The Clubber's Guide to Rapid Application Development with GNOME and Python

Davyd Madeley
GNOME Project

A Little Bit About Me

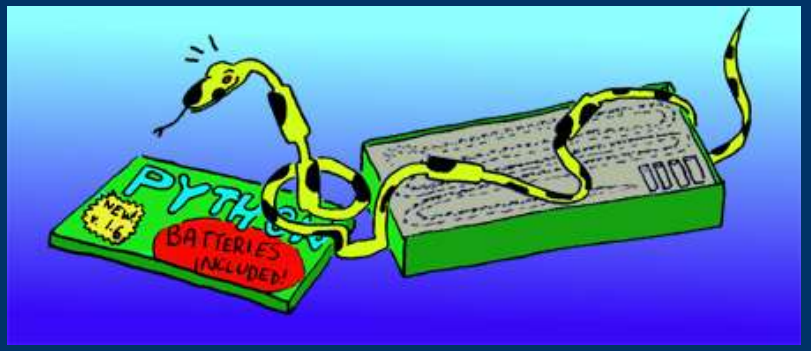
- Been using GNU/Linux and GNOME for a long while now
- Current GNOME Applets Maintainer
- One of the newer maintainers
  - Done two release cycles now
  - Experienced enough to know what I'm doing
  - New enough to still related to people outside the project
- I really like Python, it was the first Open Source language I learnt (I didn't even know C!)

About GLib and GTK+

- Object-orientated
- Libraries are written in C
- Platform independent
- Lots of useful data structures are available to us
- Helps programmer with memory management

http://www.gtk.org/

About Python



- Object-orientated
- Can interface with libraries written in C
- Platform independent
- Implements lots of useful data structures for us
- Built in memory management

http://www.python.org/

PyGTK

- Implements all of the GTK+ Widgets
- Does not implement GLib structures (blessing and a curse)
- Almost as fast as an application written in C (we benefit from GTK+ being written in C)
- Indistinguishable from a GTK+ app written in C
- LGPL licensed (like GTK+)
- Part of the official GNOME “Bindings” release
- There are currently no PyGTK applications in “Desktop”... yet.

http://www.pygtk.org/

Programming GTK+

- Uses a “main loop”
  - Don't ever block the main loop!
- Asynchronous callbacks - “signals”
- Widgets are laid out using “box packing”
  - libglade assists in language independent layout
- Everything is a GObject (except some simple data structures)
- Threaded programming is possible, and sometimes required, however be aware, not all libraries are threadsafe!

A PyGTK Program

```
1  #!/usr/bin/python
2
3  import gtk
4
5  def do_exit (window):                    # this is a signal handler
6          gtk.main_quit ()                 #  - we want to stop the main loop
7
8  window = gtk.Window ()                   # create a "Window" widget
9  label = gtk.Label ("Hello World")        # create a "Label" widget
10
11 window.add (label)                       # pack the Label in the Window
12
13 window.connect ("destroy", do_exit)      # handle the "destroy" signal with the
14                                          # do_exit() call
15
16 window.show_all ()                       # put all widgets in the tree into the
17                                          # show state
18
19 gtk.main ()                             # start the main loop
```

Adding More Signals

```
1  #!/usr/bin/python
2
3  import gtk
4
5  def do_exit (window):
6          gtk.main_quit ()
7
8  def do_click (button):
9          gtk.gdk.beep ()                 # beep at the user
10
11 window = gtk.Window ()
12 button = gtk.Button (stock=gtk.STOCK_YES)       # create a "Button" from stock
13
14 window.add (button)
15
16 button.connect ("clicked", do_click)     # handle the "clicked" signal with the
17                                          # do_click() call
18
19 window.connect ("destroy", do_exit)      # handle the "destroy" signal with the
20                                          # do_exit() call
21
22 window.show_all ()
23
24 gtk.main ()
```

More on Signals

- Signals for everything!
  - All GObjects emit signals, not just GTK+ widgets
  - Signals are inherited from ancestor classes
  - Examples are “notify”, “clicked”, “focus-in”, “change-background”, “activate”, “toggled”, “destroy”
- Full list of signals available for a GObject (widgets are GObjects too) available in the API documentation.
- API gives prototype required for callback function
  - `def callback(entry, delete_type, count, user_param1, ...)`

What is this user_param1 thing?

