# PyGTK Notebook A Journey Through Python Gnome Technologies

Peter Gill

October 9, 2012

Version 0.13 October 9, 2012

# Contents

# List of Figures

## ChangeLog

### Version 0.03

23 Dec 2008

- Added Secton on Widgets - 1.2.1 on page 13
- Added Section on First PyGTK Application - 1.2.2 on page 14
- Added Section on Layout Boxes - 1.2.3 on page 14

### Version 0.04

- Added Section on Callbacks - 1.2.4 on page 16
- Added PyGTK on Windows - C on page 175

## Version 0.05

- Added Section on PyGObject (only talks about gobject.timeout_add) - 5.2 on page 80
- Added Section on Labels - 1.3.5 on page 20
- Added Section on Check Buttons - 1.3.4 on page 19

## Version 0.06

12 Feb 2009

- Added code example for gstreamer codec installer - 6.6 on page 96
- Added Section on Buttons - 1.3.1 on page 17
- Added Section on Radio Buttons - 1.3.2 on page 18
- Added Section on Toggle Buttons - 1.3.3 on page 18
- Added Section on Text Entry - 1.3.6 on page 20

## Version 0.07

- Added Section on MessageDialog - 1.3.8 on page 23
- Added Section on Statusbar - 1.3.11 on page 29

## Version 0.08

- Updated chapter on clutter to pyclutter 1.0

## Version 0.09

- Book text license change to creative commons Attribution-ShareAlike 3.0 Unported

## Version 0.10

02 April 2010

- Add new chapter on IronPython and Gtk-Sharp
- Add basic gtk-sharp and IronPython example

## Version 0.11

- Clarify what widgets are compared to .NET winforms 1.2.1

## Version 0.12

- Fixed typos in Internalization code examples 10.3 on page 156

## Version 0.13

- Fixed added notes about using math.degrees and math.radians 3.3.1.1

# Chapter 1

# PyGTK Introduction

Please send any fixes or suggestions to peter@majorsilence.com or leave a comment at http://www.majorsilence.com/pygtk_book.

## 1.1 Introduction

This book has been created as a personal notebook that I may refer back to when I no longer remember how to program something I once did. There has been many a time that I have spent many hours figuring how to do something interesting just to forget how I did it, or where on the web it was found. As I have become tired of doing this I have decided to collect my notes and code samples in one location that is easy for myself to reference. Basically I am using open source code to write an open source book.

Hopefully this information will be useful to others as I have found that many of these topics are not currently collected together in a nice package making it easy to use.

The materials in this book are from several sources from the Internet and programming books that I have read in the past, or as in the instance of the case studies, code I have done myself.

If anything is not cited or referenced properly I now apologize to the original author and will correct it in the next edition.

Please check the books website regularly for updates and errata, the web site is located at: http://www.majorsilence.com/pygtk_book

## 1.2 PyGTK Basics

### 1.2.1 Widgets - What are they?

Before creating your first program lets get out of the way what a widget is. A widget is what makes up a program. They are all the different parts that can be used and include the following:

- Labels
- Buttons

- Menus
- Text Entries
- etc..

So basically that is what they are. If you are used to programming using .NET and winforms you would be use to hearing them referred to as controls. The buttons, labels, text areas of all programs are widgets. There are many different types available when using PyGTK and many of them will be covered in this book, but to start off this chapter will only cover a few such as buttons, labels and text entry.

### 1.2.2 Creating your first PyGTK application

First thing, create a window that will display a small message. To do this pygtk and gtk must be imported.

```
import pygtk
pygtk.require("2.0")
import gtk
```

Now create a label and a GTK window. As you can see below to set the text of a label you just supply the text when you instantiate it. To make add it to the window that you have created you use the windows add method and supply the widget (label). Then to show everything to the user you call the windows *show_ all* method. Last but not least you must call the *gtk.main()* method.

```
label = gtk.Label("Hello World!")
win = gtk.Window()
win.add(label)
win.show_all()
gtk.main()
```

If you do not call the *gtk.main()* method, nothing will happen. It is the main loop that waits for user input and and reactions. It runs all the code that is necessary to display your application.

### 1.2.3 Layout - Boxes

Adding a label to a window is good and well if not useless. What you have to do is create a layout using horizontal and vertical boxes. These boxes can hold PyGTK widgets or other vertical and horiztonal boxes. You will have one main box that will hold all other boxes, this main box will be added to the window. To add a widget to a box, or a box to another box the *pack_ start* and *pack_ end* methods are used.

Now lets expand on the first PyGTK application to include a vertical and horizontal box to layout two labels and a button.

```
import pygtk
pygtk.require("2.0")
import gtk
label_1 = gtk.Label("Hello World!")
label_2 = gtk.Label("Still in the HBox")
button = gtk.Button("This button is in the Vertical Box")
vbox = gtk.VBox()
hbox = gtk.HBox()
```

Start off by creating two labels and a button. A buttons text is set when creating the same way as a labels is by including the text when you create an instance of gtk.Button. Next two layout boxes are created.

The first box created is a vertical box and the second is a horizontal box. This boxes have the following definition gtk.HBox(homogeneous=False, spacing=0). *Homogeneous* is whether each object in the box has the same size. You can have a vertical box (gtk.VBox) or a horizontal box (gtk.HBox). This is how in PyGTK a program has its layout. Take some time and experiment using them. (I also recommend using Glade 3 (See 2.5 on page 46) to create your user interfaces instead of doing it by hand).

```
hbox.pack_start(label_1)
hbox.pack_start(label_2)
# Add the hbox as the first item in the vertical box
# that was created above
vbox.pack_start(hbox)
# Add the button as the next item in the vertical box.
vbox.pack_start(button)
```

With the layout boxes created the labels and button must be added to them. So now the pack_start method of the boxes is used. The definition of these methods is pack_start(child, expand=True, fill=True, padding=0). You have the option of using pack_start which adds the widget to the beginning of the box, or pack_end which appends the widget to the end of the box.

So this code adds label_1 to the first position of the horizontal box then adds label_2 to the next position at the beginning after label_1. Next the horizontal (hbox) is added as the first widget in the vertical (vbox) box. Next the button is added to the next position of the vertical box. When you run this a window should open up with two labels above a button.

- *child* is the widget you are adding to the box
- *expand* argument is whether to fill the extra space in the box (gtk.HBox or gtk.VBox)
- *fill* argument only has an effect if the expand argument is set to True.

All that is left is to run the program. So just like in the first program a gtk.Window is created, but instead of adding a widget such as a label directly to it a layout box is added. Here the vertical box (vbox) is added as it is the top level box that we used to hold all other widgets in the code above. Then call the show_all() method on the window to make all the widgets in the window visible. Now to actually run the program the gtk.main() method must be invoked.

```
win = gtk.Window()
win.add(vbox)
win.show_all()
gtk.main()
```

Run the program and enjoy your glorious creation.

### 1.2.4 Callbacks - Reacting to program events

A program that does not react to user input is usually a useless program. To react to user input such as a mouse click there must be assigned to a widget a signal handler. A signal handler is connected to a widget such as a gtk.Button and listens for a signal.

Take for example, a signal handler could be added to a button that reacts on a mouse click. So lets create a button and add a signal handler:

```
button = gtk.Button(“example button”)
button.connect("clicked", on_button_clicked)
```

What this code does is create a button that when "*clicked*" will call the function *on_button_clicked*. In the example below we there is no longer a gtk.HBox, only a vertical gtk.VBox is used and the button has signal handler to connect *clicked* signals to the *on_button_clicked* callback function. What this means is that when the button is clicked the function named on_button_clicked will be called.

```
import pygtk
pygtk.require("2.0")
import gtk
def on_button_clicked(widget, data=None):
  label_1.set_text("Hello " + str(data))
  label_1 = gtk.Label("Hello World!")
  label_2 = gtk.Label("Still in the HBox")
  button = gtk.Button("Click Me")
  # Connect the "clicked" signal of the button to
  # our callback function that we have named
  # on_button_clicked. It also passes the string
  # "Anything can go here" to the callback function.
  button.connect("clicked", on_button_clicked, "Anything can go here")
  vbox = gtk.VBox()
  vbox.pack_start(label_1)
  vbox.pack_start(label_2)
  vbox.pack_start(button)
  win = gtk.Window()
  win.connect("destroy", lambda wid: gtk.main_quit())
  win.add(vbox)
  win.show_all()
  gtk.main()
```

## 1.3 Widgets

Many of the widgets that are going to be discussed here will make use of a smaller gtk gui that will be shown here. However there will be a few examples that will utilize an object oriented design. Here the basic gui that creates a window and adds a vertical box (gtk.VBox) to add our test widgets into.

```
#!/usr/bin/env python
import pygtk, gtk
pygtk.require('2.0')
def main():
  win = gtk.Window(gtk.WINDOW_TOPLEVEL)
  win.connect("delete_event", lambda wid, we: gtk.main_quit())
  vbox = gtk.VBox(True, 2)
  win.add(vbox)
  # Add widget code here
  win.show_all()

if __name__ == "__main__":
  main()
  gtk.main()
```

So when adding the code, from widgets discussed below, make sure it is between the win.add(vbox) and win.show_all() lines. All the widget will be added to the widget *vbox*.

### 1.3.1 Buttons

To create a button the gtk.Button class is instantiated.

```
button = gtk.Button("Click Me")
button.connect("clicked", button_callback, "Button Click Me")
vbox.pack_start(button, True, True, 2)
```

This code creates a button that displays the text "Click Me" on the button. It then connects the buttons when clicked to the *function button_callback* and sends the data "Button Click Me" as a function argument. Make sure that the button_callback function is declared before the code that calls it.

```
def button_callback(widget=None, data=None):
  print "%s was clicked." % data
```

The function button_callback prints the out a small message that includes the "Button Click Me" string that was sent as an argument.

### 1.3.2 Radio Buttons

Radio buttons are created using the gtk.RadioButton(group, label) class. Groups are used so that only one radio button can be selected at a time within a group. The label of course being the text that is displayed along with the radio button.

To create the first radio button pass the value None in for the group. Than for each radio button you want in the group pass the first button in as the group. The following code will now show this.

```
button1 = gtk.RadioButton(None, "Radio Button 1")
button2 = gtk.RadioButton(button1, label="Radio Button 2")
button3 = gtk.RadioButton(button1, label="Radio Button 3")
```

These three lines show three radio buttons being created with the first one having a group of None. The second and third buttons however have the group set to button1. This way only one of the three buttons can be selected at one time.

Now the buttons are connected to a callback.

```
button1.connect("toggled", button_callback, "Button 1")
button2.connect("toggled", button_callback, "Button 2")
button3.connect("toggled", button_callback, "Button 3")
```

What this does is connect any toggled (switching from one button to another) signal to the function *button_callback*.

```
def button_callback(widget=None, data=None):
  print "%s was toggled %s" % (data, ("off","on")[widget.get_active()])
```

This fuction will print out the data argument "on" when the button is selected and "off" when another button is selected. What this means is that when button1 is currently selected and then button two is clicked it will print the lines:

```
Button 1 was toggled off
Button 2 was toggled on
```

### 1.3.3 Toggle Buttons

Toggle buttons are very much the same as normal buttons except they are either in a state of *on* (clicked) or *off* (not clicked). They work much the same say that radio and check buttons work. Toggle buttons are created using the gtk.ToggleButton class and take as an argument a label.

```
button1 = gtk.ToggleButton("Toggle Button 1")
button2 = gtk.ToggleButton("Toggle Button 2")
```

This code shows two toggle buttons being created. To make them useful they are connected to the *toggled* signal to call the function *button_callback* with "Button 1" and "Button 2" as function arguments.

```
button1.connect("toggled", button_callback, "Button 1")
button2.connect("toggled", button_callback, "Button 2")

def button_callback(widget=None, data=None):
  print "%s was toggled  %s" % (data, ("off",
    "on")[widget.get_active()])
```

The button_callback function will print on or off for each button as they are toggled. The widget.get_active() method can be used to decide the code path by doing one action when toggled and another action when it is toggled off.

All that is left is to add the buttons to the gtk.VBox that is in the user interface code.

```
vbox.pack_start(button1, True, True, 2)
vbox.pack_start(button2, True, True, 2)
```

### 1.3.4 Check Buttons

To create a check button with a label of "Check Me" do the following

```
check_button = gtk.CheckButton("Check Me")
```

Unlike a normal button, instead of connecting to the *clicked* signal, a check button connects a callback to a *toggled* signal. So to do some action on the above you would connect like so:

```
check_button.connect("toggled", check_button_callback, "callback data")
```

So this will call the function named *check_button_callback* whenever the check box is toggled(clicked). Take a look at the following example to see how to detect whether a check button is checked or not.

```
def check_button_callback(widget, data=None):
  print "%s was toggled: %s" % (data, ("off", "on")[widget.get_active()])
```

This function takes the check button widget and print the string data that was passed in. It also prints "off" for when the button is not clicked and "on" when the button has been clicked.

Below is the code that is needed to create the buttons and connect them to the *check_button_callback* function.

```
  button1 = gtk.CheckButton("check button 1")
  button1.connect("toggled", check_button_callback, "Button 1")
  vbox.pack_start(button1, True, True, 2)

  button2 = gtk.CheckButton("check button 2")
  button2.connect("toggled", check_button_callback, "Button 2")
  vbox.pack_start(button2, True, True, 2)
```

### 1.3.5 Labels

To create a label just do something like this but replace the labels text with your own.

```
label = gtk.Label(“Your label”)
```

If you wish to change the text later you can use the labels *set_ text* method.

```
label.set_text(“My new label”)
```

Now the label will display the text “My new label” instead of “Your label”.

### 1.3.6 Text Entries

The text entry example is slightly more complicated than the examples that have been shown so far. This is because besides the text entry, two buttons and a label will be used in this example. The first button called *print_ button* is used to print retrieve the text from the text entry and place it into the label. The second button*, clear_ button,* is used to clear the text from the text entry and label.

To create a text entry the gtk.Entry class is used. By default it is gtk.Entry(max=0). The max argument is the is the size of characters that the entry can hold. If it is set to 0 then there is no limit.

The following code creates a gtk.Entry called text_box with no limit on the size.

```
text_box = gtk.Entry()

print_button = gtk.Button("Print Text")
print_button.connect("clicked", print_callback, text_box)

clear_button = gtk.Button("Clear Text")
clear_button.connect("clicked", clear_callback)
```

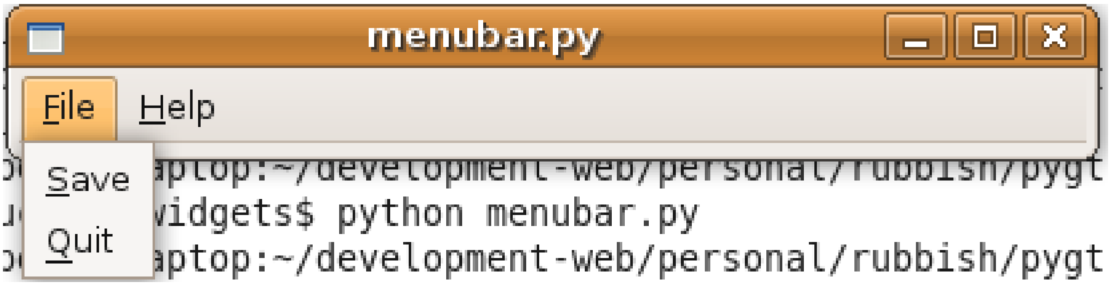
After creating a text box two buttons are created. The first, *print_ button*, is connected to the *print_ callback* function when it is clicked and passes as an argument the text_box gtk.Entry widget as an argument.

The *print_ callback* funtion receives the gtk.Entry *text_ box* as the argument data and sets the text of the global gtk.Label label to the text that was entered in the *text_ box* widget using the gtk.Entry method *get_ text()*

```
label = gtk.Label(“Hello”)
def print_callback(widget=None, data=None):
  label.set_text(data.get_text())
```

The clear_callback function clears the text in the text entry and just for fun the label as well.

```
def clear_callback(widget=None, data=None):
  text_box.set_text(“”)
  label.set_text(“”)
```



```
aptop:~/development-web/personal/rubbish/pygt
idgets$ python menubar.py
aptop:~/development-web/personal/rubbish/pygt
```

Figure 1.1: File Menu Screenshot

Now the widgets just need to be added to the gtk.VBox that is in the user interface code.

```
vbox.pack_start(label, True, True, 2)
vbox.pack_start(text_box, True, True, 2)
vbox.pack_start(print_button, True, True, 2)
vbox.pack_start(clear_button, True, True, 2)
```

Here are some methods available with gtk.Entry:

- insert_text(text, position=0)
- get_text()
- set_text(text)
- set_max_length(max)
- set_editable(is_editable) - True or False
- set_visibility(visible) - True or False
- select_region(start, end)

### 1.3.7 Menus

This section will cover adding menus to applications that most everyone should be used to. The standard menus such File -> Save, File -> Quit, and Help -> About. Of course after reading this section you will be more than capable to add what ever menu you wish.

The method used this section will be using is to create the menus using straight code. There is another method using the UIManager[1] and if you would like you can look into that instead.

There are three main class that are used in creating menus and they are:

- gtk.MenuBar - Is added to the the programs main window and is a container for gtk.Menu and gtk.MenuItem

[1] http://www.pygtk.org/pygtk2tutorial/sec-UIManager.html

- gtk.Menu - Is a container to hold sub gtk.MenuItem items
- gtk.MenuItem - Is the actual menus items the user sees and actually clicks such as “File”, “Save”, and “Quit”

Looking at the code below, it can be seen that the menu bar is created using the class gtk.MenuBar. This is the object that will be added to the main windows, in this case the top of the gtk.VBox that is being used in this example.

```
menubar = gtk.MenuBar()
file_item = gtk.MenuItem("_File")
help_item = gtk.MenuItem("_Help")
```

After the MenuBar is created two MenuItems are created, *file_item* and *help_item*, these of course will have other sub menu items attached to them that will be displayed when they are clicked. These are the main menu items that are seen in most applications along the top of the window (Eg. File, Edit, View, Tools, Help, etc...) In this case only *File* and *Help* are shown. The underscores before the F and H indicate that

Here find the menu container *file_item_sub* being created as a gtk.Menu object to hold the menu items that will be apended to the file_item MenuItem. Save and quit are both created as gtk.MenuItem objects. These are then added to file_item_sub. A few lines further down, file_item_sub will be added to file_item.

```
file_item_sub = gtk.Menu()
save = gtk.MenuItem("_Save")
quit = gtk.MenuItem("_Quit")
file_item_sub.append(save)
file_item_sub.append(quit)
```

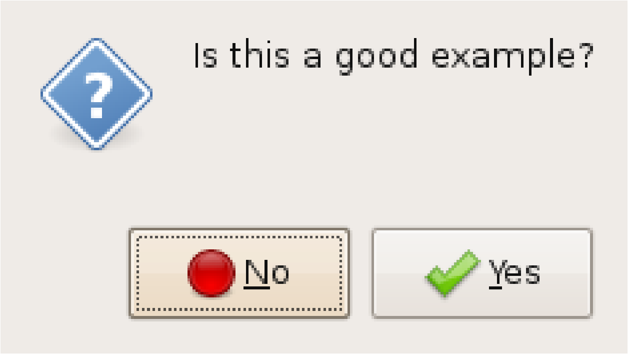
As was done with creating file_item_sub so to this done here creating help_item_sub. This is a submenu container to hold the MenuItems for the Help MenuItem.

```
help_item_sub = gtk.Menu()
about = gtk.MenuItem("_About")
help_item_sub.append(about)
```

Finally here can be seen the submenus being added to their respective parent MenuItems and then the parent MenItems being added to the MenuBar.

```
file_item.set_submenu(file_item_sub)
help_item.set_submenu(help_item_sub)
menubar.append(file_item)
menubar.append(help_item)
```

To finish off each menu item that is to have a user action connects to the activate signal that is emitted on its selection, each MenuItem calling its respective callback function. And lets not forget, the menubar is added to the gtk.VBox that was created in the base user interface code ( 1.3 on page 17).



Is this a good example?

Figure 1.2: MessageDialog Example

```
save.connect("activate", save_callback)
quit.connect("activate", quit_callback)
about.connect("activate", about_callback)
vbox.pack_start(menubar, True, True, 2)
```

For the sake of completness these are the callback functions; very simple and not very much, but you can use your own imagination as what should be done in your own program.

```
def save_callback(widget=None):
  print "Save menu item was pressed"

def quit_callback(widget=None):
  print "Quit menu item was pressed"
  gtk.main_quit()

def about_callback(widget=None):
  print "About menu item was pressed"
```

### 1.3.8 Message Dialogs

Message Dialogs are small windows that are smiple and easy to use. Using them is as simple as calling the gtk.MessageDialog class. The default constructor of this class looks like this.

```
gtk.MessageDialog(parent=None, flags=0, type=gtk.MESSAGE_INFO,
   buttons=gtk.BUTTONS_NONE, message_format=None)
```

The *parent* is either the parent window of None if none.

The flags can be one of the following:

- gtk.DIALOG_MODAL
- gtk.DIALOG_DESTROY_WITH_PARENT

- or 0 for no flags.

The *type* can be one of the following:

- gtk.MESSAGE_INFO - display an information icon
- gtk.MESSAGE_WARNING - display a warning icon
- gtk.MESSAGE_QUESTION - display a question icon
- gtk.MESSAGE_ERROR - display an error icon

The buttons available are:

- gtk.BUTTONS_NONE
- gtk.BUTTONS_OK
- gtk.BUTTONS_CLOSE
- gtk.BUTTONS_CANCEL
- gtk.BUTTONS_YES_NO
- gtk.BUTTONS_OK_CANCEL

These are the responses to the button types:

- gtk.RESPONSE_NONE
- gtk.RESPONSE_REJECT
- gtk.RESPONSE_ACCEPT
- gtk.RESPONSE_DELETE_EVENT
- gtk.RESPONSE_OK
- gtk.RESPONSE_CANCEL
- gtk.RESPONSE_CLOSE
- gtk.RESPONSE_YES
- gtk.RESPONSE_NO
- gtk.RESPONSE_APPLY
- gtk.RESPONSE_HELP

The *message_format* is the message that will be displayed. So far this seems as if it is not complicated and it is not.

Here is an example showing a MessageDialog displaying a question with buttons to answer yes or no. As can be seen the message dialog is instantied with the *parent* set to None, the *button* type is gtk.BUTTONS_YES_NO, the *flag* is gtk.DIALOG_DESTROY_WITH_PARENT. The *type* is set to gtk.MESSAGE_QUESTION to go along with the yes/no button. The message that is displayed is “Is this a good example?”.

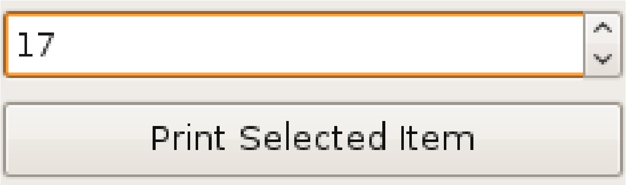


Figure 1.3: SpinButton Screenshot

```
def button_callback(widget=None):
  dialog = gtk.MessageDialog(parent = None,
      buttons = gtk.BUTTONS_YES_NO,
      flags =gtk.DIALOG_DESTROY_WITH_PARENT,
      type = gtk.MESSAGE_QUESTION,
      message_format = "Is this a good example?")

  dialog.set_title("MessageDialog Example")
  result = dialog.run()
  dialog.destroy()

  if result == gtk.RESPONSE_YES:
    print "Yes was clicked"
  elif result == gtk.RESPONSE_NO:
    print "No was clicked"
```

After the Message dialog is assigned to the variable *dialog* the title of the dialog window is set to "MessageDialog Example". To run a dialog you must use the dialogs *run* method. The dialogs run method returns the result of the buttons that was clicked. This can be used to determine the course of action.

As can be seen in the example if the Yes buttons is clicked the message "Yes was clicked" is printed and if No is clicked the message "No was clicked" is printed. Also make sure that you remember to also call the dialogs *destroy* method otherwise it will never close. So *dialog.destroy()* is called on the line immedialty following *dialog.run()*.

Finally, lets not forget the code to display the button that will run the button_callback function:

```
button = gtk.Button("Show Dialog")
button.connect("clicked", button_callback)
vbox.pack_start(button, True, True, 2)
```

As can be seen the message dialog is easy to use and it makes it simple to display information, warnings, errors, or questions to the user.

### 1.3.9 Spin Buttons

To create a spin button the gtk.SpinButton class is used.

```
spin_button = gtk.SpinButton(adjustment=None, climb_rate=0.0, digits=0)
```

The adjustment is as follows:

```
adjustment = gtk.Adjustment(value=0, lower=0, upper=0, step_incr=0,
    page_incr=0, page_size=0)
```

- value -initial value for the Spin Button
- lower - lower range value
- upper - upper range value
- step_incr - value to increment/decrement when pressing mouse button-1 on a button
- step_incr - value to increment/decrement when pressing mouse button-2 on a button
- page_size unused

In this example andjustment is created with an inital value and lower limit of 0, an upper limit of 100, a step increment of 1, a page increment 5, and page size of 0)

```
#gtk.Adjustment(value=0, lower=0, upper=0, step_incr=0, page_incr=0, page_size=0)
   adjustment = gtk.Adjustment(0, 0, 100, 1, 5, 0)
   spin = gtk.SpinButton(adjustment, 0, 0)
   vbox.pack_start(spin, True, True, 2)
```

Here a button is added that will call the button_callback function.

```
   button = gtk.Button("Print SpinButton Value")
   button.connect("clicked", button_callback, spin)
   vbox.pack_start(button, True, True, 2)
```

The button callback prints the value of the spinbutton, first as a float and secondly as an integer.

```
def button_callback(widget=None, spin=None):
  print spin.get_value()
  print spin.get_value_as_int()
```

For much more information and details on the gtk.SpinButton class see the PyGTK tutorial at: http://www.pygtk.org

### 1.3.10 Combo Box

The easy way to create and populate a ComboBox is to use one of the following functions:

```
# Setup up a read only combobox
item_list = gtk.combo_box_new_text()

# Setup a combobox that users may add to
item_list = gtk.combo_box_entry_new_text()
```

Using either of these functions setups a combo box and provides some easy to use convience functions. These are the methods that are provided when using combo_box_new_text:

- append_text(text)
- prepend_text(text)
- insert_text(position, text)
- combobox.remove_text(position)

The example that will be shown below will use the second function, gtk.combo_box_entry_new_text, because it provides everything that the gtk.combo_box_new_text does plus allows the user to update the list by typing in new data directly. If this functionality is not needed then it can be avoided by using the first function and not using the code below that pertains to adding new list items.

Now the ComboBox example will break from using the user interface supplied at the at the begginning of the widget section ( 1.3 on page 17), as it will use a slightly modified version so that it will now be used within a class. The example will use the same basic code but will now be within the CodeExample class that will be created. The only reason for this is because the author (thats me) does not like using global variables when it can be avoided.

```
class ComboExample:
  def __init__(self):
    win = gtk.Window(gtk.WINDOW_TOPLEVEL)
    win.connect("delete_event", lambda wid, we: gtk.main_quit())
    vbox = gtk.VBox(True, 2)
    win.add(vbox)
```

So far the code is the same except instead of the main function the user interface code is in the __init__ method. Now the actual code for the comboboxes.

First the list default_items is created to hold a couple of items that will be placed in the combobox, the combo box is created right beneath this using the function combo_box_entry_new_text. Using this function means that this will combobox will allow its users to enter text directly into a text entry that is provided in the combobox.

```
    default_items = ["hello", "World"]
    self.item_list = gtk.combo_box_entry_new_text()
    self.item_list.child.connect('key-press-event',
        self.item_list_changed)
    for x in default_items:
      self.item_list.append_text(x)
```

After the combobox has been created and assisigned to the variable self.item_list it is connects the key-press-event signal to all the item_list_changed method. The reason for doing this is to detect when text is entered into the combobox text entry area by the user. Following this the default_items list is appended into the combox box using the append_text method. Very simple, very easy.

To show how to retrieve the selected item a button is added that when clicked will retrieve the combobox item that is selected by calling the print_selected_item method.

```
    button = gtk.Button("Print Selected Item")
    button.connect("clicked", self.print_selected_item)
    vbox.pack_start(self.item_list, True, True, 2)
    vbox.pack_start(button, True, True, 2)
    win.show_all()
```

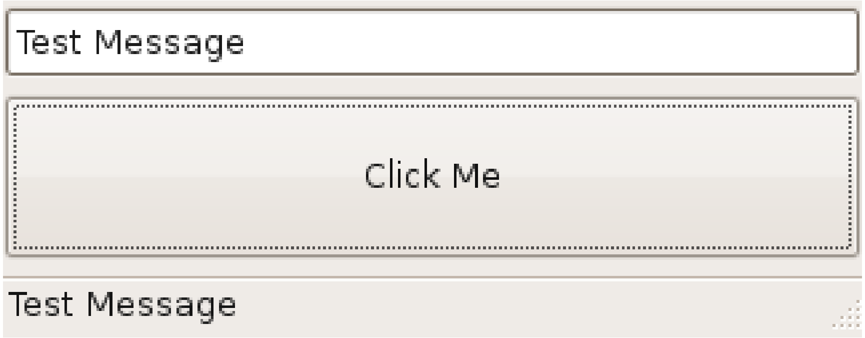
The item_list_changed method is called every time there is a changed in the combobox text entry field. What this means is everytime a character is entered by a user this method is called and checks what keyboard button is pressed. If the keyboard character pressed is Return (Enter) than the text entry is append to the item_list using the its append_text method and then sets the combobox text entry back to an empty string.

```
    def item_list_changed(self, widget=None, event=None):
      key = gtk.gdk.keyval_name(event.keyval)
      if key == "Return":
        self.item_list.append_text(widget.get_text())
        widget.set_text("")
```

The print_selected_item method is called when the button is pressed. Its sole purpose is to retrieve what item is selected in the combox. If there are no items selected then None is returend. Else the item is printed and also returned.

```
     def print_selected_item(self, widget=None):
       model = self.item_list.get_model()
       active = self.item_list.get_active()
       if active < 0:
         return None
       print model[active][0]
       return model[active][0]
```

As can be seen to retrieve the selected items the combobox item_list methods *get_model* and *get_active* most be used. The model is a gtk.TreeModel. If the active number is less than 0 then there are no selected items, otherwise is the postion of the selected item.



