<tr>
<td>DNA featuring Suzanne Vega</td>
<td>Tom's Diner</td>
<td>Rock</td>
</tr>
<tr>
<td>George Michael</td>
<td>Praying For Time</td>
<td>Rock</td>
</tr>
<tr>
<td>Gloria Estefan</td>
<td>Here We Are</td>
<td>Rock</td>
</tr>
<tr>
<td>Linda Ronstadt</td>
<td>Don't Know Much</td>
<td>Rock</td>
</tr>
<tr>
<td>Michael Bolton</td>
<td>How Am I Supposed To Live Without You</td>
<td>Blues</td>
</tr>
<tr>
<td>Paula Young</td>
<td>Oh Girl</td>
<td>Rock</td>
</tr>
<tr>
<td>Paula Abdul</td>
<td>Opposites Attract</td>
<td>Rock</td>
</tr>
<tr>
<td>Richard Marx</td>
<td>Should've Known Better</td>
<td>Rock</td>
</tr>
<tr>
<td>Rod Stewart</td>
<td>Forever Young</td>
<td>Rock</td>
</tr>
<tr>
<td>Roxette</td>
<td>Dangerous</td>
<td>Rock</td>
</tr>
<tr>
<td>Sheena Easton</td>
<td>The Lover In Me</td>
<td>Rock</td>
</tr>
<tr>
<td>Sinead O'Connor</td>
<td>Nothing Compares 2 U</td>
<td>Rock</td>
</tr>
<tr>
<td>Stevie B.</td>
<td>Because I Love You</td>
<td>Rock</td>
</tr>
<tr>
<td>Taylor Dayne</td>
<td>Love Will Lead You Back</td>
<td>Rock</td>
</tr>
<tr>
<td>The Bangles</td>
<td>Eternal Flame</td>
<td>Rock</td>
</tr>
</tbody>
</table>
```
wxPython widgets: controls

- `wx.TreeCtrl`
  - Supports images for various node states.
  - Can be virtualized by delaying the adding of child items until the parent is expanded.
**wxPython widgets: controls**

- `wx.gizmos.TreeListCtrl`

<table>
<thead>
<tr>
<th>Main column</th>
<th>Column 1</th>
<th>Column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Root item</td>
<td>col 1 root</td>
<td>col 2 root</td>
</tr>
<tr>
<td>Item 0</td>
<td>Item 0(c1)</td>
<td>Item 0(c2)</td>
</tr>
<tr>
<td>Item 1</td>
<td>Item 1(c1)</td>
<td>Item 1(c2)</td>
</tr>
<tr>
<td>Item 2</td>
<td>Item 2(c1)</td>
<td>Item 2(c2)</td>
</tr>
<tr>
<td>Item 3</td>
<td>Item 3(c1)</td>
<td>Item 3(c2)</td>
</tr>
<tr>
<td>Item 4</td>
<td>Item 4(c1)</td>
<td>Item 4(c2)</td>
</tr>
<tr>
<td>Item 4-a</td>
<td>Item 4-a(c1)</td>
<td>Item 4-a(c2)</td>
</tr>
<tr>
<td>Item 4-b</td>
<td>Item 4-b(c1)</td>
<td>Item 4-b(c2)</td>
</tr>
<tr>
<td>Item 4-b-0</td>
<td>Item 4-b-0(c1)</td>
<td>Item 4-b-0(c2)</td>
</tr>
<tr>
<td>Item 4-b-1</td>
<td>Item 4-b-1(c1)</td>
<td>Item 4-b-1(c2)</td>
</tr>
<tr>
<td>Item 4-b-2</td>
<td>Item 4-b-2(c1)</td>
<td>Item 4-b-2(c2)</td>
</tr>
<tr>
<td>Item 4-b-3</td>
<td>Item 4-b-3(c1)</td>
<td>Item 4-b-3(c2)</td>
</tr>
<tr>
<td>Item 4-b-4</td>
<td>Item 4-b-4(c1)</td>
<td>Item 4-b-4(c2)</td>
</tr>
<tr>
<td>Item 4-c</td>
<td>Item 4-c(c1)</td>
<td>Item 4-c(c2)</td>
</tr>
<tr>
<td>Item 4-d</td>
<td>Item 4-d(c1)</td>
<td>Item 4-d(c2)</td>
</tr>
<tr>
<td>Item 4-e</td>
<td>Item 4-e(c1)</td>
<td>Item 4-e(c2)</td>
</tr>
<tr>
<td>Item 5</td>
<td>Item 5(c1)</td>
<td>Item 5(c2)</td>
</tr>
<tr>
<td>Item 6</td>
<td>Item 6(c1)</td>
<td>Item 6(c2)</td>
</tr>
</tbody>
</table>
wxPython widgets

• wx.stcStyledTextCtrl
  – (wx port of Scintilla)