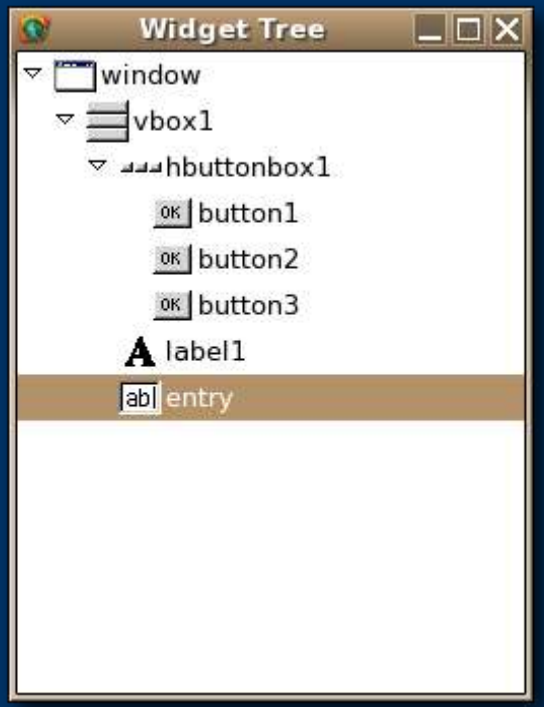
- Commonly referred to as simply “user_data”
- It is a pointer to some data structure (traditionally a struct) that will be useful inside the callback.
- Of course, this is Python, so we can pass whatever we like!!
- Probably the most useful thing to pass is a class containing the running program state (rather then make this global).
  - we don't always need this in Python, as you'll see later

Using libglade

- libglade is an XML format for defining GTK+ interfaces.
- It is also a library for reading that XML format and allowing those interfaces to be used in actual applications.
- These files can be produced in Glade, or recently in Gazpacho (a PyGTK app)
- No more clunky interface code required! Plus, now people can come along and edit/fix your interfaces with ease

Using libglade :: an example

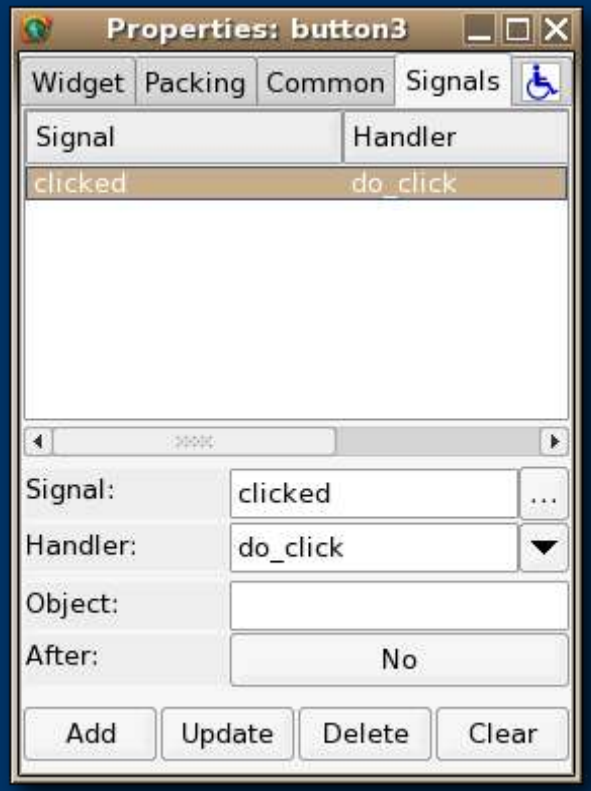
```
1  #!/usr/bin/python
2
3  import gtk
4  import gtk.glade                          # libglade support
5
6  def do_exit (window):
7          gtk.main_quit ()
8
9  def do_click (button, user_data):
10         print user_data
11         entry.set_text (button.get_label ())
12
13
14 xml = gtk.glade.XML ('libglade.glade', None, None) # load glade file
15
16 window = xml.get_widget ('window')       # get widgets from Glade
17 button1 = xml.get_widget ('button1')
18 button2 = xml.get_widget ('button2')
19 button3 = xml.get_widget ('button3')
20 entry = xml.get_widget ('entry')
21
22 button1.connect ("clicked", do_click, "button1") # connect signals as before
23 button2.connect ("clicked", do_click, "button2") # however specify a user_data
24 button3.connect ("clicked", do_click, "button3") # field to be passed to the
25                                                   # callback
26
27 window.connect ("destroy", do_exit)
28
29 window.show_all ()
30
31 gtk.main ()
```