Test Message

Test Message

Figure 1.4: Statusbar Example

```
if __name__ == "__main__":
  ComboExample()
  gtk.main()
```

### 1.3.11 Statusbar

The status bar will break from using the user interface supplied at the beggining of this widget section, as it will use a slightly modified version so that it will now be used within a class. The example will use the same basic code but will now be within the StatusbarTest class that will be created. The only reason for this is because the author (thats me) does not like using global variables for no particular reason, I just do not like doing it unless it is a constant variable.

Now that the user interface is within a class it is easy to work with multiple widgets by making them class level instance variables.

When working with the gtk.Statusbar class the important methods to know are:

- gtk.Statusbar() - Well not really a method but create an instance of the class
- pop(context_id) - Remove the top level message
- push(context_id, text) - Add a new top level message
- get_context_id(context_id) - Used to retrieve the context that is used with the *pop* and *push* methods

```
class StatusbarTest(object):
  def __init__(self):
    win = gtk.Window(gtk.WINDOW_TOPLEVEL)
    win.connect("delete_event", lambda wid, we: gtk.main_quit())
    vbox = gtk.VBox(False, 2)
    win.add(vbox)
```

So next is the code for creating the Statusbar. As can be seen once it is created a context_id variable is assinged by using the statusbars *get_ context_ id* method using the context_id "Status Test". So whenever a message needs to be popped or pushed it needs to use the context id that was created with the *get_ context_ id* method.

```
        self.statusbar = gtk.Statusbar()
        self.context_id = self.statusbar.get_context_id("Status Test")
```

The rest of the code here is common user interface code that has been common throught the widget section, all it does is create a text entry and a button.

```
        self.text_entry = gtk.Entry()
        button = gtk.Button("Click Me")
        button.connect("clicked", self.button_callback)

        vbox.pack_start(self.text_entry, False, True, 2)
        vbox.pack_start(button, True, True, 2)
        vbox.pack_start(self.statusbar, False, True, 2)

        win.show_all()
```

Here is the rest of the interesting code. First thing that is done in the button_callback function is to remove the top level message using the pop method. Next the new message is displayed to the statusbar using the push method, the text is taken from the text entry widget. To test it out run the code, type something into the text entry and click the button.

```
      def button_callback(self, widget=None):
        self.statusbar.pop(self.context_id)
        self.statusbar.push(self.context_id, self.text_entry.get_text())
```

The rest of the boring code that is needed to run the example.

```
    if __name__ == "__main__":
      StatusbarTest()
      gtk.main()
```

See 1.4 on the previous page to see what this example looks like.

# Chapter 2

# More PyGTK

Please send any fixes or suggestions to peter@majorsilence.com or leave a comment at http://www.majorsilence.com/pygtk_book.

## 2.1 Drag and Drop

I will not be writing very much about drag and drop, just enough to be useful in the slide show demonstration program that this notebook is leading towards. There are a few things we need to know.

The only part of drag and drop that we care about for this program is drag_dest_set(flags, targets, actions).

flags [1] - according to the PyGTK tutorial, flags are:

- gtk.DEST_DEFAULT_MOTION: If set for a widget, GTK+, during a drag over this widget will check if the drag matches this widget's list of possible targets and actions. GTK+ will then call drag_status() as appropriate.
- gtk.DEST_DEFAULT_HIGHLIGHT: If set for a widget, GTK+ will draw a highlight on this widget as long as a drag is over this widget and the widget drag format and action is acceptable.
- gtk.DEST_DEFAULT_DROP: If set for a widget, when a drop occurs, GTK+ will check if the drag matches this widget's list of possible targets and actions. If so, GTK+ will call drag_get_data() on behalf of the widget. Whether or not the drop is successful, GTK+ will call drag_finish(). If the action was a move and the drag was successful, then TRUE will be passed for the delete parameter to drag_finish().
- gtk.DEST_DEFAULT_ALL: If set, specifies that all default actions should be taken.

targets – is a list of target data types that are supported along with in app information such as mime types of those files that can be dragged along with some.

actions – are the actions that are to be taken with the drag and include the following:

[1] Take a look at: http://pygtk.org/pygtk2tutorial/sec-DNDMethods.html

- gtk.gdk.ACTION_DEFAULT
- gtk.gdk.ACTION_COPY
- gtk.gdk.ACTION_MOVE
- gtk.gdk.ACTION_LINK
- gtk.gdk.ACTION_PRIVATE
- gtk.gdk.ACTION_ASK

The only action we will be using is gtk.gdk.ACTION_COPY and this is only on non win32 systems. For whatever reason I do not believe anything really works on a Windows system properly. I believe this actually because I have never properly been able to get a target properly specified, thus it never works on Windows so I have never bothered to go beyond drag_dest_set(0, [], 0). I see no point.

With that you can drag a file(s) anywhere into application then bother sorting out where it goes based on the file type that it is. I am sure in more complicated applications that this would not be enough but I have never personally needed more then this.

Now back to targets on anything other then Windows (Linux programs). For a target we will want to set up its file type. It will be in the form of (string, int, int).

So what we will end up with for the target will be something such as (“text/plain”, 0, TARGET_STRING). TARGET_STRING must be an integer assigned above. It is a number that keeps track of the target throughout the program.

For flags we will probably just want to go with gtk.DEST_DEFAULT_ALL covering all the flags leaving us with less typing.

As I said before we will only use gtk.gdk.ACTION_COPY for the actions part and this will only be for the part that are running on Linux systems.

So what we end up with on Linux is a function call that looks like this:

```
drag_dest_set(gtk.DEST_DEFAULT_DROP, [(“text/plain”, 0,
TARGET_STRING), (“image/*”, 0, TARGET_IMAGE)],
gtk.gdk.ACTION_COPY)
```

While on windows we will only be using a much smaller:

```
drag_dest_set(0, [], 0)
```

We will need to attach this to a widget. In our case the widget will be the main window:

```
win = gtk.Window()
win.set_size_request(400, 400)
if sys.platform == “win32”:
    win.drag_dest_set(0, [], 0)
else:
    win.drag_dest_set(gtk.DEST_DEFAULT_DROP,
        [(“text/plain”, 0, TARGET_STRING),
        (“image/*, 0, TARGET_IMAGE)],
        gtk.gdk.ACTION_COPY)
```

The thing is that using more then the method that is being used for Windows is not needed for this program and I am only showing the other version for Linux just to introduce flags, targets, and actions.

Now that drag_dest_set has been attached to our main window widget we need to handle three singles:

- drag_motion
- drag_drop
- drag_data_received

What we do is connect them to three functions like so:

```
win.connect(“drag_motion”, self.motion_cb)
win.connect(“drag_drop”, self.drop_cb)
win.connect(“drag_data_received”, self.drag_data_received)
```

How this works is not very important for our purposes. We just want it accepting images for us. If you want more information on how this works check out the PyGTK drag and drop tutorial at http://pygtk.org/pygtk2tutorial/ch-DragAndDrop.html or check out the drag and drop demo included in the PyGTK source code found at http://www.pygtk.org.

One last thing that I want to mention is that in the function drag_data_received we will be detecting if the files are in an accepted list of file types. If they are, in this example we add them to a list. What we will do in the slide show program is add them to the Item list in the GUI using a TreeView.

What you should end up with when everything is said and done is some source code that is similar to the following.

```
import pygtk
import gtk
import sys
import os
class DragDropExample:
    def __init__(self):
        TARGET_STRING = 82
        TARGET_IMAGE = 83
        self.file_list=[] # list to hold our images
        self.accepted_types = [“jpg”, “jpeg”, “png”, “gif”, “bmp”]

        win = gtk.Window()
        win.set_size_request(400, 400)
        win.connect(“delete_event”, lambda w,e: gtk.main_quit())

        vbox = gtk.VBox(False, 0)
        hello = gtk.Label(“Test label to drag images to.”)
        vbox.pack_start(hello, True, True, 0)
        win.add(vbox)

        if sys.platform==”win32”:
            # gtk.DEST_DEFAULT_DROP, does not work on windows
            # because will not match list of possible target
            # matches if you set anything besides a blank []
            # for target on Microsoft windows, it will not call
            # drop_data_received. So we might as well leave it
            # like so and do your own detecting of the files
            # and what to do with them in drag_data_received.

            win.drag_dest_set(0, [], 0)
        else:
            win.drag_dest_set(gtk.DEST_DEFAULT_DROP,
                [(“text/plain”, 0, TARGET_STRING),
                (“image/*”, 0, TARGET_IMAGE)],
                gtk.gdk.ACTION_COPY)

        win.connect(“drag_motion”, self.motion_cb)
        win.connect(“drag_drop”, self.drop_cb)
        win.connect(“drag_data_received”,
            self.drag_data_received)
        win.show_all()

    def motion_cb(self, wid, context, x, y, time):
        context.drag_status(gtk.gdk.ACTION_COPY, time)
        return True

    def drop_cb(self, wid, context, x, y, time):
        print “drop”
        if context.targets:
            wid.drag_get_data(context, context.targets[0], time)
            print “” .join([str(t) for t in context.targets])
            return True
        return False

    def drag_data_received(self, img, context, x, y, data, info, time):
        if data.format == 8:
            print “Received %s ” % data.data

        # Checking for valid file types
        test_data = os.path.splitext(data.data)[1][1:4].lower().strip()
        if test_data in self.accepted_types:
            if sys.platform==”win32”:
                # Remove the file:/// on window systems.
                self.file_list.append(data.data[8:])
                print data.data[8:]
            else:
                # Remove the file:// on linux systems.
                self.file_list.append(data.data[7:])
                print data.data[7:]
            context.finish(True, False, time)
        else:
            context.finish(False, False, time)

if __name__ == “__main__”:
    DragDropExample()
    gtk.main()
```

## 2.2 List Boxes - gtk.TreeView

A list box in PyGTK is a little more difficult then programming one on Windows with winforms. With PyGTK you must use a TreeView. A true view is relatively complicated to use for just a list box, but it is all that is available. A wrapper can be made around a TreeView to form a generic list box. But this will not be included in this code.

A treeview takes the form of gtk.TreeView(model). The model is the type of the item being stored. What will be used here is gtk.ListStore(type).

The type of a ListStore is can be any valid python type (str, int, etc...). This stores the type data and each type becomes a column in a row.

With the information we now have we can create the tree like so:

```
liststore = gtk.ListStore(str)
treeview = gtk.TreeView(liststore)
```

The above code will create a list box with 1 column. Also it is possible to set the type of modal of the TreeView after creating an instance.

```
treeview.set_model(liststore)
```

Now, to make this useful a CellRenderer is needed. I will be using a CellRendererText.

```
cell = gtk.CellRendererText()
```

The cell is what is used to display the data from the treeview model (liststore) to the user. The cell is then added to a gtk.TreeViewColumn like so:

```
treeviewcolumn = gtk.TreeViewColumn(“Button Pushed”, cell, text=0)
```

The above code will create a TreeViewColumn with a column header of “Button Pressed” assigned the data from the CellRendererText “cell” and display the cells text to column 0.

With the treeviewcolumn created we go ahead and append it to the treeview that we created:

```
treeview.append_column(treeviewcolumn)
```

To append data to a treeview you use the following code:

```
model = treeview.get_model()
model.append([“Your Message”])
```

To remove a selected row from a TreeView you would use the following code:

```
selection = self.treeview.get_selection()
model, iter = selection.get_selected()
if iter:
model.remove(iter)
return
```

If you want more then 1 column you have to create a CellRenderer and TreeViewColumn for each and append to the treeview. You must also have a data type in the ListStore for each column that you will be using. Examine the code below to see how this is applied to making a small program with two columns.

```
import pygtk
pygtk.require(“2.0”)
import gtk
class TreeViewExample:
    def __init__(self):
        # Count the items in the item list
        self.counter = 0
        self.win = gtk.Window()
        self.win.set_size_request(400, 400)
        self.win.connect(“delete_event”, lambda w,e: gtk.main_quit())
        vbox = gtk.VBox(False, 0)
        hbox = gtk.HBox(False, 0)
        add_button = gtk.Button(“Add Item”)
        add_button.connect(“clicked”, self.add_button_clicked)
        remove_button = gtk.Button(“Remove Item”)
        remove_button.connect(“clicked”, self.remove_button_clicked)
        # Treeview Stuff
        self.liststore = gtk.ListStore(str, str)
        self.treeview = gtk.TreeView(self.liststore)
        # Add cell and column.
        # data added to treeview.
        self.cell = gtk.CellRendererText()
        self.cell2 = gtk.CellRendererText()
        # text=number is the column the text is displayed from
        self.treeviewcolumn = gtk.TreeViewColumn(“Button Pushed”,
            self.cell, text=0)
        self.treeviewcolumn2 = gtk.TreeViewColumn(
                    “Second Useless Column”, self.cell2, text=1)
                self.treeview.append_column(self.treeviewcolumn)
                self.treeview.append_column(self.treeviewcolumn2)
                vbox.pack_start(self.treeview, True, True, 0)
                vbox.pack_start(hbox, False, True, 0)
                hbox.pack_start(add_button, True, True, 0)
                hbox.pack_start(remove_button, True, True, 0)
                self.win.add(vbox)
                self.win.show_all()
            def add_button_clicked(self, w):
                self.counter += 1
                model = self.treeview.get_model()
                model.append([“Add Button Pushed %s times”
                    % self.counter, “Column 2 Message”])
            def remove_button_clicked(self, w):
                selection = self.treeview.get_selection()
                model, iter = selection.get_selected()
                if iter:
                    model.remove(iter)
                    return

        if __name__ == “__main__”:
            TreeViewExample()
            gtk.main()
```

For a much more detailed look at the available options in a TreeView visit: http://pygtk.org/pygtk2tutorial/ch-TreeViewWidget.html

### 2.2.1 Single Click - Multiple Select

Say that multiple items in the list need to be selected and by single clicking. This will be difficult to accomplish quickly wading through the official documentation[2]. Basically a few things need to be added to the above TreeView example.

First of all the *selection* that is created in *remove_button_clicked* needs to be removed as it will now be created in the __init__ method. Now selection is a class instance variable *self.selection*, change the code to match.

So in the __init__ method after

```
self.treeview = gtk.TreeView(self.liststore)
```

Please add the following two lines of code.

```
self.selection = self.treeview.get_selection()
self.selection.set_mode(gtk.SELECTION_MULTIPLE)
```

[2] Oh do I ever know it. Talk about wasted hours of my life I am never getting back.

These two lines create the selection as a class level instance and set it up to allow multiple selections. Now to work with this the *changed* signal is emitting and needs to be connected to.

```
self.selection.connect("changed", self.on_media_files_changed)
```

The above lines connects the *changed* signal that is emitted by single clicks on items to call *self.on_treeview_changed*.

```
def on_media_files_changed(self, widget=None, event=None):
  model, path = self.selection.get_selected_rows()
  for x in path:
    print model[x[0]][0] # model[path][column]
```

This method does not do much in its current form. What it does do is retrieve all the selected rows and prints out their values from column one.

## 2.3 Status Icons

Status Icons can be useful for different reasons. Personally I like to use them to hide long running applications such as my music player. I set it playing then just minimize it to the notification area on my panel. If I want to to do something with it I left click the status icon and my music player pops up. If I want to switch songs I right click on it and it pops up menu with some options, one of which includes moving to the next song.

Creating a status icons is a matter of one line of code to make it display.

```
icon = gtk.status_icon_new_from_stock(gtk.STOCK_ABOUT)
```

This creates a status icon with an icon set to the stock GTK icon[3] about.

Then it is a matter of adding two more lines of code to add left and right click ability to it.

```
icon.connect('popup-menu', on_right_click)
icon.connect('activate', on_left_click)
```

The first line here adds signal handling to catch the *popup-menu* signal. This is caught on when a right click happens. When the popup-menu signal is detected the on_right_click function is called.

The second line detects the *activate* signal when the status icon is left clicked and calls the on_left_click function.

As the example below will show, the programmer is responsible for creating the popup menu. The Status Icon Example creates a status icon, and then connects to the *popup-menu* and *activate* signal. When the popup-menu signal is activated, the on_right_click function creates and shows a popup menu by calling the make_menu function.

The make_menu function displays a menu with the options Open App and Close App. Clicking on Open App will call the function open_app which will display a message dialog by calling the function message. The same thing happens when Close App is clicked.

[3] For a full listing of GTK stock icons take a look at the list of stock icons on page on page 177 or the pygtk website at: http://www.pygtk.org/docs/pygtk/gtk-stock-items.html

Basically this is how a status icon works; just substitute the actions and functions here for what is needed for your application.

Status Icon Example

```
#!/usr/bin/env python
import gtk

def message(data=None):
  """
  Function to display messages to the user.
  """
  msg=gtk.MessageDialog(None, gtk.DIALOG_MODAL,
    gtk.MESSAGE_INFO, gtk.BUTTONS_OK, data)
  msg.run()
  msg.destroy()

def open_app(data=None):
  message(data)

def close_app(data=None):
  message(data)
  gtk.main_quit()

def make_menu(event_button, event_time, data=None):
  menu = gtk.Menu()
  open_item = gtk.MenuItem("Open App")
  close_item = gtk.MenuItem("Close App")

  #Append the menu items
  menu.append(open_item)
  menu.append(close_item)
  #add callbacks
  open_item.connect_object("activate", open_app, "Open App")
  close_item.connect_object("activate", close_app, "Close App")
  #Show the menu items
  open_item.show()
  close_item.show()

  #Popup the menu
  menu.popup(None, None, None, event_button, event_time)

def on_right_click(data, event_button, event_time):
  make_menu(event_button, event_time)

def on_left_click(event):
  message("Status Icon Left Clicked")
```

```
if __name__ == '__main__':
  icon = gtk.status_icon_new_from_stock(gtk.STOCK_ABOUT)
  icon.connect('popup-menu', on_right_click)
  icon.connect('activate', on_left_click)
  gtk.main()
```

## 2.4 File choosers

File choosers are used to select files to open or to display a save dialog to the user. This section will cover the gtk.FileChooserDialog, gtk.FileChooserButton, and will also cover using native Windows file choosers when on Windows.

### 2.4.1 gtk.FileChooserDialog

The FileChooserDialog class provides an easy to use way to display a file chooser or save dialog to end users. It is created with a few options and then is run returning succuss or failure. To start off here is a GUI with two buttons and a file filter declard that will be used to launch the file chooser and save dialog.

```
def main():
  #file filters used with the filechoosers
  text_filter=gtk.FileFilter()
  text_filter.set_name("Text files")
  text_filter.add_mime_type("text/*")
  all_filter=gtk.FileFilter()
  all_filter.set_name("All files")
  all_filter.add_pattern("*")

  window = gtk.Window(gtk.WINDOW_TOPLEVEL)
  window.set_title("Filechooser Example")
  window.connect("destroy", lambda wid: gtk.main_quit())
  window.connect("delete_event", lambda e1,e2:gtk.main_quit())

  button_save = gtk.Button("Save File")
  button_open = gtk.Button("Open File")
  button_save.connect("clicked", on_save_clicked, text_filter, all_filter)
  button_open.connect("clicked", on_open_clicked, text_filter, all_filter)
  hbox = gtk.HBox(True, 0) hbox.pack_start(button_save, True, True, 5)
  hbox.pack_start(button_open, True, True, 5)

  window.add(hbox) window.show_all()
```

As can be seen in the code above, the first thing that is done is to seta gtk.FileFilter. One filter for text files and one filter that will be for all file types. The text that is displayed with a file filter is created with the method set_name and the pattern is set using the set_pattern method. For every pattern that is to be matched against there needs to be an instance of the gtk.FileFilter.

Then the GTK window is created. After this two buttons are created; the button_save and button_open buttons. When these buttons are clicked they pass the filters that were created at the top of the function to their respective callback functions.

Now to focus on on the details of filechooser dialogs. First is the save dialog.

```
def on_save_clicked(widget, text_filter=None, all_filter=None):
  filename=None
  dialog=gtk.FileChooserDialog(title="Select a File",
    action=gtk.FILE_CHOOSER_ACTION_SAVE,
    buttons=(gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL, gtk.STOCK_SAVE,
    gtk.RESPONSE_OK))

  if (text_filter != None) and (all_filter != None):
    dialog.add_filter(text_filter)
    dialog.add_filter(all_filter)
  response = dialog.run()
  if response == gtk.RESPONSE_OK:
    filename = dialog.get_filename()
  elif response == gtk.RESPONSE_CANCEL:
    print 'Cancel Clicked' dialog.destroy()

  if filename != None:
    save_file=open(filename, 'w')
    save_file.write("Sample Data")
    save_file.close()
  print filename
```

The on_save_clicked function starts off by setting the filename to None and quickly sets up the dialog. The dialog title is set to “Select a File”. The action type of the dialog is set to save using gtk.FILE_CHOOSER_ACTION_SAVE. The buttons are set with a tuple. The button uses the stock cancel using the gtk.RESPONSE_CANCEL and the stock save button that uses the gtk.RESPONSE_OK when it is clicked.

After this the function checks to see if there are any filters that should be applied and if so it applies them.

After the filters are added, the dialog is run with its return value assigned to the variable response.

```
response = dialog.run()
```

It then checks the value of response to be of gtk.RESPONSE_OK and if so assigns the name of the file to the variable filename using:

```
filename = dialog.get_filename()
```

If the response is set to gtk.RESPONSE_CANCEL, no actions are taken.

The last action to take with the dialog is to call the destroy method. If the destroy method is not called the dialog will stay on the screen.

```
dialog.destroy()
```

The final part of the on_save_clicked function is to save the string "Sample Data" to the file that was specified to save to.

The on_open_clicked function is very similar to the on_save_clicked function. Instead of opening a dialog to save a file it opens a dialog to select a file for the application to load.

```
def on_open_clicked(widget, text_filter=None, all_filter=None):
  filename=None
  dialog=gtk.FileChooserDialog(title="Select a File",
    action=gtk.FILE_CHOOSER_ACTION_OPEN,
    buttons=(gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL,
    gtk.STOCK_OPEN, gtk.RESPONSE_OK))

  if (text_filter != None) and (all_filter != None):
    dialog.add_filter(text_filter)
    dialog.add_filter(all_filter)

  response = dialog.run()
  if response == gtk.RESPONSE_OK:
    filename = dialog.get_filename()
  elif response == gtk.RESPONSE_CANCEL:
    print 'Cancel Clicked'

  dialog.destroy()
  print "File Choosen: ", filename
```

Just like in the on_save_clicked function the on_open_clicked starts off by setting the filename to None. Then it sets up the open dialog using the gtk.FileChooserDialog. It sets the dialog title to "Select a File", the action to open with gtk.FILE_CHOOSER_ACTION_OPEN. The buttons for the dialog are set as a tuple with the button type and button response next to each other. It sets a cancel button with gtk.STOCK_CANCEL with a response of gtk.RESPONSE and open button with gtk.STOCK_OPEN with a response of gtk.RESPONSE_OK.

After it checks to see if there are filters set and if so adds filters to the dialog using the add_filter method.

The dialog is run using the run method and assigns the response to the variable response like so:

```
response = dialog.run()
```

The on_open_clicked function then checks the value of the response variable. If the response is gtk.RESPONSE_OK the file name is set by using the dialogs get_filename() method.

```
filename = dialog.get_filename()
```

If the response is gtk.RESPONSE_CANCEL no action is taken. The very last action that is taken is to call the dialogs destroy method.

```
dialog.destroy()
```

If the destroy method is not called the dialog will stay on screen.

### 2.4.2 gtk.FileChooserButton

The gtk.FileChooserButton eases the use of a open file dialog by taking care of the run and destroy code and also provides a button. This is easier than the previous section on the FileChooserDialog.

File Chooser Button

```
def main():
  #file filters used with the filechoosers
  text_filter=gtk.FileFilter()
  text_filter.set_name("Text files")
  text_filter.add_mime_type("text/*")
  all_filter=gtk.FileFilter()
  all_filter.set_name("All files")
  all_filter.add_pattern("*")

  window = gtk.Window(gtk.WINDOW_TOPLEVEL)
  window.set_title("Native Filechooser")
  window.connect("destroy", lambda wid: gtk.main_quit())
  window.connect("delete_event", lambda e1,e2:gtk.main_quit())

  button_open = gtk.FileChooserButton("Open File")
  button_open.add_filter(text_filter)
  button_open.add_filter(all_filter)
  button_open.connect("selection-changed", on_file_selected)

  window.add(button_open)
  window.show_all()

def on_file_selected(widget):
  filename = widget.get_filename()
  print "File Choosen: ", filename

if __name__ == "__main__":
  main()
  gtk.main()
```

This example starts by creating two filter types using the gtk.FileFilter class. One filter for text files and one filter for any type of file. Skip a few lines and a FileChooserButton is created like this:

```
button_open = gtk.FileChooserButton(“Open File”)
```

To retrieve the selected file from a FileChooserButton it must connect the *selection-changed* signal to a function. So this example connects the selection-changed signal to the on_file_selected function. The on_file_selected function retrieves the filename that was choosen and then prints it.

### 2.4.3 Windows File Chooser

The native GTK filechoosers are generally ok, but they are very ugly if the GTK application is running on Windows. For PyGTK apps that are running on Windows the option exists to use a native Windows file chooser dialog. The following example will show how to open a file and to save a file. This example will require that the pywin32 package be installed[4].

First off the os, win32con, and win32gui modules will need to be imported along with the pygtk and gtk modules.

```
import os
import win32gui, win32con
```

Like all the other examples about file choosers the Windows file chooser will start off with some GUI code.

```
def main():
  file_filter="""Text files\0*.txt\0All Files\0*.*\0"""

  window = gtk.Window(gtk.WINDOW_TOPLEVEL)
  window.set_title("Windows Filechooser Example")
  window.connect("destroy", lambda wid: gtk.main_quit())
  window.connect("delete_event", lambda e1,e2:gtk.main_quit())

  button_save = gtk.Button("Save File")
  button_open = gtk.Button("Open File")
  button_save.connect("clicked", on_save_clicked, file_filter)
  button_open.connect("clicked", on_open_clicked, file_filter)

  hbox = gtk.HBox(True, 0)
  hbox.pack_start(button_save, True, True, 5)
  hbox.pack_start(button_open, True, True, 5)

  window.add(hbox) window.show_all()
```

[4] See section C on page 175 for instructions on using PyGTK on Windows for more information. Or just go to http://sourceforge.net/projects/pywin32/files/ and download and install it.

First thing that is done is to create a file filter that will be used with the open and save dialogs. The file filter is in the form of "Display Name, Seperator, File Type, Seperator, Display Name, Seperator, File Type, Seperator" and looks like this:

```
file_filter="""Text files\0*.txt\0All Files\0*.*\0"""
```

The GUI creates one button to launch the save dialog and one to launch the open dialog. The button called button_save is clicked it will call the on_save_clicked function passing along the file filter. When the button called button_open is clicked, it will call the on_open_clicked function passing along the file filter.

The on_save_clicked and on_open_clicked function are very similar in form with some minor differences. Here is the on_save_clicked function.

```
def on_save_clicked(widget, file_filter=None):
  filename=None
  try:
    filename, customfilter, flags=win32gui.GetSaveFileNameW(
      InitialDir=os.path.join(os.environ['USERPROFILE'],"My Documents"),
      Flags=win32con.OFN_ALLOWMULTISELECT|win32con.OFN_EXPLORER, File='',
      DefExt='txt', Title='Save a File', Filter=file_filter, FilterIndex=0)
  except win32gui.error:
    print "Cancel clicked"

  print filename
  if filename != None:
    save_file = open(filename, 'w')
    save_file.write("Test Save Data")
    save_file.close()
  return filename
```

This is a simple funtion that takes a file filter as an argument and sets it as the filter for Windows save dialog. To use and display the save dialog the win32gui.GetSaveFileNameW function is used. Arguments that are used with it include Initial Directory, Flags, File, Default Extention, Title, File Filter, and FilterIndex. As can be seen the inital directory is set to the users My Documents folder. Flags are set to allow multiple selection. The default extention type is txt. When it is called it must be done by assigning its return value to three variables; filename, customfilter, flags.

The GetSaveFileNameW function must be used with exception handling as it will through an exception if the cancel button is clicked. So this example catches win32gui.error exceptions and prints the message "Cancel clicked" instead of crashing.

If a file has been selected to save this example saves it with the string "Test Save Data".

The GetOpenFileNameW function is used to select and open file on Windows. It is very simliar to the GetSaveFileNameW function covered above. Here is the on_open_clicked function that uses the Windows open dialog.

```
def on_open_clicked(widget, file_filter=None):
```

```
filename=None
try:
filename, customfilter, flags=win32gui.GetOpenFileNameW(
  InitialDir=os.path.join(os.environ[’USERPROFILE’],"My Documents"),
  Flags=win32con.OFN_ALLOWMULTISELECT|win32con.OFN_EXPLORER, File=”,
  DefExt=’txt’, Title=’Select a File’, Filter=file_filter, FilterIndex=0)
except win32gui.error:
  print "Cancel clicked"
print ’open file names:’, filename
return filename
```

The GetOpenFileNameW functions takes as arguments Intial Directory, Flags, File, Default Extention, Title, File Filter, and Filter Index. As can be seen the inital directory is set to the users My Documents folder. Flags are set to allow multiple selection. The default extention type is txt. When calling this function the return value must be assigned to three variables; these being the filename, customfilter, and flags.

Like the save GetSaveFileNameW the GetOpenFileNameW function requires that it used with exception handling as it will give win32gui.error if the cancel button is pressed. If everything goes as planed the function should continue to the end where it prints the message “open file names: filename”.

## 2.5 Glade 3

Glade is a program that allows the creation of the user interface graphical. Windows and dialogs can be created. Widgets can be dragged and dropped into place. Names assigned to widgets, callback functions assigned. All this is saved to a xml file with an extension of .glade.