```python
# vim: set et:sw=4:ts=4:
# Name: Main.py
# Purpose: Testing lots of stuff, controls, window types, etc.
# Author: Robin Dunn
# Created: A long time ago, in a galaxy far, far away...
# RCS-ID: $Id: Main.py,v 1.76.2.29 2003/05/23 16:47:49 RD Exp$
# Copyright: (c) 1999 by Total Control Software
# Licence: wxWindows license

import sys, os, time
from wxPython.wx import *
from wxPython.html import wxHtmlWindow

import images
```
wxPython widgets

• And many others…
Event Handling

• Most, if not all, GUI systems and toolkits are designed to be event driven, meaning that the main flow of your program is not sequential from beginning to end.

• When something happens that is of interest to you (an event), the system or toolkit calls a bit of your code that deals with that event (event handler).

• When your event handler finishes, control returns to the “main loop” and your program waits for the next event.

• While one event handler is running all others are blocked, so don’t do things that take a “long time” to complete.
Event handling

wxPython: Cross Platform GUI Toolkit
Event Handling

• Various event-handling models:
• **Callbacks**: Standalone functions associated with an event by calling a toolkit function. There are encapsulation problems.
• **Message based**: Messages sent to windows for controlling behavior, or for events.
• **Virtual methods**: One for each type of event. Solves encapsulation, but leads to clutter, inflexible classes, and many derived classes just to handle an event differently.
• **Static event tables**: Events are associated with classes and methods at compile time via a table. When the event occurs the tables are searched for a match and the method is invoked.
Event Handling

- **wxPython uses Dynamic Event Tables**
  - Built at run-time.
  - Events can be “bound” to any callable object that will serve as the Event Handler:
    - any method of the class receiving the event, or other classes
    - standalone functions
    - any object with a __call__ method
  - Handlers are connected to events with a set of binder objects:
    - wx.EVT_MENU
    - wx.EVT_PAINT
    - wx.EVT_SIZE
    - etc.
Event Handling

• The connection, or *binding*, between event and handler is made with the Bind method

```python
def Bind(self, event, handler, source=None, id=wx.ID_ANY, id2=wx.ID_ANY):
    self.Bind(wx.EVT_BUTTON, self.OnButton, theBtn)
    or
    theBtn.Bind(wx.EVT_BUTTON, self.OnButton)
```
Event Handling

• Each handler is passed an event object when called.
  – Contains information about the event

• Two classifications of event objects:
  – Classes derived from **wx.Event**
    • Events that only make sense for the window where the event took place, such as `wx.PaintEvent`, `wx.KeyEvent`, `wx.SizeEvent`, etc.

  – Classes derived from **wx.CommandEvent**
    • Events that may be of interest for any object up the “containment hierarchy,” such as `wx.MenuEvent`, `wx.NotebookEvent`, `wx.ListEvent`, etc.
Event handling

Event triggered → Triggering object → Enabled? → Should Propagate?

Yes → Has matching handler?

Yes → Event handler

No → Propagate?

Yes → Is this the App?

Yes → Container

No → Container

No → Done

Skip?
In search of Event Handlers...

```python
self.Bind(wx.EVT_BUTTON, 
    self.OnClick, 
    self.button)
```

def OnClick(self, evt):
    print "click"

`ButtonClick`
In search of Event Handlers...

```python
self.Bind(wx.EVT_BUTTON,
         self.OnClick,
         self.button)