Using libglade :: signal autoconnection

- signal autoconnection works in Python!
  - doesn't work so great in C
  - you can choose if you want to use it

```
1  #!/usr/bin/python
2
3  import gtk
4  import gtk.glade
5
6  class Signals:                          # callbacks class
7          def do_exit (self, window):
8                  gtk.main_quit ()
9
10         def do_click (self, button):
11                 entry.set_text (button.get_label ())
12
13 xml = gtk.glade.XML ('libglade2.glade', None, None)
14
15 window = xml.get_widget ('window')
16 entry = xml.get_widget ('entry')
17
18 xml.signal_autoconnect (Signals ())
19
20 window.show_all ()
21
22 gtk.main ()
```



Even More Libraries

- libxml2, libxslt
- PyGTK
  - GTK, GDK, ATK, pango, libglade
- gnome-python
  - bindings for lots of useful GNOME libraries
  - libgnome, GConf, GNOME-VFS, Bonobo
- nautilus-python, pyphany, ...
- GStreamer
  - Platform and Desktop independent multimedia framework

and many, many more...

A GConf Example

- GConf is GNOME's *asynchronous* configuration system.
  - the powerhouse behind those instant apply preferences
  - We can register our interest in certain *GConf Keys* and get callbacks when they change
  - this is exactly the same as events from widgets or other GObjects.
  - Gconf is nothing like the Windows Registry. The only things is has in common is that it has a tree structure, it can store preferences, and an application exists to make changes to it by hand.

A GConf Example

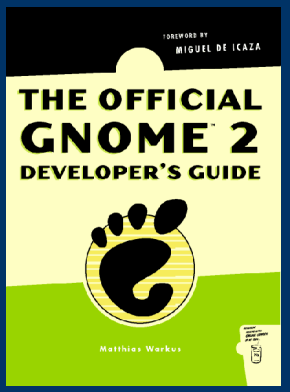
```
#!/usr/bin/python

import gtk
import gtk.glade
import gconf

class GConfExample:
        # signals
        def do_exit (self, window):
                gtk.main_quit ()
        def new_background (self, client, cnxn_id, entry, user_data):
                image = client.get_string (
                        "/desktop/gnome/background/picture_filename")
                pixbuf = gtk.gdk.pixbuf_new_from_file_at_size (image, 300, 300)
                self.entry.set_text (image)
                self.image.set_from_pixbuf (pixbuf)

        # initilisation
        def __init__ (self):
                xml = gtk.glade.XML ('gconf-example.glade', None, None)
                client = gconf.client_get_default ()

                self.image = xml.get_widget ('preview_image')
                self.entry = xml.get_widget ('image_name')

                xml.signal_autoconnect (self)
                client.add_dir ("/desktop/gnome/background",
                                gconf.CLIENT_PRELOAD_NONE)
                client.notify_add ("/desktop/gnome/background/picture_filename",
                                self.new_background)

                self.new_background (client, None, None, None)
                xml.get_widget ('window').show_all ()

if __name__ == '__main__':
        GConfExample ()
        gtk.main ()
```

In Conclusion

- The object-orientated nature of GTK+ adapts beautifully to Python.
- It is very easy to write quick GUI applications using tools provided by the GNOME Platform.
- These tools exist today, and are proven time and time again:
  - GTK+, Python, PyGTK, libxml and Glade, as well as the GNOME libraries: GConf, GNOME-VFS, etc...
- Everything you learn writing GTK+ on Python adapts to most other languages and Platforms, including Java, C, C++ and C#.

Would You Like to Know More?

- PyGTK Tutorial
  - http://www.pygtk.org/tutorial.html
- PyGTK API Reference
  - http://www.pygtk.org/pygtk2reference/
- The Official GNOME2 Developer's Guide
  - Matthias Warkus, No Starch Press
  - Available from Boffins in Perth and Amazon
- GTK+ Tutorials and Documentation
  - http://developer.gnome.org/doc/tutorials/
  - http://developer.gnome.org/doc/API/



Fin ;)

Questions?

http://www.davyd.id.au/articles.shtml
davyd@madeley.id.au