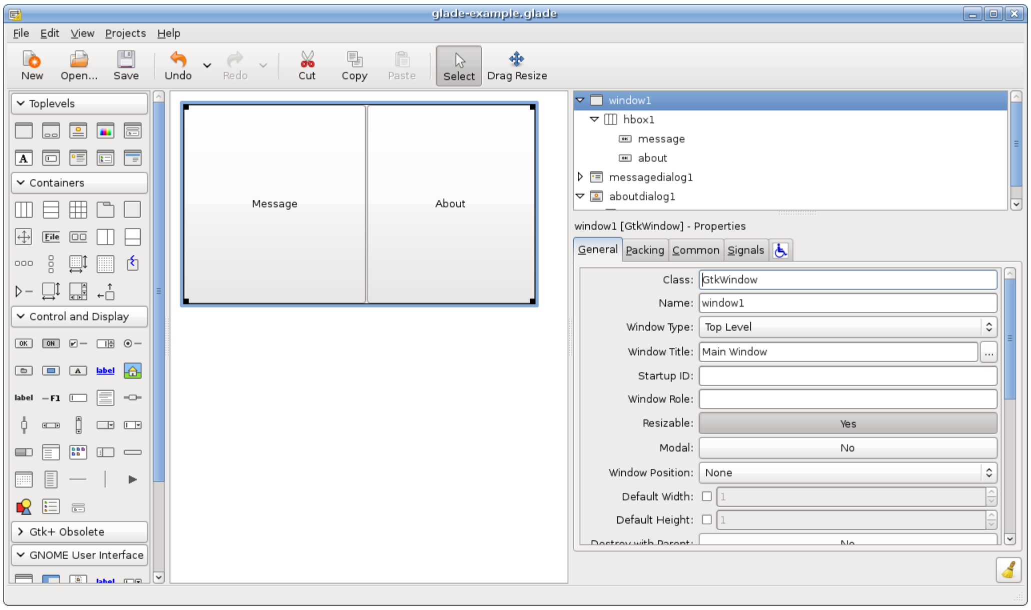
Docked on the left side of glade is the palette. The palette contains the top level elements such as:

- windows (gtk.Window)
- dialogs (gtk.Dialog etc...)

Under the Toplevels is are the Containers. The containers contain:

- Horizontal Box (gtk.HBox)
- Vertical box (gtk.VBox)
- Table (gtk.Table)
- Notebook (gtk.Notebook)
- Frame (gtk.Frame)
- etc...

After and under the Containers are the Control and Display widgets, they contain:



window1
hbox1
message
about
messagedialog1
aboutdialog1

window1 [GtkWindow] - Properties

| | |
|---|---|
| Class: | GtkWindow |
| Name: | window1 |
| Window Type: | Top Level |
| Window Title: | Main Window |
| Startup ID: | |
| Window Role: | |
| Resizable: | Yes |
| Modal: | No |
| Window Position: | None |
| Default Width: | □ 1 |
| Default Height: | □ 1 |
| Destroy with Parent: | No |

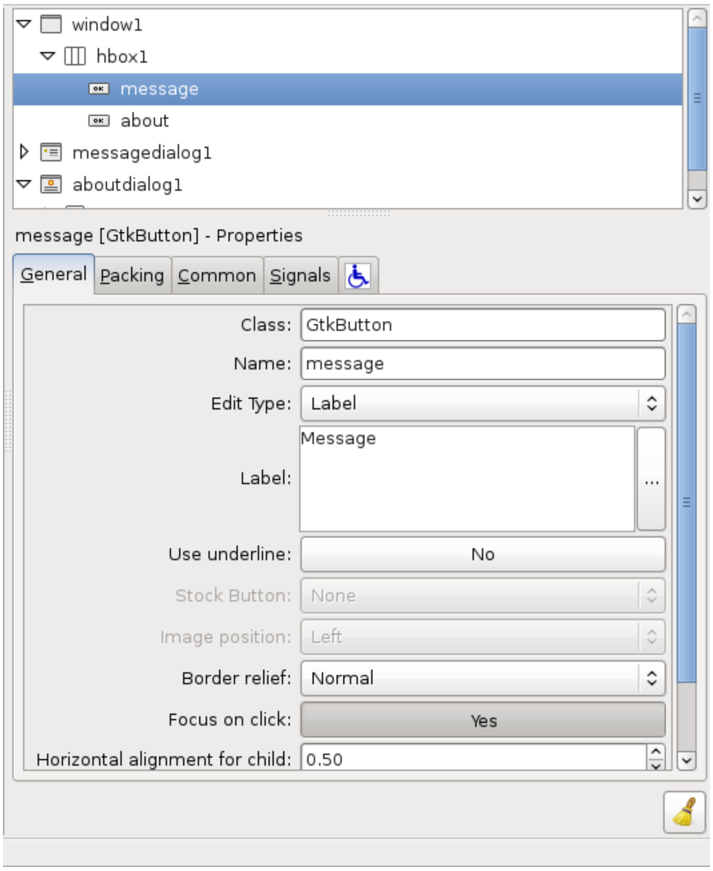
Figure 2.1: Basic Glade User Interface Designer

- Button (gtk.Button)
- Toggle Button (gtk.ToggleButton)
- Check Button (gtk.CheckButton)
- Spin Button (gtk.SpinButton)
- Raido Button (gtk.RadioButton)
- etc...

To create a simple application, from the Toplevels select and add a Window. Next select a Horizontal Box and add it to the Window. When prompted for how many items, select two. When this is done the window will be split in half horizontally with a line going down through the center (see figure 2.1). Each of these can hold a widget.

Next add two buttons from the container. The one on the left label Message and the one on the right label About. Also change the names to message and about. To do this click the first button. On the right hand side the editor should change for a button type (see figure 2.2 on the next page). As can be seen in figure2.2; the class is of type GtkButton, the name is set to message meaning that when it is called with PyGTK it uses the name message. For the Label it is set to Message. The label is what is displayed to the user as the button text.

Once the buttons have been added and setup with the names and labels then the signals that are to be caught should be added (see figure 2.3 on page 49). To add signal methods to the buttons first select the message button. Then in the editor window select the Signals tab.

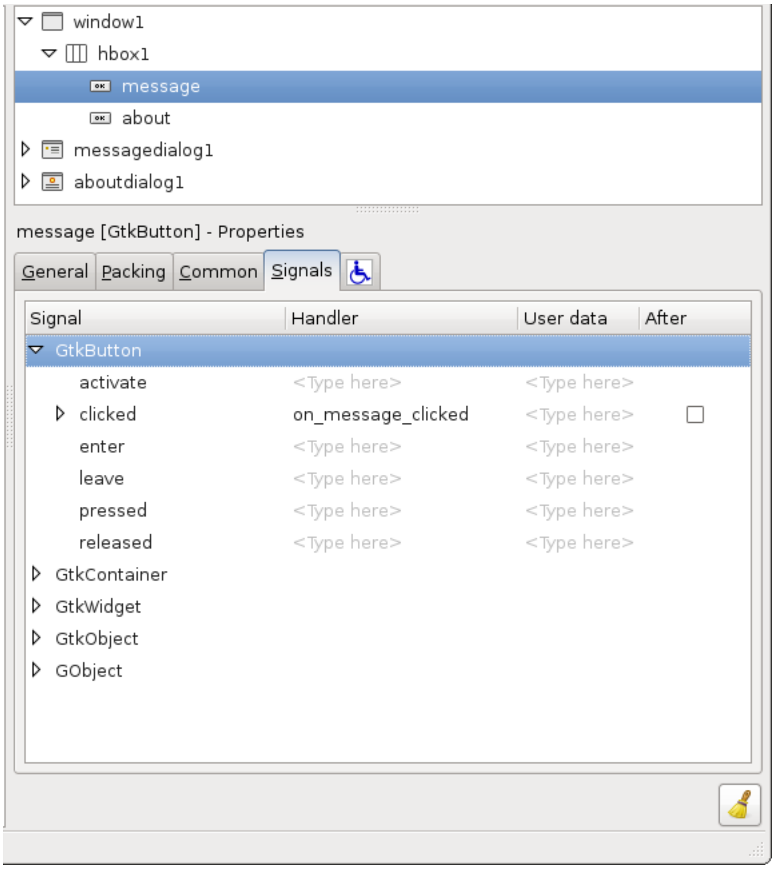


window1
hbox1
message
about
messagedialog1
aboutdialog1

message [GtkButton] - Properties

Class: GtkButton
Name: message
Edit Type: Label
Label: Message
Use underline: No
Stock Button: None
Image position: Left
Border relief: Normal
Focus on click: Yes
Horizontal alignment for child: 0.50

Figure 2.2: Glade Editor with Button



message [GtkButton] - Properties

| Signal | Handler | User data | After |
|---|---|---|---|
| GtkButton | | | |
| activate | <Type here> | <Type here> | |
| clicked | on_message_clicked | <Type here> | □ |
| enter | <Type here> | <Type here> | |
| leave | <Type here> | <Type here> | |
| pressed | <Type here> | <Type here> | |
| released | <Type here> | <Type here> | |
| GtkContainer | | | |
| GtkWidget | | | |
| GtkObject | | | |
| GObject | | | |

Figure 2.3: Signal Handler Specified

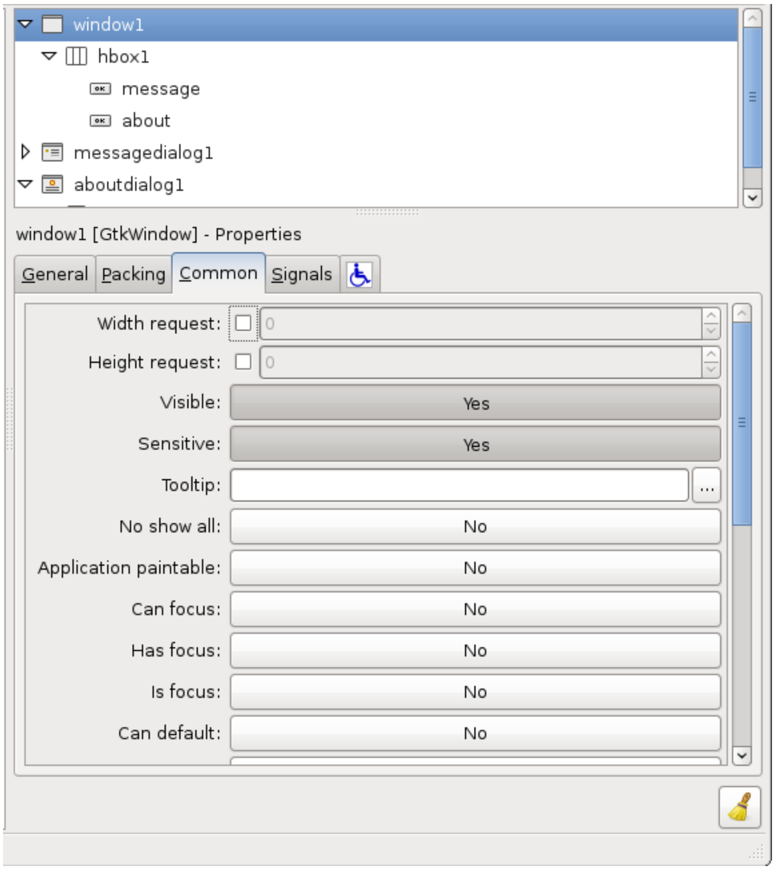
Under GtkButton there will be a signal called *clicked*. For clicked add a handler. If the handler space is clicked it will provide a default list to choose from. To see what it should look like look at figure 2.3. What is typed as the Handler is the function or method in the python code that will be called.

Now that the buttons have been added to the main window (whose name is window1) it is time to make sure that this window is visible. Select the main window and in the editor select the Common tab. Once in the Common tab find the *Visible* option and make sure it is set to *Yes* (see figure 2.4 on the following page).

Now the main window is done. Save your work. Next an about dialog will be added. To add an about dialog it is selected from the Toplevel elements on the palette. Leave it with the default name *aboutdialog1*. The about dialog will be used to show how to interact with more than one window in glade.

A PyGTK program interacts with the created glade file using gtk.glade.

```
import pygtk
```



window1 [GtkWindow] - Properties

Width request: □ 0

Height request: □ 0

Visible: Yes

Sensitive: Yes

Tooltip: ...

No show all: No

Application paintable: No

Can focus: No

Has focus: No

Is focus: No

Can default: No

Figure 2.4: Main Windows Set as Visible

```
pygtk.require('2.0')
import gtk
import gtk.glade

class GladeExample(object):
  def __init__(self):
    self.gladefile = gtk.glade.XML("glade-example.glade")
    self.gladefile.signal_autoconnect(self)
    self.main_window = self.gladefile.get_widget("window1")
    self.about_dialog = self.gladefile.get_widget("aboutdialog1")
    self.message_dialog = self.gladefile.get_widget("messagedialog1")
```

Here the class GladeExample is declared with an intiation method that connects to the glade file that was created. The glade file is loaded using the gtk.glade.XML class. It takes as arguments the glade file and optionally a widget and translation domain.

Then to use a widget as if it was created using PyGTK code it must be retrieved using the get_widget method. The get_widget method works by taking as an argument the name of the widget. In the glade example above the main windows name is window1, the about dialogs name is aboutdialog1, and the message dialog is messagedialog1. As can be seen the main window is assigned to self.main_window and so on with the about and message dialog.

What can be noticed that the buttons that were adding to the glade file to launch the about and message dialog were not assigned with the get_widget method. This is because they were set to automatically call handler functions and do not need to write code for each button to connect them. This is handled with one line of code, self.gladefile.signal_autoconnect(self). This one line will automatically connect any signal handlers that were specified in the glade file without having to write any extra code.

```
  def on_about_clicked(self, widget):
    self.about_dialog.run()
    self.about_dialog.destroy()
```

As was specified with glade, when the about button is clicked, the method on_about_clicked is called. This method displays the about dialog that was created with glade and destroys the dialog when it is closed.

```
  def on_message_clicked(self, widget):
    self.message_dialog.run()
    self.message_dialog.destroy()
```

As was specified with glade, when the message button is clicked, the method on_message_clicked is called. This method displays the message dialog that was created with glade and destroys the dialog when it is closed.

```
  def on_window1_delete_event(self, widget, event):
    gtk.main_quit()
```

the on_window1_delete_event will quite the PyGTK application when the main window(window1) is closed. This to is specified with glade under the main windows Signal tab; GtkWidget –> delete-event.

```
if __name__ == "__main__":
  app = GladeExample()
  gtk.main()
```

And of course a few lines that runs the glade example.

## 2.6 Builder

Builder refers to gtk.Builder which is the future as it is a replacement for gtk.glade. Basically what it is is including support for xml files to build applications inside of GTK itself, unlike glade which is a library. Currently the glade program does not support saving to the Builder format, but it will soon. In the mean time glade files must be converted to Builder files using *gtk-builder-convert*[5]. This program will take a glade xml file and convert it to a Builder xml file.

To convert a glade file to a Builder file the following command is issued:

```
gtk-builder-convert glade-example.glade glade-example.xml
```

Now instead of using gtk.glade.XML to access this new builder xml file, gtk.Builder is used as shown here.

```
builder = gtk.Builder()
builder.add_from_file(“glade-example.xml”)
```

Also instead of using get_widget like in the glade example (see 2.5), the method *get_object* is used.

```
main_window = builder.get_object(“window1”)
about_dialog = builder.get_object(“aboutdialog1”)
message_dialog = builder.get_object(“messagedialog1”)
```

With this done, the widgets can be used as if they were programmed normally with PyGTK.

To auto connect the signals like is avialbe using glade the following code is used.

```
builder.connect_signals(self)
```

Remember this needs to be done from within a class.

[5] For more information on gtk-builder-convert visit: http://library.gnome.org/devel/gtk/2.12/gtk-builder-convert.html.

Also if you plan on using gtk-builder-convert, gtk development files must be installed to have it installed. This is accomplished on Ubuntu by installing libgtk2.0-dev.

## 2.7 Loading Images

To load an image with PyGTK an instance of the gtk.Image class must be created. With this becomes available several methods for loading different types of images. This example will cover loading images from file and from the GTK stock images.

Loading Images

```
import pygtk, gtk
def main():
  win = gtk.Window()
  win.connect("delete_event", lambda w,e: gtk.main_quit())
  vbox = gtk.VBox(False, 0)

  image1 = gtk.Image()
  image1.set_from_stock(gtk.STOCK_DIALOG_INFO, gtk.ICON_SIZE_DND)

  image2 = gtk.Image()
  image2.set_from_file("flower.jpg")

  vbox.pack_start(image1, False, False, 5)
  vbox.pack_start(image2, False, False, 5)
  win.add(vbox)
  win.show_all()

if __name__ == "__main__":
  main()
  gtk.main()
```

This example creates a window with a gtk.VBox and adds two images. The first image is set from stock gtk images created with the set_from stock method. The set_from_stock method requires a GTK stock image and a stock size. The stock types available can be found in the appendix ( on page 177). The stock sizes include:

- gtk.ICON_SIZE_MENU
- gtk.ICON_SIZE_SMALL_TOOLBAR
- gtk.ICON_SIZE_LARGE_TOOLBAR
- gtk.ICON_SIZE_BUTTON
- gtk.ICON_SIZE_DND
- gtk.ICON_SIZE_DIALOG

The second image is loaded using set_from_file method. All this method requires is location on the computer to the image.

All that needs to be done once the images are loaded is add them to a widget. In this example they are added to gtk.VBox.

There are many different methods for loading images and they can be found at the PyGTK reference site[6].

## 2.8 Tooltips

A tooltip is used to display useful information to the screen a user puts a mouse over a widget such as label or button. To use a simple tooltip requires only on method call on the widget: set_tooltip_text

```
label = gtk.Label("Display Tooltip")
label.set_tooltip_text("This is a Tooltip")
```

When a mouse is placed over this label a tooltip will display the text "This is a Tooltip". Very simple to use and there is nothing more to be said on that.

For more fancy tooltips a custom tooltip must be created. To do this the has_tooltip property must be set to True. Then the widget that is to display the custom tooltip must connect to the query-tooltip signal. For example, the callback function can create a new tooltip by creating an gtk.HBox that holds an image and text then use set_custom on the tooltip to use this hbox.

Here is an example.

```
fancy_label = gtk.Label("A fancy Tooltip")
fancy_label.props.has_tooltip = True
fancy_label.connect("query-tooltip", on_query_tooltip)
```

So this creates a label, sets the tooltip to true using fancy_label.props.has_tooltip property, and then connects the query-tooltip signal to the function on_query_tooltip.

Here is an example of the on_query_tooltip function. This function creates a label and an image that is displayed instead of plain text.

```
def on_query_tooltip(widget, x, y, keyboard_tip, tooltip):
  hbox = gtk.HBox()
  label = gtk.Label('Fancy Tooltip with an Image')
  image = gtk.Image()
  image.set_from_stock(gtk.STOCK_DIALOG_INFO, gtk.ICON_SIZE_DND)
  hbox.pack_start(image, False, False, 0)
  hbox.pack_start(label, False, False, 0)
  hbox.show_all()
  tooltip.set_custom(hbox)
  return True
```

As can be seen this creates a gtk.HBox to hold a label and an Image. It then uses the tooltip argument to set it to a custom tooltip. A custom tooltip can be anything but this example has kept it simple for understandability sake. For more tooltip options visit the PyGTK tooltip reference page[7].

[6] The PyGTK image class can be found at: http://www.pygtk.org/docs/pygtk/class-gtkimage.html

[7] The PyGTK tooltip reference page can be found at: http://www.pygtk.org/docs/pygtk/class-gtktooltip.html

## 2.9 Summary

This section is not yet written :)

# Chapter 3

# Cairo

Please send any fixes or suggestions to peter@majorsilence.com or leave a comment at http://www.majorsilence.com/pygtk_book.

## 3.1 Introduction

Welcome to the chapter on cairo. What is cairo? Cario is a powerful 2d graphics library that lets you output to many different surfaces. Surfaces that are supported include image surfaces (png) pdf, postscript, win32, svg, quartz, and xlib. What all these different surfaces achieve will be discussed throughout the chapter; however every surface type here supports writing to png. Besides including png write support, cairo also includes png import support.

While reading about cairo in other sources you may find that it is suggested to think of cairo is as a canvas that you will paint on. This kind of works for me. You have the canvas that you can put different layers of paint on that when combined and finished produces your final output. But that is about as far I will be using this metaphor in this chapter.

Things that cairo can be used for include creating graphics, combining work, doing layout for printing, or even creating reports as pdf documents. Really the only limitation to cairo is your imagination.

Dig in and see what you can learn.

## 3.2 Basics

The first example with cairo will be some simple drawing. It will create a surface and draw a line saving to a image file. When working with cairo it must be remembered that the *cairo* module is needed and must be imported.

```
import cairo

WIDTH, HEIGHT = 400, 400

# Setup Cairo
```

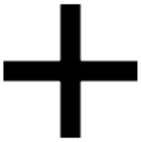


Figure 3.1: Two Straight Lines

```
surface = cairo.ImageSurface(cairo.FORMAT_ARGB32, WIDTH, HEIGHT)
context = cairo.Context(surface)

# Set thickness of brush
context.set_line_width(15)
# Draw Vertical Line
context.move_to(200, 150)
context.line_to(200, 250)
# Draw horizontal line
context.move_to(150, 200)
context.line_to(250, 200)
context.stroke()

# Output a PNG file
surface.write_to_png("cairo-draw-line1.png")
```

This example creates an ImageSurface with a width and height of 400. The ImageSurface is set to use the cairo.FORMAT_ARG32 (See below for details). The context is what actually keeps track of everything that is done to the surface and is used to draw. It is used to control how the drawing operations are used.

With a context set it is now possible to draw or perform other actions. First thing that is done is to set the line width to 15 using the *context.set_line_width(15)* method. The *set_line* method sets the width of a line for a context.

Next, using the contexts move_to method moves the position of the brush to the position specified; which in the line is x position 200 and y position 150. X coordinates are measured from the left most part of the surface. Y coordinates are measured from the top most part of the surface.

Using the contexts *line_to* method will draw a line from the current position, that was specified with the move_to method, to the new position specified with the line_to method. To display what has been drawn with the *line_to* method the contexts *stroke* method must be called. Once *context.stroke()* is called then the lines are actually applied to the surface.

See figure 3.1to see what the output should look like.

### 3.2.1 Cairo Surface Format

There are four surface options available[1] and they are:

cairo.FORMAT_ARGB32 - each pixel is a 32-bit quantity, with alpha in the upper 8 bits, then red, then green, then blue. The 32-bit quantities are stored native-endian. Pre-multiplied alpha is used. (That is, 50% transparent red is 0x80800000, not 0x80ff0000.)

cairo.FORMAT_RGB24 - each pixel is a 32-bit quantity, with the upper 8 bits unused. Red, Green, and Blue are stored in the remaining 24 bits in that order.

cairo.FORMAT_A8 - each pixel is a 8-bit quantity holding an alpha value.

cairo.FORMAT_A1 - each pixel is a 1-bit quantity holding an alpha value. Pixels are packed together into 32-bit quantities. The ordering of the bits matches the endianess of the platform. On a big-endian machine, the first pixel is in the uppermost bit, on a little-endian machine the first pixel is in the least-significant bit.

In most cases cairo.FORMAT_ARGB32 or cairo.FORMAT_RGB24 will be used.

### 3.2.2 Cairo Surfaces

cairo.ImageSurface(cairo.FORMAT_ARGB32, WIDTH, HEIGHT) - Use to render to memory buffers.

cairo.PDFSurface("drawings.pdf", WIDTH, HEIGHT) - Renders to the specified PDF file.

cairo.PSSurface("drawings.ps", WIDTH, HEIGHT) - Renders to the specified Postscript file.

cairo.SVGSurface("drawings.svg", WIDTH, HEIGHT) - Renders to the specified SVG file.

## 3.3 Drawing Context

As discussed above, a context is what allows the programmer to use the cairo surface. This section will discover different uses of the context class by making use of several different examples. For a list of the context methods used in this section please skip ahead to 3.3.4 on page 65.

### 3.3.1 Paths: Lines, Curves, Arcs

To start off this section lets take a look at line drawing again, but using it to draw more than two straight lines. This example will use the *line_to* method to create a rectangle and triangle. It will also use a new method, *arc*, to create a circle. Along with with these two methods, the color of the context will be set using the *set_source_rgb* value. These methods set points that are then used to create a path.

[1] The list of formats available are taken from the cairo website and can be found at: http://www.cairographics.org/manual/cairo-Image-Surfaces.html

This example starts by calling the *main* function. Inside the main function it creates a cairo ImageSurface with an alpha RGB format and a width and height of 400. It then creates a context from this surface. The ImageSurface renders to a memory buffer and not an image. To save to an image the surface must call the *write_to_png* method that is available to all surface types.

Next it sets the line with of the context to 15. Immediately after this it calls the draw_rectangle, draw_triangle, and draw_circle functions. These are functions that are defined in this example and are not cairo builtin methods. While cairo contexts do have a builtin method to create rectangles, this example is doing it manually just to show how to use the line_to method in different ways.

Cairo Context Basics

```
#!/usr/bin/env python
import cairo
import math

def draw_rectangle(context=None):
  x1, y1 = 25, 150 # top left corner
  x2, y2 = 25, 250 # bottom left corner
  x3, y3 = 125, 250 # bottom right corner
  x4, y4 = 125, 150 # top right corner

  context.set_source_rgb(1.0, 0.0, 0.0) # red
  context.move_to(x1, y1)
  context.line_to(x2, y2)
  context.line_to(x3, y3)
  context.line_to(x4, y4)
  context.close_path()
  context.stroke()
```

The draw_rectangle function starts off by defining four corners that will make up the rectangle.

These four x and four y coordinates create the four corners of the rectangle. Top left, bottom left, bottom right and the top right corners. To draw a rectangle the function first uses the *move_to(x1, y1)* method on the context to move the starting position to the first corner. Then it uses the *line_to(x2, y2)* method to create a line from the first corner to the second. Then it again uses the *line_to* method with x3 and y3 to create a line from the second corner to the third corner. And last with the *line_to* method it creates a line from the third to the fourth corner.

Now if you follow that lines that were created, you will notice only the left side, bottom, and right side where drawn, but all four corners were used. The line_to method could be used again to draw a line from x4 and y4 to x1 and y1, but instead the *close_path* method is used. The close path method will draw a line from the current position to the first position (since the last time the *stroke* method was called).

Also in the draw_rectangle function the context is set to draw these lines in red using the set_source_rgb(red, green, blue) method. This method takes 3 variables each with a value between 0.0 and 1.0, with 0.0 being none of that color and 1.0 being a solid color. The lower the value, the higher the opacity.

```
def draw_triangle(context=None):
  context.set_source_rgb(0.0, 1.0, 0.0) # green
  context.move_to(275, 175)
  context.line_to(375, 375)
  context.rel_line_to(-200, 0)
  context.close_path()
  context.stroke()
```

The draw_triangle method is similar to the draw_rectangle function in that it also uses the move_to, line_to and close_path methods. But it also uses the *rel_line_to* method; this method stand for relative_line_to, and moves to a new position based on the current location instead of using the absolute value of the surfaces width and height.

Like in the rectangle function, the triangle function sets the color (to green)

Next it starts by moving the starting coordinate to x coordinate 275 and y coordinate 175. Then draws a line to x 375 and y 375.

After this instead of drawing based on absolute coordinates of the surface width and height, it uses the *rel_line_to* method to draw from x 375 and 375. It uses -200 x which moves from 375 to 175 and moves 0 from y. This means there is a line drawn from (375, 375) to (175, 375).

Finally it closes the path and uses the *stroke* method to apply the lines to the surface.

```
def draw_circle(context=None):
  width, height = 100, 100
  radius = min(width, height)
  context.set_source_rgb(0.0, 0.0, 1.0) # blue
  context.arc(275, 100, radius / 2.0 - 20, 0, 2 * math.pi)
  context.stroke()
```

The draw_circle function introduces the a new method; *arc*.

The start_angle and stop_angle are specified in radians. If you do not know how to work with radians take a look at section 3.3.1.1 on the next page. Here the start angle is set to 0. The stop_angle is set to 2 * math.pi, which is 360 degrees. This arc therefore forms a full circle.

Other parts of the arc method is the x and y coordinate positions for the center of the arc. After the x and y coordinates come the radius of the arc.

```
def draw_curve(context=None):
  context.set_source_rgb(0.5, 0.0, 0.3)
  context.move_to(20, 20)
  context.curve_to (60, 100, 100, 20, 140, 100)
  context.stroke()
```

The draw_curve function is used to draw a cubic Bézier spline from the current position to x3 and y3, using x1, x2, y1, y2 as control points. If no current position is set, x1 and y1 are used as the starting position. This is accomplished using the curve_to method. The curve_to method is defined as context.curve_to(x1, y1, x2, y2, x3, y3).

```
def main():
  surface = cairo.ImageSurface(cairo.FORMAT_ARGB32, 400, 400)
```

```
    context = cairo.Context(surface)
    context.set_line_width(15)
    draw_rectangle(context)
    draw_triangle(context)
    draw_circle(context)
    draw_curve(context)

    surface.write_to_png("cairo-basics.png")

  if __name__ == "__main__":
    main()
```

#### 3.3.1.1 Radians and Degrees

If you do not know how to work with radians you are in luck, as it is very simple.

```
radians=degree*(math.pi/180)
```

You can also use.

```
radians = math.radians(degree)
```

If you want to know what the degrees of a radian is that is simple as well. Switch the degree with the radian and divide 180 by PI.

```
degree=radians*(180/math.pi)
```

You can also use.

```
degree = math.degrees(radian)
```

### 3.3.2 Text

Drawing text is with cairo is the same as drawing a line or an arc but using some specific functions for text. Start off like any other cairo application setting the type of surface and setup a context.

```
import cairo
text = "Hello to the Great Text."

surface = cairo.ImageSurface(cairo.FORMAT_ARGB32, 800, 75)
context = cairo.Context(surface)
context.set_source_rgb(0.0, 0.0, 0.0) # set to black
```

What is then needed is to set the type of font and its size. Here a Monospace font is set with a normal slant and is set to be bold (see section 3.3.2.1 on the facing page for more styles). The size of the font is set to 50.

```
context.select_font_face("Monospace", cairo.FONT_SLANT_NORMAL,
    cairo.FONT_WEIGHT_BOLD)
context.set_font_size(50)
```

Using the context that was created it is possible to retrieve information on the text that is being used with the text_extents method.

```
x_bearing, y_bearing, width, height = context.text_extents(text)[:4]
```

Last is to move the context to the location that it should be drawn. Here the text is set to draw a X coordinate 5 and at a Y coordinate that is is the height of the text. To apply the text the show_text method is now called. This method adds text to the cairo context. To show the text the stroke method is called. To finish off it is saved to a file called cairo-draw-text1.png.

```
context.move_to(5, height)
context.show_text(text)
context.stroke()
surface.write_to_png("cairo-draw-text1.png")
```

#### 3.3.2.1 Font Styles

There are more than two types of font face styles available with cairo; there are five.