self.button.Bind(
                  wx.EVT_LEFT_DOWN,
                  self.OnMouseDown)
```

![Diagram of event handling in wxPython](attachment:image.png)

- `def OnClick(self, evt): `
  - print “click”

- `def OnMouseDown(self, evt): `
  - print “I got it first!”
  - evt.Skip()
Code break...
Organizing your layout

• There are various ways to do layout:
  – Brute force
    • All widgets are positioned and sized pixel by pixel.
    • Has to be redone in every EVT_SIZE event.
    • Painful, cross-platform issues.
  – Layout Constraints
    • Powerful, but complex and verbose.
    • Deals with the relationships between widgets.
    • See the docs and demo for more details.
  – Sizers
    • Not as flexible or complex, but powerful enough.
    • Worth the pain.
Organizing your layout

• Sizers
  – Similar to LayoutManagers in Java.
  – Not as flexible as LayoutConstraints, but much simpler, once you get over the hump.
  – Relationships defined by containment within sizers or nested sizers.
  – All items (widgets or nested sizers) added to a Sizer are laid out by a specific algorithm determined by the class of sizer.
  – An item’s position within its allotted space is also controllable.
    • Empty space on borders
    • Alignment
  – You need to be able to think visually both top-down and bottom-up to capture your design
wx.BoxSizer

Horizontal BoxSizer

Horizontal layout:
- One
- Two
- Three
- Four

Vertical layout:
- One
- Two
- Three
- Four

Proportional layout:
- One
- Two
- Three
- Four

- Gets 1/3 of the free space
- Gets 2/3 of the free space
wx.StaticBoxSizer
wx.BoxSizer

box = wx.BoxSizer(wx.VERTICAL)
box.Add(wx.Button(win, -1, "one"), 0, wx.EXPAND)
box2 = wx.BoxSizer(wx.HORIZONTAL)
box2.Add(wx.Button(win, -1, "two"), 0, wx.EXPAND)
box2.Add(wx.Button(win, -1, "three"), 0, wx.EXPAND)
box2.Add(wx.Button(win, -1, "four"), 0, wx.EXPAND)
box2.Add(wx.Button(win, -1, "five"), 0, wx.EXPAND)

box3 = wx.BoxSizer(wx.VERTICAL)
box3.Add(wx.Button(win, -1, "six"), 0, wx.EXPAND)
box3.Add(wx.Button(win, -1, "seven"), 2, wx.EXPAND)
box3.Add(wx.Button(win, -1, "eight"), 1, wx.EXPAND)
box3.Add(wx.Button(win, -1, "nine"), 1, wx.EXPAND)

box2.Add(box3, 1, wx.EXPAND)
box.Add(box2, 1, wx.EXPAND)
box.Add(wx.Button(win, -1, "ten"), 0, wx.EXPAND)
wx.GridSizer
wx.GridBagSizer
Sizers in the Real World

- Can you see how to get here from there?
Sizers in the Real World

Vertical Box Sizer
Sizers in the Real World

Vertical Box Sizer

Flex Grid Sizer
Sizers in the Real World

- Vertical Box Sizer
- Flex Grid Sizer
- Horizontal Box Sizer
Sizers in the Real World

- Vertical Box Sizer
- Flex Grid Sizer
- Horizontal Box Sizer

wxPython: Cross Platform GUI Toolkit
Code break...
wxPython: Cross Platform GUI Toolkit

Drawing

• A wx.DC is a *device context* onto which graphics and text can be drawn.

• Represents a number of output devices in a generic way:
  – windows
  – printers
  – bitmaps
  – the whole screen

• The same code may be used to draw on different devices.
Drawing

- DC’s have many drawing primitives:

- And work with GDI objects:
Code break...
Tools: PyCrust

- Interactive Python Shell
- 100% Python
- Part of wxPython
- Standalone App
- Embeddable Components
```python
>>> import wx
>>> f = wx.Frame(None, -1, "Hello World")
>>> p = wx.Panel(f)
>>> b = wx.Button(p, -1, "Click me", (10,10))
```

Show(bool show=True) -> bool

Shows or hides the window. You may need to call Raise for a top level window if you want to bring it to top, although this is not needed if Show is called immediately after the frame creation. Returns True if the window has been shown or hidden or False if nothing was done because it already was in the requested state.

**Docstring:**

"""Call the specified function after the current and pending event handlers have been completed. This is also good for making GUI method calls from non-GUI threads."""
Tools: Embedding PyCrust components

# ex04.py
import wx
import wx.py
...
class MyApp(wx.App):
    def OnInit(self):
        frame = MyFrame()
        frame.Show()

        shell = wx.py.shell.ShellFrame(
            frame, locals={ 'wx': wx, 'frame': frame})
        shell.Show()

        frame.Raise()
        return True
PyCrust demo...
Tools: XRCed

- XRC is an XML based resource format
  - Used for specifying the content and layout of
    - Panels
    - Frames
    - Dialogs
    - Menus
    - Toolbars
  - Can be dynamically loaded at runtime, creating all the specified widgets
Tools: XRCed

wxPython: Cross Platform GUI Toolkit
Other tools

• wxDesigner
• wxGlade
• Boa Constructor
• DialogBlocks
• SPE
• WingIDE
• Dabo
• And many others…
Questions?
Other resources

- wxPython website: http://wxPython.org
- wxPyWiki: http://wiki.wxPython.org
- Mailists: wxPython-users, wx-users
- wxWidgets website: http://wxWidgets.org
- *wxPython in Action*