- cairo.FONT_SLANT_ITALIC
- cairo.FONT_SLANT_NORMAL
- cairo.FONT_SLANT_OBLIQUE
- cairo.FONT_WEIGHT_BOLD
- cairo.FONT_WEIGHT_NORMAL

### 3.3.3 Antialias

First lets define antialias so there is no confusion.

**Antialias:** Is the technique of minimizing the distortion artifacts created while drawing.

But what does this mean? Basically nothing if a straight line is being drawn. However if a curve or arc is being drawn it will look distorted or jagged, not very smooth at all. However with antialiasing turned on it will look smooth by setting the color correctly around the edges. The best way to understand this is to view an image. Take a look at figure 3.2 and see if you can tell the difference.

Now the question is why would you want to turn off antialiasing? I cannot think of to many reasons, but one that I can think of is for the program DeVeDe. It is a GUI application that uses a few command line applications to create DVDs from video files.

One of the programs that DeVeDe uses is dvdauthor. One of the functions of dvdauthor is to create dvd menus. And one part of the menu system is not able to handle more than four colors

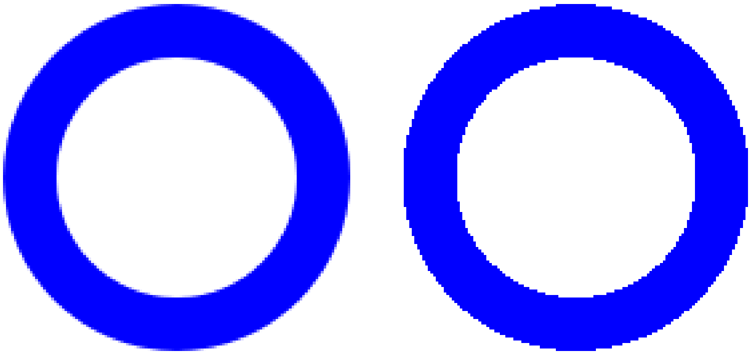


Figure 3.2: Antialias Example - As can be seen the circle on the left uses the default cairo antialias while the circle on the left turns antialias off. As can be seen when antialias is turned off the curves become jagged/distorted.

in an image including the alpha channel. With antialiasing turned on it will output images with many colors, because to make a curve look smooth it uses different shades of the color being used. However if antialias is set to none the image created with cairo will only have the colors specified and will be able to be used with dvdauthor.

#### 3.3.3.1 Changing Antialias

To change the default antialias the contexts set_antialias method is used.

```
context.set_antialias(Antialias Type)
```

To find out what the current setting is just use the context get_antialias() method.

The example below sets up a normal cairo surface and context. It then draws two circles. The first circles draws with the default antialias, which is cairo.ANTIALIAS_DEFAULT, and the second circle is drawn with antialias turned off.

```
import cairo, math

def draw_circle(context, xc, yc):
  radius = 150
  context.set_source_rgb(0.0, 0.0, 1.0)
  context.arc(xc, yc, radius / 2.0 - 20, 0, 2 * math.pi)
  context.stroke()
```

```
if __name__ == "__main__":
  surface = cairo.ImageSurface(cairo.FORMAT_ARGB32, 300, 200)
  context = cairo.Context(surface)
  context.set_line_width(20)

  draw_circle(context, 75, 100)
  context.set_antialias(cairo.ANTIALIAS_NONE)
  draw_circle(context, 225, 100)
  surface.write_to_png("cairo-antialias.png")
```

To turn off antialias, the context set_antialias method must be given the cairo.ANTIALIAS_NONE type. To see what this looks like take a look at figure 3.2.

#### 3.3.3.2 Antialias Types

The four options available for antialias are:

- cairo.ANTIALIAS_DEFAULT
- cairo.ANTIALIAS_GRAY
- cairo.ANTIALIAS_SUBPIXEL
- cairo.ANTIALIAS_NONE

### 3.3.4 Context Methods

**set_source_rgb(R,G,B)** This allows setting the color value of the context

**rel_curve_to(x1,y1,x2,y2,x3,y3)** Create a curve instead of a straight line from the current position to x3 and y3, using x1/y1 and x2/y2 as control point. Where x1, x2, x3, y1, y2, y3 are relative to the current position.

**curve_to(x1,y1,x2,y2,x3,y4)** Create a curve instead of a straight line from the current position to x3 and y3, using x1/y1 and x2/y2 as control point

**rel_line_to(x,y)** Draw a line relative to the current position with an offset of x and of y

**line_to(x,y)** Draw a line from the current position to the new position

**rel_mov_to(x,y)** Move the position relative to the current position

**move_to(x,y)** Move by an absolute position

**set_font_size(size)** set the size of the font

**arc** Draw an arc

**fill** Color the path that as been set with rectangle or line_to with the color that has been set

**rectangle(x1,y1,x2,y2)** Draw a rectangle

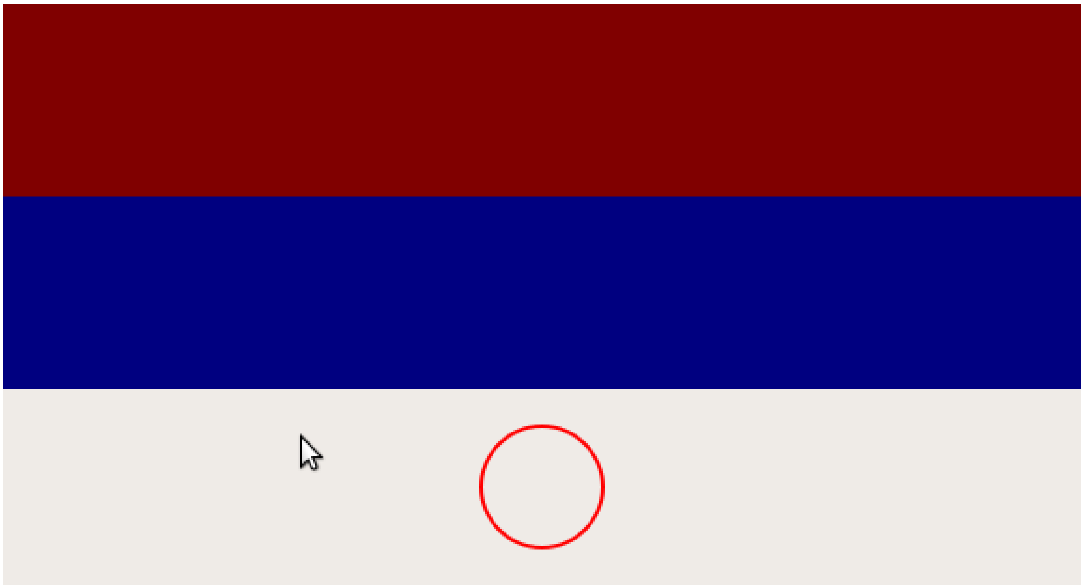


Figure 3.3: Custom PyGTK widget with Cairo

**set_antialias(type)** Set the the type of antialias that is to be used

**close_path** Draw a line from the starting position since the last time stroke was called from the current position, thus closing the path

## 3.4 Cairo and PyGTK

Cairo can be used with PyGTK by creating a custom widget. The custom widget discussed here will extend the gtk.DrawingArea class and override[2] the expose_event signal callback method; *do_expose_event*.

```
class CairoGtkOverride(gtk.DrawingArea):
  __gsignals__ = {"expose_event": "override" }

  def __init__(self):
    gtk.DrawingArea.__init__(self)

  def do_expose_event(self, event):
    context = self.window.cairo_create()
    context.rectangle(event.area.x, event.area.y,
        event.area.width, event.area.height)
```

[2] Take a look at http://www.sicem.biz/personal/lgs/docs/docs/gobject-python/gobject-tutorial.html for a tutorial on creating custom properties and signals. Overriding signals is also covered.

```
    context.clip()

    self.draw(context, *self.window.get_size())

  def draw(self, context, width, height):
    context.set_source_rgb(0.5, 0.0, 0.0)
    context.rectangle(0, 0, width, height)
    context.fill()
```

To properly override a signal in PyGTK set the class variable __gsignals__ to override the expose_event signal. In the __init__ method the class initiates its base class.

The *do_expose_event* method is the callback for the expose_event signal. It sets up a cairo context, creates a rectangle to the size of the widget. It uses the event to retrieve the size that is needed; *event.area.x* and *event.area.y* are the starting x and y coordinates while *event.area.width* and *event.area.height* are the width and height of the widget. Then the widget is set to only draw to the size of the rectangle using *context.clip()*. The last part is to call the classes *draw* method on every expose event.

The draw method takes as arguments a cairo context and a width and height of the widget. The draw method is where you can use cairo just as if it were not with PyGTK. The draw method in CairoGtkOverride draws a red rectangle.

Now that a custom widget class has been created it can be extend as much as is wanted and the draw method overwritten to draw what is desired.

```
class Circle(CairoGtkOverride):
  def draw(self, context, width, height):
    context.set_source_rgb(1.0, 0.0, 0.0)
    radius = min(width, height)
    context.arc(width / 2.0, height / 2.0,
      radius / 2.0 - 20, 0, 2 * math.pi)
    context.stroke()
```

The above code extend the gtk custom widget class that was created further up and draws a circle instead of a red rectangle.

To run the code just add these widgets to your PyGTK application the same way you would any other widget.

```
if __name__ == "__main__":
  win = gtk.Window()
  win.connect("delete-event", gtk.main_quit)
  vbox = gtk.VBox()

  override_widget = CairoGtkOverride()
  circle_widget = Circle()

  vbox.pack_start(override_widget, True, True, 0)
  vbox.pack_start(circle_widget, True, True, 0)
```

```
    win.add(vbox)
    win.show_all()
    gtk.main()
```

## 3.5 Summary

For more examples on PyGTK and cairo you can take a look at the following resources:

- http://blog.eikke.com/index.php/ikke/2007/02/17/python_cairo_xshape_and_clocks
- http://ralph-glass.homepage.t-online.de/clock/readme.html
- http://ralph-glass.homepage.t-online.de/shogi/readme.html
- http://www.cairographics.org/pycairo/resources/
- http://www.tortall.net/mu/wiki/CairoTutorial
- http://www.tortall.net/mu/wiki/PyGTKCairoTutorial
- http://www.pygtk.org/articles/cairo-pygtk-widgets/cairo-pygtk-widgets.htm
- http://www.pygtk.org/articles/cairo-pygtk-widgets/cairo-pygtk-widgets2.htm

Remember, if you want to see what is available in your cairo install, use dir(cairo) from within python to see what is available.

```
import cairo
dir(cairo)
```

# Chapter 4

# Printing

Please send any fixes or suggestions to peter@majorsilence.com or leave a comment at http://www.majorsilence.com/pygtk_book.

A requirement for printing with PyGTK is cairo so it will be helpful to read the chapter on cairo first. However it is only necessary if you wish to know what is going on. If all you want is quick and easy printing than this chapter by itself should suffice.

Cairo is not only used for drawing pretty pictures. It can be used with PyGTK to print documents or whatever it is you wish to print.

## 4.1 Print Example

This chapter will provide a simple python class that takes as arguments:

- action - The action to be performed (see section 4.2 on page 73)
- data - print the provided string
- filename - open a text file to be printed

To use the PrintExample class all you have to do is create an instance specifying some data to print and the type of print action that is to be taken. For example lets say that the text "This text is Printed" is to be printed with a print dialog being opened to the user, then the following code would be used.

```
printer = PrintExample(gtk.PRINT_OPERATION_ACTION_PRINT_DIALOG,
        "This text is Printed")
```

Inside the __init__ method of the PrintExample class the paper size(see section 4.3 on page 73) is set, page setup information is created, and a print operation is initiated.

```
class PrintExample:
  def __init__(self, action=None, data=None, filename=None):
    self.text = data
```

```
        self.layout = None
        self.font_size=12
        self.lines_per_page=0
        if action==None:
          action = gtk.PRINT_OPERATION_ACTION_PREVIEW

        paper_size = gtk.PaperSize(gtk.PAPER_NAME_A4)
        setup = gtk.PageSetup()
        setup.set_paper_size(paper_size)

        print_ = gtk.PrintOperation()
        print_.set_default_page_setup(setup)
        print_.set_unit(gtk.UNIT_MM)

        print_.connect("begin_print", self.begin_print)
        print_.connect("draw_page", self.draw_page)
        if action == gtk.PRINT_OPERATION_ACTION_EXPORT:
          print_.set_export_filename(filename)

      response = print_.run(action)
```

So first off in the __init__ method a few instance variables are created.

**self.text:** Is used to hold the data that is to be printed

**self.layout:** Is used to hold an pango layout instance

**self.font_size:** Is used to hold the font size that will be use with the layout with a pango.FontDescription instance

**self.lines_per_page:** Is used to store how many lines are available per page

Next it checks to see what action as been set. If there is no action it will set as default to show a print preview.

```
action = gtk.PRINT_OPERATION_ACTION_PREVIEW
```

Next an instance of the gtk.PaperSize class is created with a paper type of gtk.PAPER_NAME_A4 and is assigned to the variable *paper_size*. After this an instance of gtk.PageSetup is created and has a page size set by the just created instance of gtk.PaperSize *paper_size*.

The print operation instance is assigned to the variable print_ using the gtk.PrintOperation class. It uses the print setup created above and sets the unit size to millimeters.

```
print_.set_default_page_setup(setup)
print_.set_unit(gtk.UNIT_MM)
```

It then connects the signals needed to print to their methods in the PrintExample class. The needed signals are *begin_print* and *draw_page*. The begin_print signal calls a method that sets up the needed information for the print operation. The *draw_page* signal calls a method that uses the the information from the *begin_print* method to print each individual page.

```
        print_.connect("begin_print", self.begin_print)
        print_.connect("draw_page", self.draw_page)
```

Lastly, if the print action is to export it also sets the filename that it is to be exported.

As stated above the begin_print method is called with the begin_print signal and will setup the information that is needed to print using the draw_page method.

```
    def begin_print(self, operation, context):
      width = context.get_width()
      height = context.get_height()
      self.layout = context.create_pango_layout()
      self.layout.set_font_description(
          pango.FontDescription("Sans " + str(self.font_size)) )
      self.layout.set_width(int(width*pango.SCALE))
      self.layout.set_text(self.text)

      num_lines = self.layout.get_line_count()
      self.lines_per_page = math.floor(
          context.get_height() / (self.font_size/2) )
      pages = ( int(math.ceil( float(num_lines) /
          float(self.lines_per_page) ) ) )
      operation.set_n_pages(pages)
```

The begin_print method has the arguments *operation* and *context*. The operation argument will be used to set the number of pages. The context is used to get the information needed and create a pango layout. Pango is the part of gtk that is used for fonts and is needed for setting the font type, setting the width of the page and setting the text.

The the first two lines retrieve the width and the height of the of the context argument (which is a cairo context). It then creates a pango instance using the context.create_pango_layout() method and assigns this to the class instance variable self.layout from this point out obviously become a pango.Layout instance.

The next part now uses self.layout to set the font type to Sans 12. The self.font_size is set as a class instance variable in the __init__ method so that it can be used from both the begin_print and draw_page methods. It sets the self.layout with to the cairo *context* width multiplied by the pango.SCALE constant (1024). After this the text of the pango layout is then set to the text that is held in the variable self.text; which was set in the __init__ method.

The number of lines in the whole document is retrieved with by calling self.layout.get_line_count(). The number of lines per page is calculated using the context height and dividing by the font size. The font size is divided by two so the lines are not spaced to far apart[1].

The number pages is calculated by dividing the number of lines in the whole document by the number of lines per page. It then sets the number pages by calling the operation.set_n_pages method.

The draw_page method is called directly after the begin_print method. It uses the information that was stored in class instance variables and in the operation argument to print each

[1] There is a different way to do this but I found this the easiest way to start off with.

page. It also has the argument page_number. This holds the current page number that is being printed. Remember that the draw_page method is not called once, it is called once for each page that is to be printed.

```
def draw_page (self, operation, context, page_number):
  cr = context.get_cairo_context()
  cr.set_source_rgb(0, 0, 0)
  start_line = page_number * self.lines_per_page
  if page_number + 1 != operation.props.n_pages:
    end_line = start_line + self.lines_per_page
  else:
    end_line = self.layout.get_line_count()

  cr.move_to(0, 0)
  iter = self.layout.get_iter()
  i=0
  while 1:
    if i > start_line:
      line = iter.get_line()
      cr.rel_move_to(0, self.font_size/2)
      cr.show_layout_line(line)
    i += 1
    if not (i < end_line and iter.next_line()):
      break
```

First off the draw_page method creates a cairo context by calling context.get_cairo_context(). The context is assigned to cr. It then sets the color of the text to black using cr.set_source_rgb(0, 0, 0). After this the starting line for the current page to print is calculated by multiplying the current page by the number of lines per page.

It then calculates the last line that is on the page. If it is not the last page of the document the last line is the start line plus the lines per page. If it is the last page to be printed the end line is the line count of the whole document.

```
if page_number + 1 != operation.props.n_pages:
  end_line = start_line + self.lines_per_page
else:
  end_line = self.layout.get_line_count()
```

With this information the method is now able to draw the text using cairo. The context is set to the upper most left part of the page using *cr.move_to(0, 0)*.

It creates an iter of the layout that is used to iterate through each line of the document that is left. A while loop is used to move through the lines. Each time through the while loop the variable i is incremented. Once I is greater than the start line, that was calculated for this page, the line is retrieved using *iter.get_line()*. The context is moved relative to its current position by the font size divided by two. Then the text is drawn to the context using the cr.show_layout_line method.

Once the variable is as incremented to a greater value then the end line, or there are no more lines in the iter to iterate through, break is called ending the while loop and exiting the draw_page method.

## 4.2 Print Actions

There are several print actions that can be used with printing.

**gtk.PRINT_OPERATION_ACTION_PREVIEW** Show the print preview

**gtk.PRINT_OPERATION_ACTION_EXPORT** Export to a file. This requires the "export-filename" property to be set

**gtk.PRINT_OPERATION_ACTION_PRINT_DIALOG** Show the print dialog

**gtk.PRINT_OPERATION_ACTION_PRINT** Start printing immediately without showing the print dialog. Based on the current print settings.

## 4.3 Paper Sizes

There are several different predefined paper sizes that can be used with PyGTK printing. These are listed below. There is also the possibility to use a custom paper size, but this is not discussed here.

**gtk.PAPER_NAME_A3** Name for the A3 paper size.

**gtk.PAPER_NAME_A4** Name for the A4 paper size.

**gtk.PAPER_NAME_A5** Name for the A5 paper size.

**gtk.PAPER_NAME_B5** Name for the B5 paper size.

**gtk.PAPER_NAME_LETTER** Name for the Letter paper size.

**gtk.PAPER_NAME_EXECUTIVE** Name for the Executive paper size.

**gtk.PAPER_NAME_LEGAL** for the Legal paper size.

## 4.4 Summary

In summary, printing using cairo sucks but at least it is not to bad.

# Chapter 5

# Gnome Desktop Integration

Please send any fixes or suggestions to peter@majorsilence.com or leave a comment at http://www.majorsilence.com/pygtk_book.

## 5.1 GConfig

Save your applications configuration file using GConfig. This example is based off the gconfig-basic-app.py file that comes with the pygtk source code, I have changed it into something I find easier to understand.

```
import gconf, gobject, gtk

class GConfigExample:
  def __init__(self):
    client = gconf.client_get_default()
    client.add_dir ("/apps/pygtk-book-gconf-example-app",
       gconf.CLIENT_PRELOAD_NONE)
```

Before even creating the gtk window, get the default gconf client, then tell the gconf client that we are interested in the given directory. This means the gconf client will receive notification of changes to this directory, and will also cache keys under this directory. To avoid getting a copy of the whole gconf database do not add "/" as that would specify the entire database. Also gconf.CLIENT_PRELOAD_NONE is used to avoid loading all config keys on startup. If the application reads all the config keys on startup, then preloading the cache may make sense, otherwise preload none is the way to go.

After setting up the initial gconf code the gtk window is created.

```
    self.window = gtk.Window()
    self.window.set_title(“GConfig Example”)
    vbox = gtk.VBox(False, 5)
    self.window.add(vbox)
```

Next, the program will have eight labels that will show the database directory path as well as the value that is being stored. The method create_configurable_widget is used to create, display, and hook up the labels to be updated on changes to the gconf database.

```
config = self.create_configurable_widget(client,
    "/apps/pygtk-book-gconf-example-app/foo")
vbox.pack_start(config, True, True)

config = self.create_configurable_widget(client,
    "/apps/pygtk-book-gconf-example-app/bar")
vbox.pack_start(config, True, True)

config = self.create_configurable_widget(client,
    "/apps/pygtk-book-gconf-example-app/baz")
vbox.pack_start (config, True, True)

config = self.create_configurable_widget(client,
    "/apps/pygtk-book-gconf-example-app/blah")
vbox.pack_start(config, True, True)
self.window.connect(“delete_event”, lambda wid, we: gtk.main_quit())
```

Here we use the set_data method on the applications main window, setting the key to “client” and the value to the gconf object that was created abouve; *client*. As well a preferences button is created and added to the window. The preferences button will open a preference dialog that will edit the gconfig entries directly and does not interact at all with the GConfigExample class that shows reading from gconf.

```
self.window.set_data (“client”, client)
prefs_button = gtk.Button ("Preferences")
vbox.pack_end (prefs_button, False, False)
prefs_button.connect (“clicked”,   self.prefs_button_clicked_callback)

self.window.show_all()
```

Once the widget monitoring notification that was created in the create_configurable_widget method is destroyed, the notification callback is removed.

```
def configurable_widget_destroy_callback(self, widget):
  client = widget.get_data(“client”)
  notify_id = widget.get_data(“notify_id”)
  if notify_id:
    client.notify_remove (notify_id)
```

Here there is a notification callback for the value label widgets that monitor the current value of a gconf key, when a gconf value is changed so is the label within the program. Note that the *value* can be None (unset) or it can have the wrong type. The program needs to check to make sure it can survive *gconftool –break-key.*

```
def configurable_widget_config_notify(self, client, cnxn_id, entry, label):
    if not entry.value:
      label.set_text(“”)
    elif entry.value.type == gconf.VALUE_STRING:
      label.set_text( entry.value.to_string() )
    else:
      label.set_text(“!type error!”)
```

This is the create_configurable_widget method that creates the labels that are displayed. Each gconf database directory will have a label to show the location as well as one label to show the value.

```
  def create_configurable_widget(self, client, config_key):
    hbox = gtk.HBox(True)

    key_label = gtk.Label(config_key + ": ")
    label = gtk.Label (“”)

    hbox.pack_start(key_label)
    hbox.pack_start(label)

    s = client.get_string(config_key)

    if s:
      label.set_text(s)

    notify_id = client.notify_add(config_key, self.configurable_widget_config_notify, label)
```

It should be noted here that notify_id will be 0 if there is an error, so that is handled in the destroy callback.

```
    label.set_data(“notify_id”, notify_id)
    label.set_data(“client”, client)

    label.connect(“destroy”, self.configurable_widget_destroy_callback)

    return hbox

  def prefs_button_clicked_callback(self, widget):
    client = self.window.get_data(“client”)
    prefs_dialog = EditConfigValues(client)
```

Next is the code for the preference dialog. the code will be in the EditConfigValues class. It is important to know that the preference dialog will never directly edit any values in the main window, it will only edit values in the gconf database. This is to test that the program works correctly as sometimes the values will be edited using gconf-editor instead of the applications preference window.

```
class EditConfigValues:
  def __init__(self, client):
    self.dialog = gtk.Dialog ("GConfig Example Preferences",

    self.dialog.connect('response', lambda wid,ev: wid.destroy ())
    self.dialog.set_default_response (gtk.RESPONSE_ACCEPT)

    vbox = gtk.VBox(False, 5)
```

Create four labels and four text entries that are used to display the gconf location as well as the current value in an entry area, this is accomplished using the create_config_entry method.

```
    self.dialog.vbox.pack_start(vbox)
    entry = self.create_config_entry(client,
        "/apps/pygtk-book-gconf-example-app/foo", True)
    vbox.pack_start (entry, False, False)

    entry = self.create_config_entry(client,
        "/apps/pygtk-book-gconf-example-app/bar")
    vbox.pack_start (entry, False, False)

    entry = self.create_config_entry (client,
        "/apps/pygtk-book-gconf-example-app/baz")
    vbox.pack_start (entry, False, False)

    entry = self.create_config_entry (client,
        "/apps/pygtk-book-gconf-example-app/blah")
    vbox.pack_start (entry, False, False)

    self.dialog.show_all()
```

The config_entry_commit method does as its names says and commits changes to the gconf database. If the *text* string is zero-length it is unset, otherwise it is set.

```
  def config_entry_commit(self, entry, *args):
    client = entry.get_data(“client”)
    text = entry.get_chars(0, -1)
    key = entry.get_data ('key')

    if text:
          client.set_string(key, text)
        else:
          client.unset(key)
```

The create_config_entry method takes as arguments the gconf client, the config key that is to be created, as well as whether the text entry has focus. This method creates a label that shows the config key and a text entry that shows the value. Editing the text entry changes the value of the gconf value.

```
      def create_config_entry(self, client, config_key, focus=False):
        hbox = gtk.HBox(False, 5)
        label = gtk.Label(config_key)
        entry = gtk.Entry()

        hbox.pack_start(label, False, False, 0)
        hbox.pack_end(entry, False, False, 0)
```

Calling client.get_string(config_key) will print an error via the default error handler if the key is not set to a string.

```
        s = client.get_string(config_key)
        if s:
          entry.set_text(s)

        entry.set_data(“client”, client)
        entry.set_data(“key”, config_key)
```

The changes will be commited if the user moves focus away from the text entry they are in, or if they hit enter; Changes are not commited on the *changed* signal as that would mean every new character entered would be sent, instead it waits for the user to finish first. Finally if the gconf client key is not writable the text entry is set to not writtable.

```
        entry.connect(“focus_out_event”, self.config_entry_commit)
        entry.connect (“activate”, self.config_entry_commit)
        entry.set_sensitive( client.key_is_writable(config_key) )
        if focus:
          entry.grab_focus()
        return hbox

    if __name__ == "__main__":
      GConfigExample()
      gtk.main()
```

## 5.2 PyGobject

I am not going to cover very much in this section because that would be a lot, maybe in a later verion. For now this section will cover one useful function. For more information check out its documentation at http://pygtk.org/docs/pygobject/index.html.

**gobject.timeout_add(interval,callback)** is a function that will call the function specified in the callback as often as is specified by the interval until the callback function returns False.

Interval The number of seconds between calls. Eg. 1 for one second, 100 for 100 seconds. That is pretty simple

Callback The function that will be called at each interval.

I find this is a useful function to use when I want to periodically check to see if a long running process has finished. Another good example is lets say you have a music player with a progress bar, once a second while a song is playing you would want to update the progress bar. To do this you could setup a *gobject.timeout_add* to call an update function that checks the position of the currently playing song and update the progress bar with that information.

## 5.3 Gnome Menus (.desktop files)

If an application is to be added to the main menu it will need an appname.desktop file with details about the application. The .desktop file will hold various information about the application including the name, how to execute it, tool tip comment, icon, category and more.

### 5.3.1 Keys

The way that a .desktop file holds information is with keys. There are several keys and a few of them are required. The required keys are:

- Type - Application, Link, Directory
- Name - The name of the application and what will show up in the menu
- Exec - The program to execute with arguments
- URL - Only required if the entry is a Link Type

There are several other keys besides the required ones. To see what is available visit the .desktop files specification web page[1].

.desktop Example

[1] For more information on keys that can be used with .desktop files please visit: http://standards.freedesktop.org/desktop-entry-spec/latest/ar01s05.html

```
[Desktop Entry]
Version=1.0
Encoding=UTF-8
Name=Hello World
GenericName=Display Hello World
Comment=This is my first PyGTK application
X-MultipleArgs=false
Type=Application
TryExec=helloworld
Exec=helloworld
Categories=Utility
Icon=helloworld
```

Save this example as helloworld.desktop. When it is viewed with a file manger it will show up as "Hello World" because that is what the Name key is set to.

This Example sets up a .desktop with a version of 1.0. The encoding type is UTF-8. The GenericName is a generic name to describe the application. It is assigned to the Application type. It will try to execute helloworld. The category is Utility. The utility category means that it will be placed in the Accessories category in the menu. The comment is the tooltip for the that will be displayed on hovering over it. And last the icon is set to helloworld.

When using Icons it must be set to the absolute path or be installed in a location that it is able to be found. This helloworld icon is a image with the name helloworld.png and can be found on the books website. Supported icon image types are png, xpm and svg.

### 5.3.2 Category Information

Included in the keys that can be used with a .desktop file is the category key. The category is the category that the Application, Link, or Directory will be included under. If for example we have an Application and it is in the Office category; then when the main menu is opened and the office subcategory is opened the application will show up there.

Here is a list of the default categories. More categories can be found on the menu specification web page[2].

- AudioVideo - A multimedia (audio/video) application
- Audio - An audio application Desktop entry must include AudioVideo as well
- Video - A video application Desktop entry must include AudioVideo as well
- Development - An application for development
- Education - Educational software
- Game - A game
- Graphics - Graphical application

[2] If you would like more information on categories please visit: http://standards.freedesktop.org/menu-spec/menu-spec-1.0.html

- Network - Network application such as a web browser
- Office - An office type application
- Settings - Settings applications Entries may appear in a separate menu or as part of a "Control Center"
- System - System application, "System Tools" such as say a log viewer or network monitor
- Utility - Small utility application, "Accessories"

### 5.3.3 Installing and Using .desktop files

Creating a .desktop file without installing is pointless. It must be installed to be used. This section is going to use a small sample PyGTK application with a .desktop file to show how they work together. Then a small shell script will be created to install or uninstall the application, application data, and related .desktop file.

First lets create a small python program that is the main file to create the GUI and a second python file that will only have one function that returns a small message. These two files are used to show how it can be installed and set the path in the main python file to the correct install location of the supporting python modules that are included in the application[3].

```
#!/usr/bin/env python
import sys
sys.path.append("/usr/local/lib/helloworld")
import gtk
import helloworld_message

if __name__ == '__main__':
  win = gtk.Window()
  win.connect("delete_event", lambda w,e: gtk.main_quit())
  label = gtk.Label(helloworld_message.message())
  win.add(label)
  win.show_all()
  gtk.main()
```

At the very top of this example sys is import and the location /usr/local/lib/helloworld is append to the system path. The reason this is done is because this is where all the applications modules will be installed. If it does not append this directory then importing the helloworld_message module will fail.

The helloworld_message.py file only contains one function and is only two lines long.

```
def message():
  return ".desktop example program"
```

[3] A better way would probably be to install all the files to the library directory including the main python file. Then install a shell script to the binary directory that looks for and launches the directory. This way it does not need to append the to the system path the location of the applications python modules.

Now that there is a working application and a desktop file that was created above it is time to install everything. For the purposes of installing the helloworld.desktop, helloworld.py, and helloworld_message.py files a bash shell script will be used.

The shell script will take one argument that may be either –install or –uninstall. Anything other then that will display how to use this shell script. This script has been kept very simple so that it will be easy to understand.

To start off lets cover the beginning of the script.

```
#!/bin/bash
# Get script directory path.
scriptdir="‘dirname ${0}‘"
DESTDIR="${DESTDIR:-}"
```

These first few lines set the shell script to be run by bash and set the variables "scriptdir" and "DESTDIR".

Next is the installation function. This function will install the main python file as a binary and the supporting python modules and data files.

```
install_program() # arg1=bindir, arg2=datadir, arg3=pkglibdir,
        # arg4=pkgdatadir, arg5=pkgdocdir.
{
  echo ${DESTDIR}
  # Install binary data - /usr/local/bin/helloworld
  install -m 755 -d "${DESTDIR}${1}"
  install -m 755 "${scriptdir}/helloworld.py" "${DESTDIR}${1}/helloworld"

  # Install package library - /usr/local/lib/helloworld
  install -m 755 -d "${DESTDIR}${3}"
  install "${scriptdir}"/helloworld_*.py "${DESTDIR}${3}/"

  # Install package data /usr/local/share/helloworld
  #install -m 755 -d "${DESTDIR}${4}"
  #install -m 644 "${scriptdir}/helloworld.png" "${DESTDIR}${4}/"
  # Install data directory - /usr/local/share/pixmaps
  install -m 755 -d "${DESTDIR}${2}/pixmaps"
  install -m 644 "${scriptdir}/helloworld.png" "${DESTDIR}${2}/pixmaps/"

  # /usr/local/share/applications
  install -m 755 -d "${DESTDIR}${2}/applications"
  install -m 644 "${scriptdir}/helloworld.desktop" \
      "${DESTDIR}${2}/applications/"

  echo "Finished Install"
}
```

This function takes five arguments that specifiy where the binary, data, library, package data, and documentation are to be installed. It installs the helloworld.py file to /usr/local/bin/helloworld so it may be run by executing helloworld. It then install all python files that start with “helloworld_” to the /usr/local/lib/helloworld directory. If there were any data files they would be installed to /usr/local/share/helloworld directory, but since there were none those lines are commented out(they are only using the helloworld.png file as an example).

The helloworld.png file is installed to the /usr/local/share/pixmaps directory, making it usable as an icon from the helloworld.desktop file. And at the very last, helloworld.desktop is installed to the /usr/local/share/applications directory. Once this is completed the the helloworld application should show up in the menu (Applications -> Accessories -> Hello World).

The next and last function in the install script is used to uninstall the helloworld application and is much smaller then the install function.

```
uninstall_program() # arg1=bindir, arg2=datadir, arg3=pkglibdir,
        # arg4=pkgdatadir, arg5=pkgdocdir.
{
  rm -f "${DESTDIR}${1}/helloworld"
  rm -f "${DESTDIR}${1}/helloworld.py"
  rm -rf "${DESTDIR}${3}"
  rm -rf "${DESTDIR}${4}"
  rm -rf "${DESTDIR}${5}"
  rm -f "${DESTDIR}${2}/pixmaps/helloworld.png"
  rm -f "${DESTDIR}${2}/applications/helloworld.desktop"
  echo "Finished Uninstall"
}
```

The uninstall function deletes all the files that were installed and all the directories that were created by the install function. This is very simple and there is no more to say about it.

The last part is to read the arguments given to the shell script and call the right function.

```
# First arg to the script
action=$1
if test "$action" = -install
then
  echo "install selected"
  install_program "/usr/local/bin" \
        "/usr/local/share" \
        "/usr/local/lib/helloworld" \
        "/usr/local/share/helloworld" \
        "/usr/local/share/doc/helloworld"
elif test "$action" = -uninstall
then
  echo "uninstall selected"
  uninstall_program "/usr/local/bin" \
        "/usr/local/share" \
        "/usr/local/lib/helloworld" \
        "/usr/local/share/helloworld" \
        "/usr/local/share/doc/helloworld"
```

```
else
  echo ""
  echo "Usage:"
  echo " -install - Use this argument to install"
  echo " -uninstall - Use this argument to uninstall"
  echo ""
fi
```

This part of the install script reads the first argument to it and assigns it to the variable action. Then action is tested to see if it should install, uninstall, or display the accepted arguments.

That is all to creating a .desktop file for use with an application.

# Chapter 6

# Audio and Video Playback - GStreamer

Please send any fixes or suggestions to peter@majorsilence.com or leave a comment at http://www.majorsilence.com/pygtk_book.

## 6.1 Introduction

GStreamer is a multimedia framework that can be used from the simple to the more advanced. The possibilities range from playing a simple audio file or video file to creating an advanced audio/video editor.

When you are finished reading this chapter you will be able to use a high level playbin factory element to play audio and video, detect missing codecs and automatically install them, discover the file information about your audio or video files and apply all these to your very own PyGTK program.

This chapter does not cover any advanced topics but it does show you how to very quickly add the ability to play audio or video to your own program.

Enjoy the journey.

## 6.2 The Beginnings

### 6.2.1 Playbin

The playbin element is a very high level, automatic video/audio player. It will automatically detect your multimedia file type and take the correct actions for it to be played. All that needs to be done to use it is to supply the playbin with a location of a multimedia file and set the state of it to play.

**playbin Features:**

- Audio and video output ("audio-sink" and "video-sink")

- Error Handling
- EOS handling(end of stream)
- State handling
- Seeking
- Buffers network sources
- Visualization for audio supported
- Subtitle support (“suburi”)

**Playbin element:**

What is needed in every GStreamer application is an element to play your media with. In the case of this chapter all that we are going to use is the “playbin” factory. You create this using the gst.element_make_factory(factory, element_name) method like so:

```
player_name = gst.element_make_factory(“playbin”, “YourElementName”)
```

**Set location of the multimedia file:**

Setting the location of the file is done with the player_name.set_property(“uri”, “location”) like so:

```
player_name.set_property(“uri”, “file:///home/peter/myvideo.avi”)
```

**Set state of the multimedia file:**

Some states that the multimedia file may be set to include:

- gst.STATE_PLAYING – Used to start playing
- gst.STATE_PAUSED – Used to pause file
- gst.STATE_NULL – Used to stop file

To set the state of the “player_name” just created above to play the file you would set_state(state) method like so:

```
player_name.set_state(gst.STATE_PLAYING)
```

To set the file to be paused you would:

```
player_name.set_state(gst.STATE_PAUSED)
```

To altogether stop the file that is playing you would set the state like so:

```
player_name.set_state(gst.STATE_NULL)
```

### 6.2.2 Bus - watching for GStreamer signals

The GStreamer bus is what allows for receiving signals from GStreamer. It is important because it will allow your program to detect things such as errors or the end of the audio or video stream.

When the end of stream is detected it is the programs responsibility to set the state back to gst.STATE_NULL. Otherwise if you try to load in another file or play the same file again it will not play because the state is already set to gst.STATE_PLAYING.

The bus is not difficult to use and it will only add a few more lines to the program and one extra function to handle the messages.

So if we have created a player bin using the gst.element_make_factory method and have called it *player_name* then we can create a bus and watch it like so:

```
bus = player_name.get_bus()
bus.add_signal_watch()
bus.connect(“message”, on_message)
```

This creates a bus from the player_name playbin, adds a signal watcher, and connects the bus to send signals to the function on_message when messages are detected.

The function message can detect whatever type of message that GStreamer has but in the examples in this chapter it will focus on errors and detecting when the end of stream has occurred so that the program will reset the state to gst.STATE_NULL.

An example message function looks like this:

```
def on_message(self, bus, message):
    # Detect end of stream and set state to to NULL
    if message.type == gst.MESSAGE_EOS:
        self.player_name.set_state(gst.STATE_NULL)
    elif message.type == gst.MESSAGE_ERROR:
        self.player_name.set_state(gst.STATE_NULL)
        (err, debug) = message.parse_error()
        print “Error: %s” % err, debug
```

## 6.3 Playing Audio

Playing audio with PyGST is a very simple matter that only requires a few lines of code to get the audio playing.

As the was just covered we will use the gst.element_make_factory function and set it up with a PyGTK GUI. But besides that we will create a false video sink using the gst.element_make_factory function so that if the multimedia file is a video, only the audio portion is played. This is because we are using the high level “playbin” element which will automatically play everything. So if all you want played is the audio, the video must be redirected.

Create a multimedia playbin to play the audio and redirect all video to a fake video sink that is added to the multimedia pipeline:

```
# Create the player_name sink
player_name = gst.element_make_factory(“playbin”, “Multimedia Player”)
```

```
# Create the fake video sink
fake_video_sink = gst.element_make_factory(“fakesink”, “Fake sink for Videos”)
#Add the fake video sink to the player
player_name.set_property(“videosink”, fake_video_sink)
```

If a fake video sink is not created and a video file is played ,it will pop up a window with the video playing in it. This will really subtract from the professional feel of your application.

Now what is needed is to add the audio source using the player_name.set_property method and set the state to playing. This is just like what is discussed earlier and is now shown below:

```
player_name.set_property(“uri”, “file:///home/peter/mymusic.mp3”)
player_name.set_state(gst.PLAYING)
```

It really is that simple.

For a full example of how to hook up audio and video to a PyGTK application please review the example at the end of the chapter.

## 6.4 Playing Video

Playing audio is very simple and is much like playing audio except that with playing video there is no fake video sink created to hide the video.

To play video a playbin must be created using the gst.element_make_factory function. Then set the location of the video file with the newly created playbin and then set the playbin state to gst.PLAYING.

```
# Create the player_name sink
player_name = gst.element_make_factory(“playbin”, “Multimedia Player”)
# Set the location of the video file
player_name.set_property(“uri”, “file:///home/peter/myvideo.avi”)
# Start playing the video.
player_name.set_state(gst.PLAYING)
```

It really is much shorter to play a video then it is an audio file. But remember that you should also hook up a bus, as shown in the section “bus – watching for GStreamer signals”, to your video to catch messages. However there is a problem with this code.

If you play a video with this code the video will open up in its own window. If the video opening in its own window is good enough for your program so be it; however I believe that for most programs the video will be better suited in a widget inside of the application.

### 6.4.1 Play the Video in you Application

To play a video file in your own application you use a gtk.DrawingArea widget to play the video. You create a gtk.DrawingArea and sync it with the video using the bus that has been created to watch for GStreamer messages.

Create your gtk.DrawingArea like so:

```
self.videowidget = gtk.DrawingArea()
self.videowidget.set_size_request(400, 250)
```

Then add this widget to your PyGTK window.

Now you sync the video to your videowidget using the bus. If your bus name is *bus* you would enable sync messages and connect it to a function with the following code:

```
bus.enable_sync_message_emission()
bus.connect(“sync-message::element”, self.on_sync_message)
```

This code enables the sync message and then connects any signals to be forwarded to the *self.on_sync_message* function. The on_sync_message function will hook the video up to the gtk.DrawingArea widget that has been created to show the video.

Here is an example function showing how to play a video.

```
def on_sync_message(self, bus, message):
    if message.structure is None:
        return False
    if message.structure.get_name() == “prepare-xwindow-id”:
        if sys.platform == “win32”:
            win_id = self.videowidget.window.handle
        else:
            win_id = self.videowidget.window.xid
        assert win_id
        imagesink = message.src
        imagesink.set_property(“force-aspect-ratio”, True)
        imagesink.set_xwindow_id(win_id)
```

Now when the state of your playbin element is set to play using gst.PLAYING, the video will be played inside of your application instead of opening up in its own window.

For a full example of how to hook up audio and video to a PyGTK application please review the example at the end of the chapter.

### 6.4.2 Play Video Example

This example will be referred to in following sections and when adding things such as seeking and will be expanded upon in the file example at the end of the chapter.

```
#!/usr/bin/env python
import pygst
pygst.require("0.10")
import gst, pygtk, gtk
import sys

class Main(object):
    def __init__(self):
        self.multimedia_file=""
```

```
# Create the GUI
self.win = gtk.Window()
self.win.set_title("Play Video Example")
self.win.connect("delete_event",
    lambda w,e: gtk.main_quit())

vbox = gtk.VBox(False, 0)
hbox = gtk.HBox(False, 0)
self.load_file =
    gtk.FileChooserButton("Choose Audio File")
self.play_button =
    gtk.Button("Play", gtk.STOCK_MEDIA_PLAY)
self.pause_button =
    gtk.Button("Pause", gtk.STOCK_MEDIA_PAUSE)
self.stop_button =
    gtk.Button("Stop", gtk.STOCK_MEDIA_STOP)
self.videowidget = gtk.DrawingArea()
# You want to expand the video widget or
# else you cannot see it
self.videowidget.set_size_request(400, 250)

self.load_file.connect("selection-changed",
    self.on_file_selected)
self.play_button.connect("clicked", self.on_play_clicked)
self.pause_button.connect("clicked", self.on_pause_clicked)
self.stop_button.connect("clicked", self.on_stop_clicked)

hbox.pack_start(self.play_button, False, True, 0)
hbox.pack_start(self.pause_button, False, True, 0)
hbox.pack_start(self.stop_button, False, True, 0)
vbox.pack_start(self.load_file, False, True, 0)
vbox.pack_start(self.videowidget, True, True, 0)

vbox.pack_start(hbox, False, True, 0)
self.win.add(vbox)
self.win.show_all()

# Setup GStreamer
self.player = gst.element_factory_make(
    "playbin", "MultimediaPlayer")
bus = self.player.get_bus()
bus.add_signal_watch()
bus.enable_sync_message_emission()
#used to get messages that GStreamer emits
bus.connect("message", self.on_message)
```

```
        #used for connecting video to your application
        bus.connect("sync-message::element",
            self.on_sync_message)

    def on_file_selected(self, widget):
        self.multimedia_file = self.load_file.get_filename()

    def on_play_clicked(self, widget):
        self.player.set_property(’uri’,
            "file://" + self.multimedia_file)
        self.player.set_state(gst.STATE_PLAYING)

    def on_pause_clicked(self, widget):
        self.player.set_state(gst.STATE_PAUSED)

    def on_stop_clicked(self, widget):
        self.player.set_state(gst.STATE_NULL)

    def on_message(self, bus, message):
        if message.type == gst.MESSAGE_EOS:
            # End of Stream
            self.player.set_state(gst.STATE_NULL)
        elif message.type == gst.MESSAGE_ERROR:
            self.player.set_state(gst.STATE_NULL)
            (err, debug) = message.parse_error()
            print "Error: %s" % err, debug

    def on_sync_message(self, bus, message):
        if message.structure is None:
            return False
        if message.structure.get_name() == "prepare-xwindow-id":
            if sys.platform == “win32”:
                win_id = self.videowidget.window.handle
            else:
                win_id = self.videowidget.window.xid
            assert win_id
            imagesink = message.src
            imagesink.set_property("force-aspect-ratio", True)
            imagesink.set_xwindow_id(win_id)

if __name__ == "__main__":
    Main()
    gtk.main()
```

## 6.5 Multimedia Info

Now what I should mention here is that this code is more or less the unmodified example that comes with the PyGST source code. Copyright (C) 2006 Andy Wingo, LGPL Version 2.

Lets say that you are going to play a video and you want to know some information about it. Maybe you want to know what the video width and height is to set a proper size on your video widget. Or maybe you want to know the length of the video. Well this information is very easy to find out using GStreamer.

First you will have to import the GStreamer discoverer like so:

```
from gst.extend import discoverer
```

Now that you have the discoverer imported you can access information about the video file with only a few lines of code.

We create a discover function that will be the main work area that hooks everything together.

The discover functions includes an in-line function that is connected to using a gobject main loop since this is a command line example. If this code is used in a PyGTK GUI it will will run fine without the gobject main loop because it is already running in the applications GTK main loop.

If the the file is discovered to be a multimedia file it is then sent to the succeed function where it prints out information about the file.

If it fails and is not recognized as a multimedia file then it prints out an error message and exits the gobject main loop.

```
def discover(path):
    def discovered(d, is_media):
        if is_media:
            succeed(d)
        else:
            print “error: %r does not appear to be a media file” % path
            # Exit the gobject main loop
            # Remove this in a pygtk program.
            sys.exit(1)
    d = discoverer.Discoverer(path)
    # Connect discovered to the inline function discovered.
    d.connect(“discovered”, discovered)
    d.discover()
    # comment out the gobject.MainLoop.run() in a pygtk program.
    gobject.MainLoop().run()
```

The succeed method is called from the discover function when the file is detected as a multimedia file. It can be used to print out or save information about the video or audio file.

Data available for video files include:

- is_video
- video_length

- fps - videorate.num / videorate.denom
- videocaps
- videowidth
- videoheight

Data available for audio files include:

- is_audio
- audiocaps
- audiofloat
- audiorate
- audiowidth
- audiodepth
- audiolength
- audiochannels

```
def succeed(d):
    print(“media type”, d.mimetype)
    print(“has video”, d.is_video)
    if d.is_video:
        print(“video length (ms)”, d.videolength / gst.MSECOND)
        print(“framerate (fps)”, “%s/%s” % (d.videorate.num, d.videorate.denom))
    print(“has audio”, d.is_audio)
    if d.is_audio:
        print(“audio caps”, d.audiocaps)
        print(“audio format”, d.audiofloat and “floating-point” or “integer”)
        print(“audio length (ms)”, d.audiolength / gst.MSECOND)
    # Exit gobject main loop.
    sys.exit(0)
```

All that is left is to run the discover file with a path to a multimedia file specified. To read in the file location and do the proper handling of it you could use the following code:

```
if __name__ == “__main__”:
    if len(sys.argv) != 2:
        print >> sys.stderr, “usage: script_name.py PATH-TO-MEDIA-FILE”
        sys.exit(1)
    path = sys.argv.pop()
    if not os.path.isfile(path):
        print >> sys.stderr, “error: file %r does not exist” % path
```

```
            print >> sys.stderr, “usage: gst-discover PATH-TO-MEDIA-FILE”
            sys.exit(1)
        discover(path)
```

For a full example of how to retrieve information from an audio or video file from a PyGTK application, please review MediaInfo class in the example at the end of the chapter. Also you can check out the PyGST examples on this books website.

## 6.6 Codec Buddy - Auto install multimedia Codecs

Tested with Ubuntu 8.10

Taking advantage of the gst.pbutils allows a program to automatically install available codecs or provide the user of the program a choice of actions to take.

```
#!/usr/bin/python
import pygst, gst, pygtk, gtk
pygst.require("0.10")
class InstallMissingCodecExample(object):
  def __init__(self):
    # Gtk Gui
    self.win = gtk.Window()
    self.win.set_title("Install Missing Codec Example")
    self.win.connect("delete_event", lambda w,e: gtk.main_quit())
    self.load_file = gtk.FileChooserButton("Choose Audio File")
    self.load_file.connect("selection-changed", self.on_file_selected)
    self.win.add(self.load_file)
    self.win.show_all()

    # Setup GStreamer
    self.player = gst.element_factory_make("playbin",
      "MultimediaPlayer")
    bus = self.player.get_bus()
    bus.add_signal_watch()
    bus.connect("message", self.on_message)

  def on_file_selected(self, widget):
    print "Selected: ", self.load_file.get_filename()
    multimedia_file = self.load_file.get_filename()
    self.player.set_property('uri', "file://" + multimedia_file)
    self.play()

  def play(self):
    self.player.set_state(gst.STATE_PLAYING)
    # Codec Buddy Methods
  def on_message(self, bus, message):
    import gst
    if message.type == gst.MESSAGE_ERROR:
      self.player.set_state(gst.STATE_NULL)
      (err, debug) = message.parse_error()
      print "Error: %s" % err, debug
    elif message.type == gst.MESSAGE_EOS:
      # End of Stream
      self.player.set_state(gst.STATE_NULL)
    elif message.type == gst.MESSAGE_ELEMENT:
      """  CodicBuddy Stuff  """
      st = message.structure
      if st and st.get_name().startswith('missing-'):
        self.player.set_state(gst.STATE_NULL)
        if gst.pygst_version >= (0, 10, 10):
          import gst.pbutils
          detail = gst.pbutils.missing_plugin_message_get_installer_detail(message)
          gst.pbutils.install_plugins_async([detail],
            context, self.install_plugin)

  def install_plugin(self, result):
    if result == gst.pbutils.INSTALL_PLUGINS_SUCCESS:
      gst.update_registry()
      self.play()
      return
    if result == gst.pbutils.INSTALL_PLUGINS_USER_ABORT:
      dialog = gtk.MessageDialog(parent=None,
          flags=gtk.DIALOG_MODAL, type=gtk.MESSAGE_INFO,
          buttons=gtk.BUTTONS_OK, message_format=
          "Plugin installation aborted.")
      dialog.run()
      dialog.hide()
      return
    error.show("Error", "failed to install plugins: %s" %
      str(result))

if __name__ == "__main__":
  InstallMissingCodecExample()
  gtk.main()
```

## 6.7 Seeking - Basic Position Seeking

Seeking allows multimedia software to display the position in the audio or video stream and also may allow the user to skip to a different section of the media file they are watching or listening to.

### 6.7.1 Displaying the Current Position

When playing a media file it may be a good idea to display the current position relative to the duration of the file as a courtesy to the user.

Displaying the duration of the file and current time position will require adding two methods to the play video ( 6.4.2 on page 91) example found earlier in the chapter.

In the __init__ method of the Main class the variables time_format, duration and is_playing are added.

```
self.time_format = gst.Format(gst.FORMAT_TIME)
self.duration = None
self.is_playing = False
```

The time_format will be used when seeking the the duration of the media file and seeking the current position of the file. The duration will be a string to display the length of the media file. the is_playing variable will be used to let the methods that are going to soon be added know if the player is playing or not.

The is_playing variable must be set to False anytime the file is not playing. This includes the end of stream message in the on_message method and when the pause and stop buttons are clicked.

Further down in the __init__ method a label called time_label is added and that is it for the GUI changes to display the time.

```
self.time_label = gtk.Label(“00:00 / 00:00”)
```

Going through the different methods, besides setting is_playing to False in the on_stop_clicked and on_pause_clicked methods, in the on_play_clicked it must be set to True. But after setting it to playing by clicking the play button the application must be able to update the GUI with the new current position every second.

To update the GUI every second the on_play_clicked button adds the following:

```
timer = gobject.timeout_add(1000, self.update_time_label)
```

This will create a timer that is executed every one second calling the method update_time_label. It will execute every one second as long a update_time_label returns true.

Skipping down to below the on_sync_message method there is the new function update_time_label.

```
def update_time_label(self):
  """
  Update the time_label to display the current location
  in the media file as well as update the seek bar
  """
  if self.is_playing == False:
    print "return false"
  return False
print "update_time_label"
if self.duration == None:
  try:
    self.length = self.player.query_duration(self.time_format, None)[0]
    self.duration = self.convert_time(self.length)
  except:
    self.duration = None

if self.duration != None:
  self.current_position = self.player.query_position(self.time_format, None)[0]
  current_position_formated = self.convert_time(self.current_position)
  self.time_label.set_text(current_position_formated + "/" + self.duration)

  # Update the seek bar
  # gtk.Adjustment(value=0, lower=0, upper=0, step_incr=0, page_incr=0, page_size=0)
  percent = (float(self.current_position)/float(self.length))*100.0
  adjustment = gtk.Adjustment(percent, 0.00, 100.0, 0.1, 1.0, 1.0)       self.seeker.set_adju

return True
```

If the *is_playing* variable is set to False the *update_time_label* method will return False. This method starts to be called when the play button is called and is called every one second until it returns False.

If the duration of the file has not yet been set it will be set here. The duration is found in nanoseconds and is converted to a string by passing it into the convert_time method. The duration variable will be reset to None each time a new media file is added to be played. If the duration is not None then it never set again unless a new file is loaded.

After duration of the file is found the *current_position* variable is calculated every time the *update_time_label* is called. The current position is found the same was as the duration except that the *query_position* function is used. Then the time in nanoseconds is converted to a person understandable string.

Once the duration and the current position is found it is displayed to the user by setting the text of the time_label like so:

```
self.time_label.set_text(current_position_formated + “/” + self.duration)
```

As was just discussed above, the convert_time function is used to convert the time of the media file from nanoseconds to human readable string. This code is adapted from a tutorial[1] found on the PyGST documentation site. It takes the time in nanoseconds and converts it to human readable string in the format of HH::MM::SS and then returns it.

```
def convert_time(self, time=None):
```

[1] This is licensed under the LGPL Version 3 and can be found at: http://pygstdocs.berlios.de/pygst-tutorial/seeking.html

```
# convert_ns function from:
# http://pygstdocs.berlios.de/pygst-tutorial/seeking.html
# LGPL Version 3 - Copyright: Jens Persson
if time==None:
  return None

hours = 0
minutes = 0
seconds = 0
time_string = ""

time = time / 1000000000 # gst.NSECOND

if time >= 3600:
  hours = time / 3600
  time = time - (hours * 3600)
if time >= 60:
  minutes = time / 60
  time = time - (minutes * 60)
#remaining time is seconds
seconds = time

time_string = time_string + str(hours).zfill(2) + ":" +
    str(minutes).zfill(2) + ":" + str(seconds).zfill(2)

#return time in Hours:Minutes:Seconds format
return time_string
```

If the time passed in is None it immediately returns None. The method divides the passed in time by 1 000 000 000 and then proceeds to calculate the hours, minutes, and seconds; creating a nice string HH:MM:SS to view by a human.

When the time_string has been completed it is returned to be used by the user interface.

### 6.7.2 Seeking a New Position

Like displaying the position and duration, seeking a new position in a media file will use the methods convert_time as well as update_time_label, but it will also use a horizontal scaler to that it will let the user slide to a new position.

To allow a user to update the position of the media file a horizontal scaler needs to be added. To use a scaler you must create an adjustment first.

```
self.adjustment = gtk.Adjustment(0.0, 0.00, 100.0, 0.1, 1.0, 1.0)
```

This creates a new adjustment with a starting value of 0, lower limit of 0.00, upper limit of 100.0, and step increment of 0.1, page increment of 1.0, and page size of 1.0. The adjustment is used with the the horizontal scaler that is to be created to control the place in the media file.

```
self.seeker = gtk.HScale(self.adjustment)
self.seeker.set_draw_value(False)
self.seeker.set_update_policy(gtk.UPDATE_DISCONTINUOUS)
```

The first line creates the seeker and the set_draw_value(False) line keeps the format-value signal from being admitted and the value of the current position is not displayed.

On the third line the seeker is set to update in a discontinuous way. What this means is that it will only be updated when a button-release-event signal is emitted.

The new method being added for seeking is seeker_button_release_event and the signals is connected like this:

```
self.seeker.connect(“button-release-event”, self.seeker_button_release_event)
```

When the scaler button is released this method is called and the media file is set to a new position.

```
def seeker_button_release_event(self, widget, event):
    print "seeker_button_release_event"
    value = widget.get_value()
    if self.is_playing == True:
        duration = self.player.query_duration(self.time_format, None)[0]
        time = value * (duration / 100)
        print self.convert_time(time)
        self.player.seek_simple(self.time_format, gst.SEEK_FLAG_FLUSH, time)
```

When the self.seeker is released, its current position is retrieved and the position to reposition the media file is calculated.

```
time = value * (duration / 100)
```

After the new position is determined, the media file is set to it using the seek_simple function. The seek_simple function takes a GStreamer time format, a seek flag, and the new time[2], returning True if it succeeds.

## 6.8 Volume Control

Adding the option to control the volume of audio or video from individual programs is accomplished using the gtk.VolumeButton.

A volume button is created like any other widget in PyGTK.

```
volume_button = gtk.VolumeButton()
```

Then hook the volume button up to the value-changed signal with a method to control the volume.

[2] http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer/html/GstElement.html

```
volume_button.connect("value-changed", self.on_volume_changed)
```

The last piece to complete is to create the method that is being used to increase and decrease the volume.

```
def on_file_selected(self, widget, value=0.5)
    self.player.set_property("volume", float(value))
    return True
```

What this method does is to control the volume and increase by the percent raised on the volume slider. The default value is 0.5 if it is not specified.

### 6.8.1 Volume Control Example

```
class Main(object):
    def __init__(self):
        self.win = gtk.Window()
        self.win.set_title("Volume Control Example")
        self.win.set_default_size(200, -1)
        self.win.connect("delete_event", lambda w,e: gtk.main_quit())

        hbox = gtk.HBox(False, 0)
        self.load_file = gtk.FileChooserButton("Choose Audio File")\
        self.load_file.connect("selection-changed", self.on_file_selected)
        volume_button = gtk.VolumeButton()
        volume_button.connect("value-changed", self.on_volume_changed)

        hbox.pack_start(self.load_file, True, True, 0)
        hbox.pack_start(volume_button, False, True, 0)
        self.win.add(hbox)
        self.win.show_all()

        self.player = gst.element_factory_make("playbin", "MultimediaPlayer")
        bus = self.player.get_bus()
        bus.add_signal_watch()
        bus.enable_sync_message_emission()
        bus.connect("message", self.on_message)

    def on_file_selected(self, widget):
        self.player.set_property("uri", "file://" + self.load_file.get_filename())
        self.player.set_state(gst.STATE_PLAYING)

    def on_volume_changed(self, widget, value=10):
        self.player.set_property("volume", float(value))
        return True
    def on_message(self, bus, message):
        if message.type == gst.MESSAGE_EOS:
            self.player.set_state(gst.STATE_NULL)
        elif message.type == gst.MESSAGE_ERROR:
            self.player.set_state(gst.STATE_NULL)
            (err, debug) = message.parse_error()
            print "Error: %s" % err, debug

if __name__ == "__main__":
    Main()
    gtk.main()
```

## 6.9 Example

The purpose of this section is to provide a full media example that includes a user interface written with PyGTK. The application will be able to play, pause, or stop audio and video. Also the program will be able to discover information about the media file such as its length, width and height, and audio format using the MediaInfo class.

Using the information found with the MediaInfo class the user interface will be able to resize its video display to match the width and height of the video.

The GstPlayer class will wrap the GStreamer functions to make it easy to separate the multimedia and user interface functionality. The user interface will use a separate class called VideoWidget to display the video in also to make it easier to reuse.

### 6.9.1 MediaInfo Class

The MediaInfo class is very similar to the media info section covered earlier in the chapter. What the MediaInfo class does here is to create an easy to use wrapper around the GStreamer discoverer functions, providing accessor methods to the data.

Though there are many different pieces of information that can be discovered about a media file, for this example it will be kept short. For more information on the different variables that can be used to access the information please refer to the *multimedia info* section earlier in this chapter.

The only information that is of interest at the moment is if the media file is audio or video. If it is a video file, the width and the video height is of interest. If seeking is involved then the length of the file is also of interest, but this is not covered in this example.

```
class MediaInfo:
    def __init__(self, path):
        def discovered(d, is_media):
            if is_media:
                self.succeed(d)
            else:
                self.fail(path)
        self.__finished = False
        self.__is_media=False
        self.__video_width = 0
        self.__video_height = 0
        self.__is_video = False
        self.__is_audio = False
        self.__video_length = 0.0
        self.__frame_rate = “”
        self.__is_fullscreen = False
        print “path: “, path
        d = discoverer.Discoverer(path)
        #print help(d.discover)
        d.connect(“discovered”, discovered)
        d.discover()
    def fail(self, path):
        print “error: %r does not appear to be a media file” % path
        self.__is_media = False
    def succeed(self, d):
        print “File discover success”
        self.__is_media = True
        self.__mimetype = d.mimetype
        self.__is_video = d.is_video
        if self.__is_video:
            self.__video_width = d.videowidth
            self.__video_height = d.videoheight
            # Retrieve the video length in minute
            self.__video_length = ((d.videolength / gst.MSECOND) / 1000) / 60
            self.__frame_rate = “%s/%s” % (d.videorate.num, d.videorate.denom)
        self.__finished = True
    def poll(self):
        return self.__finished
    def is_media(self):
        return self.__is_media
    def is_video(self):
        return self.__is_video
    def is_audio(self):
        return self.__is_audio
    def get_width(self):
        return self.__video_width
    def get_height(self):
        return self.__video_height
```

The MediaInfo class starts off by creating an inline function called discovered which is called by connecting the *discovered* signal, near the end of the init function, to the *discovered* function. The init function also creates sever class instance variables that is used to store information about the media file.

If the file loaded is detected as being a media file it is sent to the method *succeed.* If the file is not a media file the fail method is called, an error message is printed to the console, and the class variable self.__is_media is set to false indicating the file is not a media file.

If the succeed method is called it will set the class variable self.__is_media to true. It will then set the mime type using the variable self.__mimetype. It will check to see if the media file is an audio file or a video file. It will set self.__is_video to true if it is a video file.

The height and width of a video file is stored respectively in the variables self.__video_height and self.__video_width.

Then there are a few methods to retrieve these variables that are of interest to the programmer. The methods return the variables that are obviously in their names. get_height() returns self.__video_height and so on with the other methods.

The only other method that is included is the *poll* method. Since the discovery of media information is not instantaneous, if you attempt to use any of the get methods to retrieve information such as the height or width of a video, it may return the initialization values of the variables which is zero.

The *poll* method will return True when the *succeed* method has finished, indicating that all the variables have been assigned.

If the poll method returns False you must wait to use the information provided by the MediaInfo class.

### 6.9.2 GstPlayer Class

The GstPlayer class is used to control the GStreamer instance, watch the bus, handle GStreamer errors and messages, and sync the GStreamer video to video widget created with the VideoWidget class below.

```
class GstPlayer(object):
    def __init__(self, videowidget):
        # Setup GStreamer
        self.videowidget = videowidget
        self.player = gst.element_factory_make(“playbin”, “MultimediaPlayer”)
        bus = self.player.get_bus()
        bus.add_signal_watch()
        bus.enable_sync_message_emission()
        #used to get messages that gstreamer emits
        bus.connect(“message”, self.on_message)
        #used for connecting video to your application
        bus.connect(“sync-message::element”, self.on_sync_message)
    def set_location(self, location):
        self.player.set_property(“uri”, “file://” + location)
    def play(self):
        print “playing”
        self.player.set_state(gst.STATE_PLAYING)
    def pause(self):
        print “paused”
        self.player.set_state(gst.STATE_PAUSED)
    def stop(self):
        print “stoped”
        self.player.set_state(gst.STATE_NULL)
    def on_message(self, bus, message):
        if message.type == gst.MESSAGE_EOS: # End of Stream
            self.player.set_state(gst.STATE_NULL)
        elif message.type == gst.MESSAGE_ERROR:
            self.player.set_state(gst.STATE_NULL)
            (err, debug) = message.parse_error()
            print “Error: %s” % err, debug
    def on_sync_message(self, bus, message):
        if message.structure is None:
            return False
        if message.structure.get_name() == “prepare-xwindow-id”:
        self.videowidget.set_sink(message.src)
        message.src.set_property(“force-aspect-ratio”, True)
```

The GstPlayer class starts off with a video widget being passed in. This video widget is used to sync the GStreamer video with. After this the GStreamer pipeline is created using gst.element_factory_make using the *playbin* element and with the identifier MultimediaPlayer.

```
self.player = gst.element_factory_make(“playbin”, “MultimediaPlayer”)
```

Next a bus is created to be used with the pipeline and signal watching is added. Signals are connected to the self._on_message method.

```
bus = self.player.get_bus()
bus.add_signal_watch()
bus.connect(“message”, self.on_message)
```

After this is used the enable_sync_message_emission() method on the bus to enable the player to sync the GStreamer video with the video widget that is being used. If this is not done, a separate window will be opened that is outside of the running application to play the video in. The sync emissions signals are directed to the self.on_sync_message method.

```
bus.enable_sync_message_emission()
bus.connect(“sync-message::element”, self.on_sync_message)
```

The on_sync_message method uses the video widget that has been passed in during the initialization of GstPlayer to attach the video to the application. This is a little hocus pocus that I am not really sure of what is happening. It seems to be communicating somewhat with the underlying X Windows about the window id.

Then the set_sink method that is in video widget class is set to the message src. The set_sink method is used for convenience. The VideoWidget class that is discussed in the next section.

The very last part is to set force the aspect ratio of the video.

```
def on_sync_message(self, bus, message):
    if message.structure is None:
            return False
        if message.structure.get_name() == “prepare-xwindow-id”:
            self.videowidget.set_sink(message.src)
            message.src.set_property(“force-aspect-ratio”, True)
```

The rest of what is covered by the GstPlayer class is very self explanatory. A *play*, *stop*, and *pause* method to play, pause, or stop the media file. And there is also a method call *set_ location* that sets the location of the media file that is to be played.

### 6.9.3 VideoWidget

The VideoWidget class is a subclass of gtk.DrawingArea. It is created to ease the use of displaying videos inside of applications and is used in conjunction with the user interface and GstPlayer class.

```
class VideoWidget(gtk.DrawingArea):
    “””
    Extend gtk.DrawingArea to create our own video widget.
    “””
    def __init__(self):
        gtk.DrawingArea.__init__(self)
        self.imagesink = None
        self.unset_flags(gtk.DOUBLE_BUFFERED)
    def do_expose_event(self, event):
        if self.imagesink:
            self.imagesink.expose()
            return False
        else:
            return True
    def set_sink(self, sink):
        if sys.platform == “win32”:
            win_id = self.window.handle
        else:
            win_id = self.window.xid
        assert win_id
        self.imagesink = sink
        self.imagesink.set_xwindow_id(win_id)
```

VideoWidget starts off with the initialization of the DrawingArea parent class. It then proceeds to set the imagesink to none. Last in the initialization is to unset the double buffering.

The set_ sink method asserts that the window xid is available and then proceeds to set the imagesink to the sink that is passed in. The sink is passed in from the GstPlayer class. Then while still in the set_ sink method the imagesink sets the id of the xwindow to self.window.xid.

With the VideoWidget class now completed it is simple to use to display videos. To use it with a PyGTK application you would just initialize in your PyGTK GUI code like so:

```
videowidget = VideoWidget()
```

### 6.9.4 User Interface

A good user interface is required to make the ability to play media files useful. The user interface will allow the user to interact with the program, open audio or video files, and if the file is a video then resize it to fullscreen.

```
class Main(object):
    """
    The Main class is the Gui. It creates an instance of the
    GstPlayer class and the FileInfo class. It is what the user
    interacts with and controls what happens.
    """
    def __init__(self):
        #Store the location of the multimedia file
        self.multimedia_file = None
        # To be used with the FileInfo Class
        self.file_info = None

        # Create the GUI
        self.win = gtk.Window()
        self.win.set_title("Play Video Example 2")
        self.win.connect("delete_event", lambda w,e:
             gtk.main_quit())
        vbox = gtk.VBox(False, 0)
        self.control_box = gtk.HBox(False, 0)

        # Control Buttons
        self.load_file_button = gtk.FileChooserButton(
            "Choose Audio File")
        self.play_button = gtk.Button("Play",
            gtk.STOCK_MEDIA_PLAY)
        self.pause_button = gtk.Button("Pause",
            gtk.STOCK_MEDIA_PAUSE)
        self.stop_button = gtk.Button("Stop", gtk.STOCK_MEDIA_STOP)

        # Video Widget Stuff
        self.videowidget = VideoWidget()
        self.videowidget.set_size_request(400, 250)

        # Signals and Callbacks
        self.load_file_button.connect("selection-changed",
            self.on_file_selected)
        self.play_button.connect("clicked", self.on_play_clicked)
        self.pause_button.connect("clicked", self.on_pause_clicked)
        self.stop_button.connect("clicked", self.on_stop_clicked)
        # Fullscreen stuff
        self.win.connect("key-press-event",
            self.on_win_key_press_event)
        self.win.connect("window-state-event",
            self.on_window_state_event)

        self.control_box.pack_start(self.play_button,
            False, True, 0)
        self.control_box.pack_start(self.pause_button,
            False, True, 0)
        self.control_box.pack_start(self.stop_button,
            False, True, 0)
        vbox.pack_start(self.load_file_button, False, True, 0)
        vbox.pack_start(self.videowidget, True, True, 0)

        # You want to expand the video widget or else you
        #cannot see it
        vbox.pack_start(self.control_box, False, True, 0)
        self.win.add(vbox)
        self.win.show_all()
        self.gst_player = GstPlayer(self.videowidget)

    def fullscreen_mode(self):
        """
        Called from the on_win_key_press_event method. If the
        program is in fullscreen this method will unfullscreen
        it. If the program is not in fullscreen it will set it
        to fullscreen. This method will also hide the controls
        while in fullscreen mode.
        """
        if self.__is_fullscreen:
            self.win.unfullscreen()
            self.control_box.show()
            self.load_file_button.show()
        else:
            self.control_box.hide()
            self.load_file_button.hide()
            self.win.fullscreen()

   def on_win_key_press_event(self, widget, event):
        """
        Handle any key press event on the main window.
        This method is being used to detect when the ESC key
        is being pressed in fullscreen to take the
        window out of fullscreen
        """
        key = gtk.gdk.keyval_name(event.keyval)
        if key == "Escape" or key == "f":
            self.fullscreen_mode()

   def on_window_state_event(self, widget, event):
        """
        Detect window state events to determine whether in
        fullscreen or not in fullscreen
        """
        self.__is_fullscreen = bool(event.new_window_state &
        print "Is fullscreen: ", self.__is_fullscreen

    def on_file_selected(self, widget):
        print "Selected: ", self.load_file_button.get_filename()
        self.multimedia_file = self.load_file_button.get_filename()

        # Do not call method from here immediately.
        # FileInfo.poll() will return false when it is ready.
        # Usually a second or two.
        self.file_info = MediaInfo(self.multimedia_file)
        self.gst_player.set_location(
             self.multimedia_file )

    def on_play_clicked(self, widget):
        print "play clicked"
        print "Video (width, height): ",
            self.file_info.get_width(),
            self.file_info.get_height()

        self.videowidget.set_size_request(
            self.file_info.get_width(),
            self.file_info.get_height() )

        self.gst_player.play()

    def on_pause_clicked(self, widget):
        print "pause clicked"
        self.gst_player.pause()

    def on_stop_clicked(self, widget):
        print "stop clicked"
        self.gst_player.stop()

if __name__ == "__main__":
        Main()
        gtk.main()
```

The Main class starts off by initializing a few variables and creating the required user interface code.

The multimedia_file is set to None and will store the location of the media file that is to be played.

The file_info variable is set to None and will be later used to initialize the MediaInfo class.

After declaring the first class instance variables the Main class creates the user interface, sets the title to "Play Video Example 2", and adds a few buttons to play, pause, and stop the video. It also adds a gtk.FileChooserButton video to select the media files that will be played.

Next it creates a video widget using the VideoWidget class that was described above.

At the very bottom of the of the __init__ method the class variable self.gst_player is initialized as an instance of the GstPlayer class using the self.videowidget instance that was created.

```
    self.gst_player = GstPlayer(self.videowidget)
```

On the clicked signal is emitted from the play, pause, or stop button; then the methods on_play_clicked, on_pause_clicked, and on_stop_clicked are called respective to their buttons.

When a file is selected with the load_file_button the on_file_selected method is called.

In the section of the code commented as full screen stuff it will connect key-press-event signals and window-state-event signals to the on_win_key_press_event and on_window_state_event methods. The on_win_key_press_event method will detect if the key pressed is the "F" or "Esc" key and if so call the fullscreen_mode() method. If it is fullscreen it will unfullscreen the video. If it is not in full screen it will set it to fullscreen.

The on_window_state_event detects changes in the window state. All that it is used for is to set the variable self.__is_fullscreen to True or false. This variable is used in the method fullscreen_mode() to either hide the control buttons (play, pause, stop, load_file) or show them. If the variable is set to False it will set these widgets to be displayed and unfullscreen the video widget. If the self.__full_screen is True it will hide all the control widgets and set the video widget to fullscreen with self.win.fullscreen().

Next is the on_file_selected method. This method is called when the a file selected from the gtk.FileChooserButton load_file_button. It stores the location of the file in the variable self.multimedia_file. After the file location has been stored it is used to create an instance of the MediaInfo class.

```
    self.file_info = MediaInfo(self.multimedia_file)
```

And finally in the on_file_selected method the location of the media file is loaded into the the GstPlayer instance self.gst_player.

```
    self.gst_player.set_location(self.multimedia_file)
```

The on_play_clicked method will use the MediaInfo instance self.file_info to get the width and height of the video and set the size of the self.videowdiget to the correct dimensions to display the video.

After this it will set the video playing by calling the play method in the GstPlayer class:

```
self.gst_player.play()
```

As with the on_play_clicked method the on_pause_clicked and on_stop_clicked methods will use call the pause and stop methods from the GstPlayer class.

```
self.gst_player.pause()
self.gst_player.stop()
```

The only thing that is left to do is to make sure the Main class is called and this is accomplished at the bottom of the source code, which detects if this is file is the main file being run and enters the GTK mainloop.

```
if __name__ == “__main”:
    Main()
    gtk.main()
```

## 6.10 Summary

Basically GStreamer is a very powerful framework with many options available. Even though this chapter only covered a small portion of what is available, as a programmer you should now be able to add audio and video play back to your application.

As well as seek the position of the media file and display the current location and length of the file to the user.

For more information using GStreamer visit the following sites:

1. http://pygstdocs.berlios.de/ Contains tutorials and documentation on python GStreamer

2. The main C documentation. If you do not know C this may not be of use, but it may, anyway http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/index.html

3. Check out all the examples that come with the PyGST source at http://webcvs.freedesktop.org/gstreamer/gst-python/examples/

4. And of course check out the examples that come with this books website http://www.majorsilence.com/rubbish/pygtk-book/

# Chapter 7

# Dbus Interprocess Communication

Please send any fixes or suggestions to peter@majorsilence.com or leave a comment at http://www.majorsilence.com/pygtk_book.

## 7.1 Introduction

DBus is used for interprocess communication between applications. Simply put this means that applications can retrieve information from one another by accessing special methods that are provided by DBus.

**ObjectPath** An application may export an object to represent itself or different parts of itself. For example Rhythmbox exports an object representing the Play List and an object representing the current playing song. Eg. /org/gnome/Rhythmbox/Player

**BusName** The name of the application as exposed through DBus. Eg. org.gnome.Rhythmbox

**Interface** Is used to access methods through DBus. Eg. org.gnome.Rhythmbox

This chapters purpose is to show how to control other applications with DBus and how to add DBus to your PyGTK applications so that you can expose functionality of your applications to others.

## 7.2 Controlling Applications

First off is going to be an example of how to use dbus to communicate with another application. This example will communicate with the rhythmbox music player. The reason for using rhythmbox is because it is it is rather ubiquitous in the gnome distro world.

```
#!/usr/bin/env python
import os, gobject, dbus
from dbus.mainloop.glib import DBusGMainLoop
import gtk
```

The above code imports the needed code to work with this example. What is needed to work with DBus is the *dbus* module and *DBusGMainLoop*. The dbus module is used for the common dbus interactions while DBusGMainLoop is used to work with gobject main loops, which PyGTK uses.

Here the class DBusExample is created with the __init__ method setting up the dbus.

```
class DBusExample(object):
  def __init__(self):
    # Do before session or system bus is created.
    dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
    self.bus = dbus.SessionBus()

    self.proxy_object = self.bus.get_object('org.gnome.Rhythmbox',
        '/org/gnome/Rhythmbox/Player')
    self.player = dbus.Interface(self.proxy_object,
        'org.gnome.Rhythmbox.Player')

    self.bus.add_signal_receiver(self.on_song_changed,
        dbus_interface="org.gnome.Rhythmbox.Player",
        signal_name="playingUriChanged")

    self.init_gui()
    self.list_available_commands()
```

To begin with a DBus SessionBus is created. This allows connecting to other applications. If this example were connecting to a system process it would use a SystemBus. Once the bus is created, a proxy object is assigned to self.proxy_object using the self.bus.get_object method. The get_object method takes as arguments the applications Bus Name(113) and Object Path(113).

After the proxy_object has been created it is used to create an interface to the available methods. The interface self.player is created using the dbus.Interface class. It is initlized with the self.proxy_object and using the org.gnome.Rhythmbox.Player interface. This interface provides for methods to control and retrieve information on the currently playing song.

Below this a signal handler is created on the bus to catch the playingUriChanged Signal from the interface org.gnome.Rhythmbox.Player and call the on_song_changed method.

Lastly in the __init__ method the init_gui method is called. The init_gui method is rather insignificant as it creates a small gui with a few buttons. However the callback methods for those buttons use the self.player interface to control rhythmbox.

```
  def init_gui(self):
    win = gtk.Window()
    win.connect("delete_event", lambda w,e:gtk.main_quit())
    vbox = gtk.VBox()
    hbox = gtk.HBox()

    self.output = gtk.Label("")
  vbox.pack_start(self.output, False, True, 0)
  mute=gtk.Button("Mute")
  play_pause=gtk.Button("Play/Pause")
  previous=gtk.Button("Previous")
  next=gtk.Button("Next")

  mute.connect("clicked", self.on_mute_clicked)
  play_pause.connect("clicked", self.on_play_pause_clicked)
  previous.connect("clicked", self.on_previous_clicked)
  next.connect("clicked", self.on_next_clicked)

  hbox.pack_start(mute, False, True, 0)
  hbox.pack_start(play_pause, False, True, 0)
  hbox.pack_start(previous, False, True, 0)
  hbox.pack_start(next, False, True, 0)
  vbox.pack_start(hbox, False, True, 0)
  win.add(vbox)
  win.show_all()
```

The init_gui method above creates a small user interface with a play/pause, mute, previous and next button to control rhythmbox. It also as a label that is used by the on_song_changed callback method, that was specified in the __init__ method, to display the path and name of the current playing song.

```
def on_mute_clicked(self, widget):
  if self.player.getMute():
    self.player.setMute(False)
  else:
    self.player.setMute(True)
```

The on_mute_clicked method checks to see if rhythmbox is muted, if it is it will unmute it. If it is not muted it will set it to mute. This is accomplished using the self.player interface with the setMute method, which takes a boolean argument.

```
def on_play_pause_clicked(self, widget):
  if self.player.getPlaying():
    self.player.playPause(False)
  else:
    self.player.playPause(True)
```

The on_play_pause_clicked method will set rhythmbox to play if it is paused and pause it if it is playing. This is accomplished using the self.player interface with the playPause method, which takes a boolean argument.

```
def on_previous_clicked(self, widget):
  self.player.previous()
```

The on_previous_clicked method will set rhythmbox to play the previous played song. This is accomplished using the self.player interface with the previous method.

```
    def on_next_clicked(self, widget):
      self.player.next()
```

The on_next_clicked method will set rhythmbox to play the next song. This is accomplished using the self.player interface with the next method.

```
    def on_song_changed(self, data):
      path, filename = os.path.split(self.player.getPlayingUri())
      self.output.set_text("Path: " + path + "\nFilename: " + filename)
```

The on_song_changed method is called when the playingUriChanged signal is emitted. It retrieves the current songs current uri, splitting it into a path and file name, and displays it using a gtk label. It should also be pointed out that instead of using the getPlayUri() method, the data argument could be used as it is the uri of the current song as well.

And last lets not forget the small amount of code to run this example

```
  if __name__ == "__main__":
    app = DBusExample()
    gtk.main()
```

## 7.3 Adding DBus to your Applications

Controlling other applications using DBus is one thing but it is not enough if you application needs to allow others to control it. To let other applications have access to your program requires explosing methods of sub class of dbus.service.Object.

### 7.3.1 Creating a DBus Service

To start off a few modlues need to be imported. The import ones are the DBus ones.

```
  #!/usr/bin/env python
  import os, gobject, dbus, dbus.service
  from dbus.mainloop.glibimport DBusGMainLoop
  import gtk
  output_label = None
```

So of the above modules dbus, dbus.service and DBusGMainLoop are what are important for allowing other applications to connect to his one. After the imports there is the output_label which will be used as a global to create a gtk.Label to display messages that are received through DBus.

After this DBusObject class is created; it can be named whatever you want as long as it subclasses dbus.service.Object. As you will see it is not necessary to create __init__ method with this class as the parent classes can be used.

```
class DBusObject(dbus.service.Object):
 # Display and message to gtk label and return message to caller
 @dbus.service.method(’com.majorsilence.MessageInterface’,
   in_signature=”, out_signature=’s’)
 def display_welcome_message(self):
  global output_label
  output_label.set_text("Welcome to dbus.")
  return "Welcome to dbus."
```

To expose methods for the @dbus.service.method decorator is used, specifying the DBus Interface that the method will available on and the methods in (arguments) and out (return value) signatures. Here the interface is specified as com.majorsilence.MessageInterface, so any application calling this method would have to use com.majorsilence.MessageInterface. After the decorator declare the method as normal. The method name is the same name that will be exposed.

So what we end up with here is a method called display_welcome_message that returns a string, s meaning it is a dbus.String type (see 7.5 on page 121). As can be seen it sets the label to “Welcome to dbus” and returns the same message to the calling program.

Moving on to the next method, it takes a string as an argument emits a signal and completion and returns nothing.

```
# Set gtk label to the message that is passed
 @dbus.service.method(dbus_interface=’com.majorsilence.MessageInterface’, in_signature=’s’, ou
 def set_message(self, s):
  global output_label
  if not isinstance(s, dbus.String):
   print "not string"
   return
  output_label.set_text(s)
  #emit signal
  self.message_signal()
```

As before and like all exposed DBus methods the @dbus.service.method decorator is used. This method has the same DBus Interface as the first method, com.majorsilence.MessageInterface, and an in_signature of s meaning a dbus.String (see 7.5 on page 121).

The method is set_message, it takes as an argument a string. It checks to make sure it was passed a string, if it was it will set the label to the string that was passed in. The interesting thing about this method compared to the first one is that it emits a signal on completion. It does this by calling the self.message_signal() method as its last act.

The self.message_signal is the method that is described next. It to uses a dbus decorator, but instead of using the @dbus.service.method decorator, it uses the @dbus.service.signal decorator. What this means is that when this method is called it will emit a signal that can be caught using the add_signal_receiver method that was described in 7.2 on page 113.

```
 @dbus.service.signal(’com.majorsilence.MessageInterface’)
 def message_signal(self):
  return
```

As can be seen the message_signal method uses the @dbus.service.signal decorator and specifies the com.majorsilence.MessageInterface. If it is to include data with its signal it should also have a out_signature specifying the correct type.

All that is left is the the main() function that is used to setup a very small PyGTK GUI and create the neccesary DBus initiation.

```
def main():
  # Create GTK Gui
  global output_label
  win = gtk.Window()
  win.connect("delete_event", lambda w,e:gtk.main_quit())
  output_label = gtk.Label("This message will change through using dbus.")
  win.add(output_label)
  win.show_all()

  # Start DBus Service
  dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
  session_bus = dbus.SessionBus()
  name = dbus.service.BusName("com.majorsilence.MessageService", session_bus)
  object = DBusObject(session_bus, "/TestObject")

  gtk.main()
```

The important part of the code starts after the # Start DBus Service comment. These four lines of code are what makes available dbus and makes it possible to expose method of the application to any other DBus capable program. First DBus must be set to use the glib gobject main loop (the same that PyGTK uses), without this it will not work. Next it creates a session bus that allows applications to connect to a bus. After this it uses the session bus to create a bus name using the dbus.service.BusName class. It takes as arguements the session bus that was created and the interface com.majorsilence.MessageService.

Finally the object is create calling the DBusObject class that we have created, using the session bus that we have created and using the /TestObject object path.

```
if __name__ == "__main__":
  main()
```

Of course do not forget to call the main function that runs the the example PyGTK DBus service application.

### 7.3.2 Connecting to your DBus Service

Controlling your own application through DBus is very similiar to how the first example controlled Rhythmbox. This is a small application that will call the two exposed methods from 7.3.1 and handle the signal that is emitted.

```
#!/usr/bin/env python
```

```
import os, gobject,dbus
from dbus.mainloop.glib import DBusGMainLoop
import gtk
class DBusClient(object):
  def __init__(self):
    # Do before session or system bus is created.
    dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
    self.bus = dbus.SessionBus()
    self.proxy = self.bus.get_object('com.majorsilence.MessageService',
        '/TestObject')
    self.control_interface = dbus.Interface(self.proxy,
        'com.majorsilence.MessageInterface')
    self.bus.add_signal_receiver(self.on_message_recieved,
        dbus_interface="com.majorsilence.MessageInterface",
        signal_name="message_signal")
```

As can be seen, connect to the bus name com.majorsilence.MessageSerive using the objec path /TestObject. Next the interface is created using self.proxy and the interface com.majorsilence.MessageInterface. Finally the signal message_signal is handled by connecting it to the self.on_message_recieved method when it is emitted from the com.majorsilence.MessageInterface interface.

```
    win = gtk.Window()
    win.connect("delete_event", lambda w,e:gtk.main_quit())
    vbox = gtk.VBox()
    hbox = gtk.HBox()

    self.text_message=gtk.Entry()
    set_message=gtk.Button("Set Message")
    display_message=gtk.Button("Display Welcome Message")
    set_message.connect("clicked", self.on_set_message_clicked)

    display_message.connect("clicked",
        self.on_display_message_clicked)

    hbox.pack_start(set_message, False, True, 0)
    hbox.pack_start(display_message, False, True, 0)
    vbox.pack_start(self.text_message, False, True, 0)
    vbox.pack_start(hbox, False, True, 0)
    win.add(vbox)
    win.show_all()

  def on_message_recieved(self):
    print "message_signal caught"
```

When the signal is emitted it does nothing prints a message to the console.

```
    def on_set_message_clicked(self, widget):
      message = self.text_message.get_text()
      self.control_interface.set_message(message)
```

When the set message button is clicked it grabs the text from the text entry and uses the self.control_interface to set the label in the serve appliction to whatever text was typed in.

```
    def on_display_message_clicked(self, widget):
      print self.control_interface.display_welcome_message()
```

When the display message button is clicked it calls the exposed method display_welcome_message() which is a method with a predfined message that is displayed to the DBus service applications label.

```
  if __name__ == "__main__":
    app = DBusClient()
    gtk.main()
```

The code to to actually run the example.

## 7.4 Finding Exposed Methods

Now you are asking yourself "it is all good and well that I can access functionalty throught DBus, but how do I find what is available?". Well this is actionally fairly simple and is accomplished using introspection. Basically form is

```
  your_interface.Introspect(dbus_interface='org.freedesktop.DBus.Introspectable')
    def list_available_commands(self):
```

Here is an example using rhythmbox. It lists all the available methods, signals and properties of the interface that is used. It is printed as xml as that is the form that DBus uses.

```
  import gobject, dbus
  from dbus.mainloop.glib import DBusGMainLoop

  dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
  bus = dbus.SessionBus()
  proxy_object = bus.get_object('org.gnome.Rhythmbox',
      '/org/gnome/Rhythmbox/Player')
  player = dbus.Interface(proxy_object,
      'org.gnome.Rhythmbox.Player')
  print player.Introspect(dbus_interface=
          'org.freedesktop.DBus.Introspectable')
```

That is all that is to it, very simple, very easy.

## 7.5 Types

When using introspection a print out of the xml will be displayed, for instance a piece of it may look like this.

```
<method name="playPause">
  <arg name="arg0" type="b" direction="in"/>
</method>
```

What this small piece means is that there is a method that is available called *playPause*. It takes one argument. Its type is b meaning it is a boolean. The direction is in, meaning it recieves input, if the direction is out is returns a value.

It is important to know what the different types are so here is a list.

| | |
|---|---|
| b | dbus.Boolean, bool |
| d | dbus.Double, float |
| g | dbus.Signature |
| i | dbus.Int32, int |
| n | dbus.Int16 |
| o | dbus.ObjectPath |
| q | dbus.UInt16 |
| s | dbus.String, dbus.UTF8String, str, unicode |
| t | dbus.UInt64 |
| u | dbus.UInt32 |
| x | dbus.Int64, long |
| y | dbus.Byte |

DBus also supports for container types.

ax dbus.Array, list - a is an array and the x is the type that is used. X here means it is an array of dbus.UInt64/long

ay dbus.ByteArray, str - Is a more effienct array

(types) dbus.Struct, tuple - The signature of is either None or a string representing the contents of the struct. The signature '(iis)' would be used for two integers and a string.

a{xy} dbus.Dictionary, dict - a is the key and y is the value. So a{si} would be a dictionary with strings for keys and integers for values.

v variants

## 7.6 Summary

Although you are probably not a DBus expert from this chapter, it should have given you a good enough understanding to start accessing other applications and add some basic support to your own application. What you need to do is experiment a little and make sure that you fully understand DBus and maybe read up on it a little more.

Some other resources that you may want to check out are:

- `http://dbus.freedesktop.org/doc/dbus-python/doc/tutorial.html`
- `http://dbus.freedesktop.org/doc/dbus-python/`
- `http://dbus.freedesktop.org/doc/dbus-python/api/index.html`
- `http://www-128.ibm.com/developerworks/linux/library/l-dbus.html`
- `http://www.madsoft.org/2008/06/10/interfacing-banshee-10-with-dbus-and-python/`
- `http://mumble.sourceforge.net/DBus`

# Chapter 8

# Clutter

Please send any fixes or suggestions to peter@majorsilence.com or leave a comment at http://www.majorsilence.com/pygtk_book.

## 8.1 Introduction

Note: This chapter is about pyclutter 1.0 and no longer contains information on older versions.

The best way to describe clutter is to quote its home page which states:

Clutter is an open source software library for creating fast, visually rich and animated graphical user interfaces.

Clutter can be integrated with many Linux technologies including GStreamer, Cairo a GTK+. It also is portable and runs on Windows and OSX which allows for cross platform goodness.

But how is clutter used? This is actually very simple. Instead of creating a gtk.Window as with using PyGTK, with clutter a clutter.Stage is created. And instead of using widgets, Actors are used. This is actually rather neat. We have Stages on which to do our work and Actors that perform.

Some base Actors included with clutter are:

- Texture
- CloneTexture
- Text - For all things text. Replaces Label and Entry
- Rectangle - For creating Rectangles
- Label - displaying Labels (deprecated as of 1.0)
- Entry - For entering text (deprecated as of 1.0)

A stage is created by using the clutter.Stage object like so:

```
stage = clutter.Stage()
```

The size of stage can be set:

```
stage.set_size(800, 400)
```

The title of the stage can be set using the stage.set_title() method:

```
stage.set_title("Hey Hey, My First Clutter App")
```

To make sure that a clutter application is shut down properly make sure to add

```
stage.connect('destroy', clutter.main_quit)
```

## 8.2 Colors

The colour of a stage can be set using set_color() method and using the clutter.color_from_string()[1] method. The clutter parse method can take several different colour inputs including the colours as Text or RGB Notation.

The colour of a stage can be set:

```
stage.set_color(clutter.color_from_string("red"))
stage.set_color(clutter.Color(255, 0, 0))
```

Colours can be applied to more then just stages, they may also be applied to all the Actors that will be shown in the next section.

## 8.3 User Input

### 8.3.1 Keyboard

It is very easy to catch user input. For catching keyboard events the stage must connect the "key-press-event" to a handler.

```
# key-press-event is for the keyboard
stage.connect("key-press-event", on_key_press_event)
def on_key_press_event(self, stage, event):
    # event has the following:
    # event.hardware_keycode
    # event.keyval - ascii (or unicode) value of the key
    # event.modifier_state
    # event.put
    # event.source
    # event.time
    # event.type
    print "keyval ", event.keyval
    try:
```

[1] Formally clutter.color_parse(...)

```
            print “key pressed “, chr(event.keyval)
        except ValueError:
            print “Key pressed not recognized ascii character.”
            print “Returning from key pressed function”
            return
```

What is probably going to be the most useful when handling a key press event is the character pressed. The event.keval can be converted using the builtin function *chr*. For example in ascii 97 is a. So if we press a and have the following code

```
print chr(event.keyval)
```

an “a” will be printed to the screen.

### 8.3.2 Mouse

Catching input from a mouse is also very easy and all that needs to be done is to handle the “button-press-event”. The handler function takes two arguments “stage” and “event” and is handled like so:

```
stage.connect(“button-press-event”, on_button_press_event)
def on_button_press_event (stage, event):
    #mouse button
    #event.button - 1 left click
    #             - 3 right click
    #             - 2 left and right clicked same time
    #
    # event.x - X Coordinate of button press
    # event.y - Y Coordinate of button press
    #
    print “mouse button %d pressed at (%d, %d)” % (event.button, event.x, event.y)
```

## 8.4 Actors

### 8.4.1 Text

```
class clutter.Text(clutter.Actor)
    get_text()
    set_text(text) - Sets the text in the entry
    set_editable(Boolean)
    set_reactive(Boolean)
    set_position(xPos, yPos)
    ...
```

You can set text in it using the set_text(text) method and get text using the get_text() method. If you want to use the Text actor as a label you would set it to *set_editable(False)*. Here is an example with a Text actor being created and added to a stage. You will have to click on the Text actor to be able to type in it, just like a normal gtk or windows text field.

```
class EntryExample:
    def __init__(self):
       self.stage = clutter.Stage()
       self.stage.set_size(400, 400)
       self.stage.set_color(clutter.color_from_string("red"))
       self.text = clutter.Text()
       self.text.set_text("Text Entry")
       self.text.set_color(clutter.color_parse("green"))
       self.text.set_size(150, 50)
       self.text.set_position(200, 200)
       self.text.set_reactive(True)
       self.text.set_editable(True)

       self.text.connect("button-press-event",
          self.on_mouse_press_event)
       self.text.connect("key-press-event",
          self.on_key_press_event)
       self.stage.connect(’destroy’,
          clutter.main_quit)

        self.stage.add(self.text)
        self.stage.show_all()

    def on_mouse_press_event(self, actor, event):
       self.stage.set_key_focus(self.text)
       return False

    def on_key_press_event(self, actor, event):
        print "Text Actor is: ", actor.get_text()
        print "Key pressed is: ", unichr(event.keyval)

if __name__ == "__main__":
    app = EntryExample()
    clutter.main()
```

As can be seen in the code above the stage color is set to red, the text color in the Text actor is set to green. The text actor is set to have width of 150 and a height of 50. The position of the Text actor on the stage is set to x 200 y 200. The actor is also set to be editable and reactive. Simple enough.

Three call back functions are added. One callback for when a mouse button is pressed over the Text actor. One callback event for when a key is pressed on the keyboard while the focus is in the Text actor. Finally the last callback event is for the stage destroy signal which is connected above to *clutter.main_ quit* to insure the program exits properly.

### 8.4.2 Rectangles

```
class clutter.Rectangle(clutter.Actor):
    clutter.Rectangle(color=None)
    get_color()
    set_color(color)
    get_border_color()
    set_border_color(color)
    get_border_width()
    set_border_width(width)
```

Creating rectangles does not entail much. Basically you just call the clutter.Rectangle class and you are done. Of course you will probably want to do more to it then that and add it to a stage. As can be seen in the class outline of a clutter.Rectangle above there is not much to work with in and of themselves making them easy to work with.

Setting a border width on a Rectangle will increase its size bye the border size multiplied by 2. So if your rectangle is set to 200 wide and you add a border of 20 you end up with a rectangle that is 240 wide.

### 8.4.3 Textures

```
class clutter.Texture(Actor)
        set_area_from_rgb_data(data, has_alpha, x, y, width, height, rowstride, bpp, flags, err
            - Updates a sub-region of the pixel data in a Texture.
        set_from_rgb_data(data, has_alpha, width, height, rowstride, bpp, flags, error)
            - Sets the Texture from RBG data.
        set_from_yuv_data(data, width, height, flags, error)
            - Sets the Texture from a YUV image data.
        set_from_file(filename, error) - obvious
        set_filter_quality(filter_quality)
        ...
```

Loading an image file and setting it up as a texture is very simple.

```
purple_flower = clutter.Texture(filename=”flower.jpg”)
```

Look at that, only one line of code and we have a texture that is ready to be used in our pyclutter program. Now all that needs to be done is add the purple_flower to the stage.

```
import clutter
stage = clutter.Stage()
stage.set_size(400, 400)
purple_flower = clutter.Texture(filename="flower.jpg")
(width, height) = purple_flower.get_size()
stage.add(purple_flower)
stage.show_all()
stage.connect('destroy', clutter.main_quit)
clutter.main()
```

#### 8.4.3.1 Cloning a textue

```
class clutter.CloneTexture(Actor)
    get_parent_texture()
    set_parent_texture(Texture)
```

Cloning a texture is as easy to create as the original texture.

```
# Create the original Texture
purple_flower = clutter.Texture(filename=”flower.jpg”)
# Create the cloned Texture
clone = clutter.CloneTexture(purple_flower)
```

And it is that simple. Now the actual work starts with making the texture do what it is supposed to be doing. It should probably be properly placed, maybe have some behaviours (this is discussed later), setting up time lines.

Lets make the Texture and the cloned texture show up on the stage now.

```
clone.set_position(200, 200)
stage.add(purple_flower, clone)
stage.show_all()
stage.connect(’destroy’, clutter.main_quit)
```

Now the the original texture is in the upper most left corner of the stage and the clone is displayed at coordinates x 200 and y 200.

```
import clutter
def create_texture(fName):
    image = clutter.Texture(filename=fName)
    (width, height) = image.get_size()
    return image

stage = clutter.Stage()
stage.set_size(400, 400)

# Create the original Texture from a picture of a flower
purple_flower = create_texture("flower.jpg")

# Create a clone of the origial Texture cloned_flower = clutter.CloneTexture(purple_flowe
cloned_flower.set_position(200, 200)

stage.add(purple_flower, cloned_flower) stage.show_all() stage.connect(’destroy’, clutte
clutter.main()
```

### 8.4.4 Labels

Labels have been deprecated and as of pyclutter 1.0 you they should not be used. Instead the Text actor should be used and will be demonstrated here. To see more about the Text actor please see 8.4.1 on page 125.

```
class clutter.Text
    set_text(text)
    set_editable(Boolean)
    set_color(color)
    ....
```

So here is the example of use Text as a label. Bascially all that is being done is using the method *set_ editable(False)* to make sure users cannot change the text.

```
import clutter
stage = clutter.Stage()
stage.set_size(400, 400)
label = clutter.Text()
label.set_editable(False)
label.set_text("Clutter Label Text")
label.set_color(clutter.color_from_string("brown"))
# If no position is given it defaults to the upper most left corner.
stage.add(label)
stage.show_all()
stage.connect('destroy', clutter.main_quit)
clutter.main()
```

## 8.5 Animations

### 8.5.1 Timelines

```
class clutter.Timeline:
     __init__(duration)
         fps - Frames per second
         num_frames - The total number of frames
         duration - The duration of the timeline, in milliseconds
    def get_duration()
    def set_duration(msecs)
    def get_direction() - retrieves the direction of the timeline, either forward or backward.
    def set_direction()
    def get_loop()
    def set_loop(True or False) - Set the timeline to loop
    def get_progress()
    def start() - Start the timeline
    def pause() - Pause the timeline
    def stop()
    def rewind()
```

To use a time line you will want to add it to a clutter actor. The timeline is setup and then the animation effect that it is to be applied to it.

So lets create a time line with a duration of 3000(3 seconds).

```
timeline = clutter.Timeline(duration=3000)
```

Next we will set the timeline to loop using the set_loop() method. This sets the timeline to loop once it has finished each run through.

```
timeline.set_loop(True)
```

### 8.5.2 Alpha

```
class clutter.Alpha(goject.GObject)
    clutter.Alpha(timeline, func, data)
        def get_alpha()
        def get_timeline()
        def set_func(func)
        def set_timeline(timeline)
```

Next if you want to apply an animation you will want to setup the effect that is going to happen to your chosen object. So what is going to happen now is to create a clutter.Alpha object. Then create a clutter.BehaviourOpacity object using our just created alpha and apply this action to our rectangle. Then start the timeline running which will start the animation.

```
alpha = clutter.Alpha(timeline, clutter.EASE_IN_OUT_BOUNCE)
behaviour = clutter.BehaviourOpacity(0xdd, 0, alpha)
behaviour.apply(rect)
# start the timeline running, thus starting the animation
timeline.start()
```

The timeline can be setup anywhere and then started at any time using the timeline.start() method and stopped with the timeline.stop() method.

If we put all this together we get a working application that is only a few lines long.

What is needed to be known about clutter.Alpha is that it is a function of time not pixel form of alpha.

There are many predefined Clutter.Alpha functions that can be used with the clutter.Alpha class to effect the behaviour of the timeline. You will just have to experiment with them to see what suits your needs.

- clutter.EASE_IN_OUT_BOUNCE
- clutter.EASE_IN_BACK
- clutter.EASE_IN_BOUNCE
- clutter.EASE_IN_CIRC
- clutter.EASE_IN_CUBIC
- clutter.EASE_IN_ELASTIC

- clutter.EASE_IN_EXPO
- clutter.EASE_IN_OUT_BACK
- clutter.EASE_IN_OUT_CIRC
- clutter.EASE_IN_OUT_CUBIC
- clutter.EASE_IN_OUT_ELASTIC
- clutter.EASE_IN_OUT_EXPO
- clutter.EASE_IN_OUT_QUAD
- clutter.EASE_IN_OUT_QUART
- clutter.EASE_IN_OUT_QUINT
- clutter.EASE_IN_OUT_SINE
- clutter.EASE_IN_QUAD
- clutter.EASE_IN_QUART
- clutter.EASE_IN_QUINT
- clutter.EASE_IN_SINE
- clutter.EASE_OUT_BACK
- clutter.EASE_OUT_BOUNCE
- clutter.EASE_OUT_CIRC
- clutter.EASE_OUT_CUBIC
- clutter.EASE_OUT_ELASTIC
- clutter.EASE_OUT_EXPO
- clutter.EASE_OUT_QUAD
- clutter.EASE_OUT_QUART
- clutter.EASE_OUT_QUINT
- clutter.EASE_OUT_SINE

### 8.5.3 BehaviourOpacity

Please see the section 8.5.2 on page 130 before reading this section. Using Behaviour Opacity

```
import clutter
class Blinker:
    def __init__(self):
        self.stage = clutter.Stage()
        self.stage.set_color(clutter.color_from_string(“red”))
        self.stage.set_size(400, 400)
        self.stage.set_title(“My Blinking (BehaviourOpacity) Rectangle Example”)
        self.rect = clutter.Rectangle()
        self.rect.set_color(clutter.color_from_string(“green”))
        self.rect.set_size(200, 200)
        rect_xpos = self.stage.get_width() / 4
        rect_ypos = self.stage.get_height() / 4
        self.rect.set_position(rect_xpos, rect_ypos)
        self.timeline = clutter.Timeline(duration=3000)
        self.timeline.set_loop(True)
        alpha = clutter.Alpha(self.timeline, clutter.EASE_IN_OUT_SINE)
        self.behaviour = clutter.BehaviourOpacity(alpha=alpha , opacity_start=0xdd, opaci
        self.behaviour.apply(self.rect)
        self.timeline.start()
        self.stage.add(self.rect)
        self.stage.show_all()
        self.stage.connect(’destroy’, clutter.main_quit)
if __name__ == “__main__”:
    app = Blinker()
    clutter.main()
```

### 8.5.4 BehaviourRotate

```
class clutter.BehaviourRotate(Behaviour)
    clutter.BehaviourRotate(alpha(optional),  angle_end, angle_start)
        def get_axis()
        def set_axis(axis)
            -clutter.Z_AXIS
            -clutter.Y_AXIS
            -clutter.X_AXIS
        get_bounds(angle_start, angle_end)
        set_bounds(angle_start, angle_end)
        get_center(x, y, z)
        set_center(x, y, z)
        get_direction()
        set_direction(direction)
        ...
```

The clutter.BehaviourRotate class allows you to set a rotate behaviour on an a chosen actor.

The example that is to follow will create a Rectangle and add it to a stage and then we will create a time line. The timeline will control a BehaviourRotate that will affect the Rectangle (though any Actor will do).

At this point I will assume you know how to create a rectangle and add it to a stage, so from this point out I will just focus on the Behaviours.

So we have to create a timeline that will control the behaviour.

```
timeline = clutter.Timeline(duration=3000)
timeline.set_loop(True)
alpha = clutter.Alpha(timeline, clutter.EASE_IN_OUT_SINE)
```

Now that the timeline that is going to be used with the behaviour has been created we can create the behaviour itself. You will should notice that the axis is set to clutter.Z_AXIS, also available are clutter.X_AXIS and clutter.Y_AXIS.

```
rotate_behaviour = clutter.BehaviourRotate(axis=clutter.Z_AXIS, angle_start=0.0, angle_end=359.
rotate_behaviour.set_alpha(alpha)
rotate_behaviour.apply(rect)
```

So a instance of BehaviourRotate is created, rotating on the z axis, using the alpa timeline created above and this is applied to the rectangle instance rect.

```
import clutter
stage = clutter.Stage()
stage.set_size(400, 400)
rect = clutter.Rectangle()
rect.set_color(clutter.color_from_string("red"))
rect.set_size(100, 100) rect.set_position(150, 150)
timeline = clutter.Timeline(duration=3000)
timeline.set_loop(True)
alpha = clutter.Alpha(timeline, clutter.EASE_IN_OUT_SINE)
rotate_behaviour = clutter.BehaviourRotate(
    axis=clutter.Z_AXIS, angle_start=0.0, angle_end=359.0)
rotate_behaviour.set_alpha(alpha)
rotate_behaviour.apply(rect)
timeline.start()
stage.add(rect)
stage.show_all()
stage.connect('destroy', clutter.main_quit)
clutter.main()
```

### 8.5.5 BehaviourScale - Not Finished

```
class clutter.BehaviourScale(Behaviour)
```

```
    clutter.BehaviourScale(x_scale_start, y_scale_start, x_scale_end, y_scale_end, alpha
        def get_bounds()
        def set_bounds(x_scale_begin, y_scale_begin, x_scale_end, y_scale_end)
```

### 8.5.6 BehaviourDepth

```
class clutter.BehaviourDepth
    clutter.BehaviourDepth(depth_start, depth_end)
        def set_bounds(depth_start, depth_end)
        def get_bounds(depth_start, depth_end)
```

BehaviourDepth works like the other behaviours that have been discussed above. You create your behaviour specifying your desired action and attach it to a timeline.

With BehaviourDepth the only options (and required) are the start depth and end depth. Play around with them a little to get your desired result.

```
timeline = clutter.Timeline(duration=6000)
timeline.set_loop(True)
alpha = clutter.Alpha(timeline, clutter.EASE_IN_OUT_SINE)
rotate_behaviour = clutter.BehaviourDepth(0, 250)
rotate_behaviour.set_alpha(alpha)
rotate_behaviour.apply(rect)
```

All you have to do to start the required behaviour is start the time line and watch the results.

```
import clutter
stage = clutter.Stage()
stage.set_size(400, 400)
rect = clutter.Rectangle()
rect.set_color(clutter.color_from_string("red"))
rect.set_size(100, 100) rect.set_position(150, 150)
timeline = clutter.Timeline(duration=6000)
timeline.set_loop(True) alpha = clutter.Alpha(
    timeline, clutter.EASE_OUT_BOUNCE)
rotate_behaviour = clutter.BehaviourDepth(0, 250)
rotate_behaviour.set_alpha(alpha)
rotate_behaviour.apply(rect)
timeline.start()
stage.add(rect)
stage.show_all()
stage.connect('destroy', clutter.main_quit)
clutter.main()
```

## 8.6 Groups and Positioning

Groups allow the programmer to group together many actors. Instead of the Actor references the colors and position of the stage, they reference off of the group that they are in. Groups also allow for relative positioning, as in the positions of Actors are placed relative to their parent Group.

For example if you have a Rectangle and it is inside Group when you set the position of the rectangle to (100, 100), it is relative to the position of the group. So in the following snippet of code, the Rectangle is is being set to (100, 100) in the Group but relative to the stage it is set to (200, 200)

```
group = clutter.Group()
group.set_position(100, 100)
rect = clutter.Rectangle()
rect.set_size(100, 100)
rect.set_position(100, 100)
group.add(rect)
```

## 8.7 Summary

For more information about clutter you can visit its web site at:

http://www.clutter-project.org

For more information on the pyclutter api you can download the pyclutter source from the clutter project site and view the documentation or load the documentation in a python console using:

```
import clutter
help(clutter)
```

Also for each section of pyclutter covered in this chapter full source examples are on the books website. You are encouraged to download and inspect these and expand upon them to further learn how to best use clutter for your own projects. The address for these downloads is http://www.majorsilence.com/rubbish/pygtk-book/examples/.

# Chapter 9

# Embedded Web Browsers

Please send any fixes or suggestions to peter@majorsilence.com or leave a comment at http://www.majorsilence.com/pygtk_book.

## 9.1 Introduction

This chapter is going to explore how a PyGTK application can have the web embedded into them. It will cover using the firefox engine gecko, webkit, and Internet Explorer.

## 9.2 gtkmozembed

Using gtkmozemebed to embed mozilla firefox inside a PyGTK application is the easiest of the options in this chapter. A browser is initialized and added to the PyGTK window.

What is needed to start off with is to import the needed python modules.

```
import gtk
import gtkmozembed
```

Along with the browser the application will probably want some buttons to control how the web pages. Included will be a back, forward, refresh, stop, go, and home button. Also there will be a address bar. These will be connect to methods to control their actions.

Web Browser PyGTK Init Method

```
class ExampleBrowser(object):
  def __init__(self):
    data =
    """
<html><head><title>Hello</title></head> <body> PyGTK using MozEmbed to embed a web browser. </b
    """
    win = gtk.Window()
    win.set_size_request(800, 600)
```

```
        win.connect("delete_event", lambda w,e: gtk.main_quit())

        vbox = gtk.VBox(False, 0)
        control_box = gtk.HBox(False, 0)

        back = gtk.Button("Back")
        forward = gtk.Button("Forward")
        refresh = gtk.Button("Refresh")
        stop = gtk.Button("Stop")
        home = gtk.Button("Home")
        # no limit on address length
        self.address = gtk.Entry(max=0)
        go = gtk.Button("Go")

        control_box.pack_start(back, True, True, 2)
        control_box.pack_start(forward, True, True, 2)
        control_box.pack_start(refresh, True, True, 2)
        control_box.pack_start(stop, True, True, 2)
        control_box.pack_start(home, True, True, 2)
        control_box.pack_start(self.address, True, True, 2)
        control_box.pack_start(go, True, True, 2)

        back.connect("clicked", self.on_back_clicked, None)
        forward.connect("clicked", self.on_forward_clicked, None)
        refresh.connect("clicked", self.on_refresh_clicked, None)
        stop.connect("clicked", self.on_stop_clicked, None)
        home.connect("clicked", self.on_home_clicked, data)
        self.address.connect("key_press_event", self.on_address_keypress)
        go.connect("clicked", self.on_go_clicked, None)

        vbox.pack_start(control_box, False, True, 2) self.browser = gtkmozembed.MozEmbed()
        #gtkmozembed.set_profile_path("/tmp", "foobar")
        vbox.add(self.browser)

        win.add(vbox)
        win.show_all()
        ## self.browser.load_url('http://www.pygtk.org')
        self.browser.render_data(data, long(len(data)), 'file:///', 'text/html')
        # Load file from file system
        #self.browser.load_url('file:///path/to/file/name.html')
```

This code creates a PyGTK window and creates a control box, adds some buttons to the control box and then adds it to the window.

The import part to see though is the gtkmozembed part.

```
self.browser = gtkmozembed.MozEmbed()
self.browser.render_data(data, long(len(data)), 'file:///', 'text/html')
```

This small piece of code initializes the web browser and leaves the an instance with the name self.browser. It then displays the message, “PyGTK using MozEmbed to embed a web browser”.

With newly created browser it is now possible to access all the methods that are available such as going forward, backward, and entering addresses.

Mozilla Callback Methods

```
def on_back_clicked(self, widget=None, data=None):
  print "Back button clicked."
  if self.browser.can_go_back():
    self.browser.go_back()

def on_forward_clicked(self, widget=None, data=None):
  print "Forward button clicked."
  if self.browser.can_go_forward():
    self.browser.go_forward()

def on_refresh_clicked(self, widget=None, data=None):
  print "Refresh button clicked."
  self.browser.reload(gtkmozembed.FLAG_RELOADNORMAL)

def on_stop_clicked(self, widget=None, data=None):
  print "Stop Button Clicked."
  self.browser.stop_load()

def on_home_clicked(self, widget=None, data=None):
  print "Home Button clicked."
  print "Back button only works on actual pages and not render_data"
  self.browser.render_data(data, long(len(data)), 'file:///', 'text/html')

def on_go_clicked(self, widget=None, data=None):
  print "Go Button Clicked."
  self.browser.load_url(self.address.get_text())
```

These methods should be self explanatory. For example there is the on_back_clicked method which checks if the browser is able to go back to a previous page with:

```
self.browser.can_go_back()
```

If it passes this check, meaning there is a page to go back to, it goes back to the previous page with the following code:

```
self.browser.go_back()
```

As can be seen with each of the signaled methods, they are just like the on_back_clicked methods. They are very easy to use and not much else to that.

### 9.2.1 Running a PyGTK Mozembed Application

This section is very important if the gtkmozembed application is *seg faulting*. Depending on which operating system mozembed is being used on it may very well give a *segmentation fault*. This can be very frustrating to deal with if the reason is not known. However since I have run into this problem several times myself on a couple of systems I can give a few hints.

Basically the problem is that some mozilla libraries that are needed cannot be found by the application, so to run the app it needs to be started with a shell script instead of just running the python script.

#### 9.2.1.1 Ubuntu Gutsy

To use gtkmozembed on Ubuntu Gutsy, the path variables LD_LIBRARY_PATH and MOZILLA_FIVE_HOME must be set. So what should done is create a shell script. For example, create with a shell script with the name mozilla_embed_start.sh and put the following code inside:

gtkmozembed Run Script (Gutsy)

```
#!/bin/bash
export LD_LIBRARY_PATH=/usr/lib/firefox
export LD_MOZILLA_FIVE_HOME=/usr/lib/firefox
python your_gtkmozembed_application.py
```

Now run this code in the same directory as *your_gtkmozembed_ application.py* file and it should run with no segmentation fault.

#### 9.2.1.2 Ubuntu Feisty, Edgy, and Dapper

On Ubuntu Feisty, Edgy, and Dapper the path variable LD_LIBRARY_PATH must be set to the firefox library directory.

A shell script must also be created just like for Ubuntu Gutsy.

gtkmozembed Script (Feisty, Edgy, Dapper)

```
#!/bin/bash
export LD_LIBRARY_PATH=/usr/lib/firefox
python your_gtkmozembed_application.py
```

Now run this shell script in the same directory as your gtkmozembed application and it should now be running without any segmentation faults.

#### 9.2.1.3 Other Distributions

If you are running a different distribution of Linux or maybe a BSD the variables that need to be set may be different or the path to the firefox libraries may be different.

From here you are on your own. I suggest you try one of the Ubuntu solutions as one of them will probably work as long as you put in the correct firefox library path.

## 9.3 Internet Explorer

Using Internet Explorer with PyGTK is an interesting exercise that took me quite of bit of web searching before finding how to do this. Because this is not something that I need often I will be using a somewhat modified sample found on a mailing list[1] .

Just to be clear this is for Microsoft Windows only. In fact the only reason I am writing this here is so I do not forget myself. I once spent several hours creating a custom application for a specific purpose that had the ability to preview output html internally for ease of use. Then only to find out that gtkmozembed is not for windows. But I needed it to run on windows and linux. So here is how I got Internet Explorer to work on windows with PyGTK.

To start, the comtypes package will need to be installed and can be found at: http://sourceforge.net/projects/comtypes/. Also pywin32 should be installed.

Once this has been installed Internet explorer can be taken advantage of. First start off by importing the required modules and creating the required ctypes variables.

```
import win32con
from ctypes import *
from ctypes.wintypes import *
from comtypes import IUnknown
from comtypes.automation import IDispatch, VARIANT
from comtypes.client import wrap
kernel32 = windll.kernel32
user32 = windll.user32
atl = windll.atl
```

Of course PyGTK will need to be imported like any other GTK application.

```
import pygtk
pygtk.require(“2.0”)
import gtk
```

Now a class will be created call GUI with an __init__ method with the following code to setup the window. This will basically just be like using the gtkmozembed window.

```
self.home_url = "http://www.majorsilence.com/"

self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
self.win.set_title("Example Webbrowser that works on Linux and Windows") self.win.connect("dest
self.win.set_size_request(750, 550)
self.win.realize()
# Create a VBox to house the address bar and the IE control.
self.main_vbox = gtk.VBox()
```

[1] Mailing list found at at: http://www.mail-archive.com/comtypes-users@lists.sourceforge.net/msg00084.html and a direct link to the code found is: http://www.mail-archive.com/comtypes-users@lists.sourceforge.net/msg00084/ie_in_gtk.py

```
control_box = gtk.HBox(False, 0)

back = gtk.Button("Back")
forward = gtk.Button("Forward")
refresh = gtk.Button("Refresh")
stop = gtk.Button("Stop")
home = gtk.Button("Home")
self.address = gtk.Entry(max=0)
go = gtk.Button("Go")

control_box.pack_start(back, True, True, 2)
control_box.pack_start(forward, True, True, 2)
control_box.pack_start(refresh, True, True, 2)
control_box.pack_start(stop, True, True, 2)
control_box.pack_start(home, True, True, 2)
control_box.pack_start(self.address, True, True, 2)
control_box.pack_start(go, True, True, 2)

back.connect("clicked", self.on_backward_clicked, None)
forward.connect("clicked", self.on_forward_clicked, None)
refresh.connect("clicked", self.on_refresh_clicked, None)
stop.connect("clicked", self.on_stop_clicked, None)
home.connect("clicked", self.on_home_clicked, None)
self.address.connect("key_press_event", self.on_address_keypress)
go.connect("clicked", self.on_go_clicked, None)

self.main_vbox.pack_start(control_box, False, True, 2)

self.win.add(self.main_vbox)
self.win.show_all()

# Initialize all the Internet Explorer things
self.init_ie()
```

As can be seen with this example, the initialization function most creates a nice window to view web pages in. This is to be used with the rest of the code.

At the very end of the initialization is found the method call *self.init_ie()*, this is what sets up all the Internet Explorer stuff. I will be very honest here and say I am not sure how it all works since I do not really care to much about Windows programming, but I know that it does work.

So to take a look at the init_ie method what is found is the following:

```
def init_ie(self):
  # Create a DrawingArea to host IE and add it to the hbox.
  self.container = gtk.DrawingArea()
  self.main_vbox.add(self.container)
```

```
        self.container.show()
        # Make the container accept the focus and pass it to the control;
        # this makes the Tab key pass focus to IE correctly.
        self.container.set_property("can-focus", True)
        self.container.connect("focus", self.on_container_focus)
        # Resize the AtlAxWin window with its container.
        self.container.connect("size-allocate", self.on_container_size)
        # Create an instance of IE via AtlAxWin.
        atl.AtlAxWinInit()
        hInstance = kernel32.GetModuleHandleA(None)
        parentHwnd = self.container.window.handle
        self.atlAxWinHwnd = user32.CreateWindowExA(0, "AtlAxWin", self.home_url,
          win32con.WS_VISIBLE | win32con.WS_CHILD | win32con.WS_HSCROLL |
          win32con.WS_VSCROLL, 0, 0, 100, 100, parentHwnd, None, hInstance, 0)

        # Get the IWebBrowser2 interface for the IE control.
        pBrowserUnk = POINTER(IUnknown)()
        atl.AtlAxGetControl(self.atlAxWinHwnd, byref(pBrowserUnk))

        # the wrap call querys for the default interface
        self.browser = wrap(pBrowserUnk)
        # Create a Gtk window that refers to the native AtlAxWin window.
        self.gtkAtlAxWin = gtk.gdk.window_foreign_new(long(self.atlAxWinHwnd))
        # By default, clicking a GTK widget doesn't grab the focus away from
        # a native Win32 control.
        self.address.connect("button-press-event", self.on_widget_click)
```

All I can say about this is that it works. If you can figure it out good for you. Now lets focus on some of the methods that are needed to work with Internet Explorer that are connected to in the init_ie method.

Here is the on_widget_clicked method:

```
      def on_widget_click(self, widget, data):
        control self.win.window.focus()
```

This method is used with Internet Explorer because on Windows, by default a GTK application does not grab control from native win32 api.

Next is the on_container_size method.

```
      def on_container_size(self, widget, sizeAlloc):
        self.gtkAtlAxWin.move_resize(0, 0, sizeAlloc.width, sizeAlloc.height)
```

This is used to make sure the gtk.Drawing container is properly sized[2].

The last special method is on_container_focus.

[2] Well as far as I can tell.

```
def on_container_focus(self, widget, data):
  rect = RECT()
  user32.GetWindowRect(self.atlAxWinHwnd, byref(rect))
  ieHwnd = user32.WindowFromPoint(POINT(rect.left, rect.top))
  user32.SetFocus(ieHwnd)
```

Apparently this method is used to pass the focus to Internet Explorer by passing the handle of the Internet Explorer control.

And now are the backward, forward, stop, refresh, and home buttons.

```
def on_backward_clicked(self, widget=None, data=None):
  try:
    self.browser.GoBack()
  except:
    pass # No page to go back to

def on_forward_clicked(self, widget=None, data=None):
  try:
    self.browser.GoForward()
  except:
    pass

def on_refresh_clicked(self, widget=None, data=None):
  self.browser.Refresh()

def on_stop_clicked(self, widget=None, data=None):
  self.browser.Stop()

def on_home_clicked(self, widget=None, data=None):
  #self.browser.GoHome()
```

When using the GoBack and GoForward methods they must be used with error handling or they will will crash the program. The Refresh method is used to refresh, the Stop method is used to stop ad GoHome is used to go to the browsers home page.

The on_go_clicked method takes the address entered in the address bar and loads that page.

```
def on_go_clicked(self, widget=None, data=None):
  v = byref(VARIANT())
  self.browser.Navigate(self.address.get_text(), v, v, v, v)
```

Loading the page that is in the address uses the Navigate method, pass in the address, then a variant to the rest[3].

And finally the on_address_keypress method. This is the last method to be used with the Internet Explorer example. All this method does is watch for the Enter to be pressed and then calls the on_go_clicked method.

[3] I really have no idea what this does.

## 9.4 Mozilla and IE Example

This section will show an example of how to use Internet Explorer or Mozilla as the engine depending on the operating system that is being used.

Mozilla and Internet Explorer

```
"""
Embedding IE in pygtk via AtlAxWin and ctypes.
"""
# needs the comtypes package from http://sourceforge.net/projects/comtypes/
import sys
import pygtk
pygtk.require("2.0")
import gtk
if sys.platform=="win32":
    import win32con
    from ctypes import *
    from ctypes.wintypes
    import * from comtypes
    import IUnknown from comtypes.automation
    import IDispatch, VARIANT from comtypes.client
    import wrap
    kernel32 = windll.kernel32
    user32 = windll.user32
    atl = windll.atl
else:
    import gtkmozembed
class GUI:
    def __init__(self):
        self.home_url = "http://www.majorsilence.com/"

        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.set_title("Example Web browser that works on Linux and Windows")
        self.win.connect("destroy", gtk.main_quit) self.win.set_size_request(750, 550)
        self.win.realize()

        self.main_vbox = gtk.VBox()

        control_box = gtk.HBox(False, 0)
        back = gtk.Button("Back")
        forward = gtk.Button("Forward")
        refresh = gtk.Button("Refresh")
        stop = gtk.Button("Stop")
        home = gtk.Button("Home")
        self.address = gtk.Entry(max=0) # no limit on address length
        go = gtk.Button("Go")
        control_box.pack_start(back, True, True, 2)
        control_box.pack_start(forward, True, True, 2)
        control_box.pack_start(refresh, True, True, 2)
        control_box.pack_start(stop, True, True, 2)
        control_box.pack_start(home, True, True, 2)
        control_box.pack_start(self.address, True, True, 2)
        control_box.pack_start(go, True, True, 2)

        back.connect("clicked", self.on_backward_clicked, None)
        forward.connect("clicked", self.on_forward_clicked, None)
        refresh.connect("clicked", self.on_refresh_clicked, None)
        stop.connect("clicked", self.on_stop_clicked, None)
        home.connect("clicked", self.on_home_clicked, None)
        self.address.connect("key_press_event", self.on_address_keypress)
        go.connect("clicked", self.on_go_clicked, None)

        self.main_vbox.pack_start(control_box, False, True, 2)

        self.win.add(self.main_vbox)
        self.win.show_all()

        if sys.platform=="win32":
            self.init_ie()
        else:
            self.init_mozilla()

    def init_ie(self):
        # Create a DrawingArea to host IE and add it to the hbox.
        self.container = gtk.DrawingArea()
        self.main_vbox.add(self.container)
        self.container.show()

        # Make the container accept the focus and pass it to the control;
        # this makes the Tab key pass focus to IE correctly.
        self.container.set_property("can-focus", True)
        self.container.connect("focus", self.on_container_focus)

        # Resize the AtlAxWin window with its container.
        self.container.connect("size-allocate", self.on_container_size)

        # Create an instance of IE via AtlAxWin.
        atl.AtlAxWinInit()
        hInstance = kernel32.GetModuleHandleA(None)
        parentHwnd = self.container.window.handle
```

```
            self.atlAxWinHwnd = user32.CreateWindowExA(0, "AtlAxWin", self.home_url,
                win32con.WS_VISIBLE | win32con.WS_CHILD | win32con.WS_HSCROLL |
                win32con.WS_VSCROLL, 0, 0, 100, 100, parentHwnd, None, hInstance, 0)

            # Get the IWebBrowser2 interface for the IE control.
            pBrowserUnk = POINTER(IUnknown)()
            atl.AtlAxGetControl(self.atlAxWinHwnd, byref(pBrowserUnk))

            # the wrap call queries for the default interface
            self.browser = wrap(pBrowserUnk)

            # Create a Gtk window that refers to the native AtlAxWin window.
            self.gtkAtlAxWin = gtk.gdk.window_foreign_new(long(self.atlAxWinHwnd))

            # By default, clicking a GTK widget doesn't grab the focus away from
            # a native Win32 control.
            self.address.connect("button-press-event", self.on_widget_click)

        def init_mozilla(self):
            self.browser = gtkmozembed.MozEmbed()
            self.main_vbox.add(self.browser)
            self.browser.load_url(self.home_url)

        def on_backward_clicked(self, widget=None, data=None):
            if sys.platform=="win32":
                try:
                    self.browser.GoBack()
                except:
                    pass # No page to go back to
            else:
                if self.browser.can_go_back():
                    self.browser.go_back()

        def on_forward_clicked(self, widget=None, data=None):
            if sys.platform=="win32":
                try:
                    self.browser.GoForward()
                except:
                    pass
            else:
                if self.browser.can_go_forward():
                    self.browser.go_forward()

        def on_refresh_clicked(self, widget=None, data=None):
            if sys.platform=="win32":
```

```
            self.browser.Refresh()
        else:
            self.browser.reload(gtkmozembed.FLAG_RELOADNORMAL)

    def on_stop_clicked(self, widget=None, data=None):
        if sys.platform=="win32":
            self.browser.Stop()
        else:
            self.browser.stop_load()

    def on_home_clicked(self, widget=None, data=None):
        if sys.platform=="win32":
            # To go to Internet explorer's default home page use:
            #self.browser.GoHome()
            v = byref(VARIANT())
            self.browser.Navigate(self.home_url, v, v, v, v)
        else:
            self.browser.load_url(self.home_url)

    def on_go_clicked(self, widget=None, data=None):
        if sys.platform=="win32":
            v = byref(VARIANT())
            self.browser.Navigate(self.address.get_text(), v, v, v, v)
            #print dir(self.browser)
        else:
            self.browser.load_url(self.address.get_text())

    def on_address_keypress(self, widget, event):
        if gtk.gdk.keyval_name(event.keyval) == "Return":
            print "Key press: Return"
            self.on_go_clicked(None)

    def on_widget_click(self, widget, data):
        # used on win32 platform because by default a gtk application does
        # not grab control from native win32 control
        self.win.window.focus()

    def on_container_size(self, widget, sizeAlloc):
        self.gtkAtlAxWin.move_resize(0, 0, sizeAlloc.width, sizeAlloc.height)

    def on_container_focus(self, widget, data):
        # Used on win32 with Internet Explorer
        # Pass the focus to IE. First get the HWND of the IE control; this
        # is a bit of a hack but I couldn't make IWebBrowser2._get_HWND work.
        rect = RECT()
```

```
            user32.GetWindowRect(self.atlAxWinHwnd, byref(rect))
            ieHwnd = user32.WindowFromPoint(POINT(rect.left, rect.top))
            user32.SetFocus(ieHwnd)

    if "__name__" == "__main__":
        gui = GUI()
        gtk.main()
```

# Chapter 10

# Internationalization

Please send any fixes or suggestions to peter@majorsilence.com or leave a comment at http://www.majorsilence.com/pygtk_book.

Must install intltool package on linux systems to provide the tools and scripts that are needed to extract the needed information from the python scripts and the programs glade files.

- gettext.bindtextdomain(domain, localedir) - Bind the text to main to the locale directory that is specified. Where the binary .mo files are looked for.
- gettext.textdomain(domain) - Sets the current global domain to the domain argument. If domain is none then the current global domain is returned.
- gettext.translation(domain, localedir, languages, class, fallback, codeset) - Set the domain and the locale directory. All this chapter will be interested in is the first two arguments domain and localedir.
- gettext.install(domain, localedir, unicode, codeset, names) - Install the function _() in the python builtin namespace so that it may be used easily from any python module within a program.

## 10.1 Python/PyGTK Translation

To start off here is a very small program that has been setup for localization.

```
import pygtk, gtk
pygtk.require("2.0")
import locale, gettext

APP="translation-example"
DIR="po"

locale.setlocale(locale.LC_ALL, ”)
gettext.bindtextdomain(APP, DIR)
gettext.textdomain(APP)
lang = gettext.translation(APP, DIR)
_ = lang.gettext
gettext.install(APP, DIR)
```

To start off the variable APP is set to "translation-example" and is used to set the domain for the translation.

```
class TranslationExample(object):
  def __init__(self):
    self.label_1 = gtk.Label( _("Hello World!") )
    label_2 = gtk.Label( _("Still in the HBox") )
    button = gtk.Button( _("Click Me") )

    button.connect("clicked", self.on_button_clicked,
        _("Anything can go here") )
    vbox = gtk.VBox()
    vbox.pack_start(self.label_1) vbox.pack_start(label_2)
    vbox.pack_start(button)

    win = gtk.Window()
    win.connect("destroy", lambda wid: gtk.main_quit())
    win.add(vbox)
    win.show_all()

  def on_button_clicked(self, widget, data=None):
    self.label_1.set_text( _("Hello ") + str(data) )

if __name__ == "__main__":
  TranslationExample()
  gtk.main()
```

For more indepth coverage of gettext visit http://docs.python.org/library/gettext.html. To download the tools from windows get them from the gnu site ftp://ftp.gnu.org/gnu/gettext/gettext-tools-0.13.1.bin.woe32.zip.

Now use the gettext command tool to extract the needed strings from all the python files and create the translation-example.pot file.

```
gettext -language=Python -keyword=_ -keyword=N_
    -output=translation-example.pot translation-example.py
```

Now for each language that will be available for the application a .po file must be created. So if Canadian English is the language is to be used:

```
msginit -input=translation-example.pot -locale=en_CA
```

Will output a en_CA.po file. American English would be:

```
msginit -input=translation-example.pot -locale=en_US
```

Will output an en_US.po file. German would be:

```
msginit -input=translation-example.pot -locale=de_DE
```

This of course would output de.po.

Finally the .po files must be edited and the localized language put into their proper places. Just make sure that when the .po files are created that the *charset* is set to *utf-8*.

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2009-02-17 16:01-0330\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=utf-8\n"
"Content-Transfer-Encoding: 8bit\n"

#: translation-example.py:20
msgid "Hello "
msgstr ""

#: translation-example.py:23
msgid "Hello World!"
msgstr ""

#: translation-example.py:24
msgid "Still in the HBox"
msgstr ""

#: translation-example.py:25
msgid "Click Me"
msgstr ""

#: translation-example.py:29
msgid "Anything can go here"
msgstr ""
```

Now what you need to do is edit the .po files so that the empty msgstr have the translated text. So what this means is that:

```
#: translation-example.py:20
msgid "Hello "
msgstr ""
```

Would become in German:

```
#: translation-example.py:20
msgid "Hello "
msgstr "Guten Tag"
```

Once all the strings are translated then the .po file must be converted into a binary .mo file and placed in its proper folder. So the en_CA.po file would be converted into translation-example.mo and placed in the folder ./po/en_CA/LC_MESSAGES/.

```
msgfmt -output-file=translation-example.mo en_CA.po
```

Now copy the translation-example.mo file to the folder ./po/en_CA/LC_MESSAGES/. To test the the translated copy do the following:

```
LANG=lang python myapp.py
```

So to test translation-example.py with german that would become:

```
LANG=de_DE.UTF-8 python translation-example.py
```

It should be noted that on some systems that the .UTF-8 part is not needed.

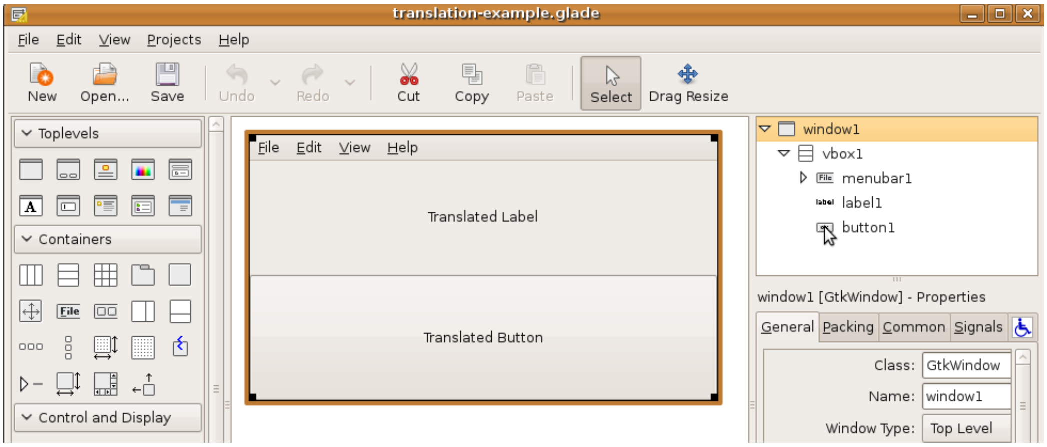
## 10.2 gtk.glade Translation

Translating a project that makes use of a glade file is easy. It just takes a few extra commands to extract the needed text strings. To start off here is an example program that makes use of the translation-example.glade file (See Figure 10.1).

```
import pygtk
pygtk.require("2.0")
import gtk, gtk.glade
import locale, gettext

APP="translation-example"
DIR="po-glade"

locale.setlocale(locale.LC_ALL, ”)
gettext.bindtextdomain(APP, DIR)
gettext.textdomain(APP)
```



Class: GtkWindow
Name: window1
Window Type: Top Level

Figure 10.1: Glade Translation Project

```
lang = gettext.translation(APP, DIR)
_ = lang.gettext
gettext.install(APP, DIR)

class TranslationExample(object):
  def on_button_clicked(self, widget, data=None):
    self.label_1.set_text( _("Hello ") + str(data) )

  def __init__(self):
    self.gladefile = gtk.glade.XML("translation-example.glade")
    gtk.glade.bindtextdomain(APP, DIR)
    self.gladefile.signal_autoconnect(self)

    self.main_window = self.gladefile.get_widget("window1")
    self.main_window.connect("delete_event", lambda wid, we: gtk.main_quit())
    self.main_window.show_all()

if __name__ == "__main__":
  TranslationExample()
  gtk.main()
```

Create a translation-example.glade.h file by running intltool-extract on translation-example.glade. This is needed to extract the strings to translate with the gettext command line tool.

```
intltool-extract -type=gettext/glade translation-example.glade
```

Now use the xgettext command tool to extract the needed strings from all the python files as well as the translation-example.glade.h header file that was created and create the translation-example.pot file.

```
xgettext -language=Python -keyword=_ -keyword=N_
    -output=translation-example.pot translation-example.py
    translation-example.glade.h
```

Now for each language that will be available for the application a .po file must be created. So if Canadian English is the language is to be used:

```
msginit -input=translation-example.pot -locale=en_CA
```

Will output a en_CA.po file. American English would be:

```
msginit -input=translation-example.pot -locale=en_US
```

Will output an en_US.po file. German would be:

```
msginit -input=translation-example.pot -locale=de_DE
```

This of course would put de.po.

Finally the .po files must be edited and the localized language put into their proper places. To do this please refer back to 10.1 on page 153 as it shows you how to use the *msgfmt* command and proper way to do the translations.

## 10.3 gtk.Builder Translation

Translating a project that makes use of a gtk.Builder file is easy. It just takes a few extra commands to extract the needed text strings. To start off here is an example program that makes use of the translation-example.glade file (See Figure 10.1). First this file must be translated to a gtk.Builder file using the gtk-builder-convert (See section 2.6 on page 52) script.

```
import pygtk
pygtk.require("2.0")
import gtk
import locale, gettext

APP="translation-example"
DIR="po-glade"

locale.setlocale(locale.LC_ALL, ”)
# This is needed to make gtk.Builder work by specifying the
# translations directory
locale.bindtextdomain(APP, DIR)

gettext.bindtextdomain(APP, DIR)
gettext.textdomain(APP)
lang = gettext.translation(APP, DIR)
_ = lang.gettext
```

```
gettext.install(APP, DIR)

class TranslationExample(object):
  def on_button_clicked(self, widget, data=None):
    self.label_1.set_text( _("Hello ") + str(data) )

  def __init__(self):
    self.gladefile = gtk.Builder()
    self.gladefile.set_translation_domain(APP)
    self.gladefile.add_from_file("translation-example.xml")
    self.gladefile.connect_signals(self)

    self.main_window = self.gladefile.get_object("window1")
    self.main_window.connect("delete_event", lambda wid, we: gtk.main_quit())
    self.main_window.show_all()

if __name__ == "__main__":
  TranslationExample()
  gtk.main()
```

Translating a gtk.Builder xml file uses the exact same commands as translating a glade file, however .glade is replaced with .xml for the file that is being used. So create a translation-example.xml.h file by running intltool-extract on translation-example.xml. This is needed to extract the strings to translate with the gettext command line tool.

```
intltool-extract -type=gettext/glade translation-example.xml
```

Now use the xgettext command tool to extract the needed strings from all the python files as well as the translation-example.glade.h header file that was created and create the translation-example.pot file.

```
xgettext -language=Python -keyword=_ -keyword=N_
    -output=translation-example.pot translation-example.py
    translation-example.xml.h
```

Now for each language that will be available for the application a .po file must be created. So if Canadian English is the language is to be used:

```
msginit -input=translation-example.pot -locale=en_CA
```

Will output a en_CA.po file. American English would be:

```
msginit -input=translation-example.pot -locale=en_US
```

Will output an en_US.po file. German would be:

```
msginit -input=translation-example.pot -locale=de_DE
```

This of course would put de.po.

Finally the .po files must be edited and the localized language put into their proper places. To do this please refer back to 10.1 on page 153 as it shows you how to use the *msgfmt* command and proper way to do the translations.

## 10.4 Testing Translations

To make sure that the translation is working properly it should be tested. This section will go into a bit more detail on setting this up.

First the language suppport files that the application has been translated into must be installed on the operating system. This section assumes ubuntu is the test system and the examples are geared toward it.

So lets assume the test system is ubuntu and German is the language that is to be tested. The easiest way to make sure that German language support is installed is to install the *language-support-de* package. This package will install all the german translation packages for the test system. If you wish you do not need to install this meta package, but can hunt down all the individual packages for german support.

Now make sure that the .mo files, in this case translation-example.mo, are copied to each of their respective language folders; Eg ./po/en_CA/LC_MESSAGES/. To test the the translated copy do the following:

```
LANG=lang python myapp.py
```

So to test translation-example.py with german that would become:

```
LANG=de_DE.UTF-8 python translation-example.py
```

It should be noted that on some systems that the .UTF-8 part is not needed.

### 10.4.1 Testing on Win32/Win64

From the command line:

```
SET Lang=de_DE
myapp.py
```

Another problem on Windows with gtkbuilder is that that it will not be translated in a pygtk application. You have to force it using ctypes[1]. At least at the time of writting (pygtk 2.16 with gtk 2.16 and 2.18)

After this line of code

```
gettext.install(APP,localedir=DIR)
```

You will then try something like this:

[1] For more information see https://bugzilla.gnome.org/show_bug.cgi?id=574520

```
try:
    libintl = ctypes.cdll.LoadLibrary("C:\\GTK\\gtk-2.16.6\\bin\\intl.dll")
    libintl.bindtextdomain(APP, DIR)
except:
    print "Error Loading translations into gtk.builder files"
```

## 10.5 Translation Cheatsheet

Small quick cheetsheet of the commands that are needed to translate.

```
intltool-extract -type=gettext/glade translation-example.glade
```

Extract from both glade/builder and python scripts

```
xgettext -language=Python -keyword=_ -keyword=N_
    -output=translation-example.pot translation-example.py
    translation-example.glade.h
```

Canadian English

```
msginit -input=translation-example.pot -locale=en_CA
```

American English

```
msginit -input=translation-example.pot -locale=en_US
```

German

```
msginit -input=translation-example.pot -locale=de_DE
```

Change charset in each .po file to "charset=UTF-8" and put in each translation string. Create binary .mo files for each .po file and place them in their proper ./po/LANG/LC_MESSAGES/ folder.

```
msgfmt -output-file=translation-example.mo en_CA.po
msgfmt -output-file=translation-example.mo en_US.po
msgfmt -output-file=translation-example.mo de_DE.po
```

Test each language the application using each language that it has been translated into.

```
LANG=en_CA.UTF-8 python translation-example.py
LANG=en_US.UTF-8 python translation-example.py
LANG=de_DE.UTF-8 python translation-example.py
```

## 10.6 Locale Lists

To be able to use and test any of these locale languages the language support packages for your linux distrubtion must be installed. On ubuntu these start with *language-support* and can be found using the synaptic package manager. So for german it would be *language-support-de*.

Here is a short list of locales[2] that can be translated to.

| | |
|---|---|
| **en_US** | English, United States of America |
| **en_CA** | English, Canada |
| **en_AU** | English, Australian |
| **en_GB** | English, Great Britain/United Kingdom |
| **es_MX** | Spanish, Mexico |
| **es_ES,** | Spanish, Spain |
| **de_DE** | Germany, German |
| **fr_FR** | French, France |
| **fr_CA** | French, Canadian |
| **it_IT** | Italian, Italy |
| **ru_RU** | Russian, Russia |
| **pt_BR** | Portuguese, Brazil |

## 10.7 Summary

For more information on this topic please see these sites:

- http://docs.python.org/library/gettext.html
- http://www.learningpython.com/2006/12/03/translating-your-pythonpygtk-application/
- http://faq.pygtk.org/index.py?req=show&file=faq22.002.htp

[2] On my ubuntu system there is a very nice list at /usr/share/i18n/SUPPORTED. This is a big list that does not include long form of the location of the locale.

# Chapter 11

# IronPython and Gtk-Sharp

Please send any fixes or suggestions to peter@majorsilence.com or leave a comment at http://www.majorsilence.com/pygtk_book.

## 11.1 Introduction

The purpose of this chapter is to introduce using Gtk with IronPython. It will include a few short examples covering:

- Layouts with Gtk.VBox and Gtk.HBox
- Gtk.Buttons
- Gtk.Entry
- Widget Events (Callbacks)
- Gtk.MessageDialog
- Gtk.Label
- Gtk.CheckButton
- Gtk.RadioButton
- Gtk.ComboBox
- Gtk.Statusbar
- Gtk.StatusIcon

## 11.2 Example 1

Example 1 shows the basics of using:

- Layouts with Gtk.VBox
- Gtk.Buttons
- Gtk.Entry
- Widget Events (Callbacks)
- Message Dialogs

To use Gtk Sharp from IronPython first you need to import the clr and add a reference to the gtk-sharp. Once this is finished you can import Gtk. The example below creates one window, adds a Gtk.Entry and Gtk.Button. The button has one event which is the self.HelloWorld function. The self.HelloWorld function displays a MessageDialog that will change the gtk.Entry default value to “Hello World!” if Yes is clicked. A Gtk.VBox is created and added to the window. This vbox is used to pack the self.textentry1 and button vertically. You can also use a Gtk.HBox instead or a combination of Gtk.VBox and Gtk.HBox.

Gtk.Application.Init() must be called before using Gtk and Gtk.Application.Run() starts the Gtk main event loop. The window has the DeleteEvent attached to call the self.DeleteEvent function. The self.DeleteEvent function alls Gtk.Application.Quite() which exits the application.

```
import clr
clr.AddReference('gtk-sharp')
import Gtk

class GtkExample(object):
    def __init__(self):
        Gtk.Application.Init()
        self.window = Gtk.Window("Hello World")
        self.window.DeleteEvent += self.DeleteEvent

        vbox = Gtk.VBox()

        button = Gtk.Button("Show Message")
        button.Clicked += self.HelloWorld

        self.textentry1 = Gtk.Entry("Default Text")
        vbox.PackStart(self.textentry1)

        vbox.PackStart(button)

self.window.Add(vbox)
        self.window.ShowAll()
        Gtk.Application.Run()
        def DeleteEvent(self, widget, event):
            Gtk.Application.Quit()

        def HelloWorld(self, widget, event):
            m = Gtk.MessageDialog(None, Gtk.DialogFlags.Modal, Gtk.MessageType.Info, \
                Gtk.ButtonsType.YesNo, False, 'Change the text entry to "Hello World?"')

            result = m.Run()
            m.Destroy()

            if result == int(Gtk.ResponseType.Yes):
                self.textentry1.Text = "Hello World!"

    if __name__ == "__main__":
        GtkExample()
```

## 11.3 Summary

At this point you should be able to create a basic Gtk application using IronPython and be able to extrapolate based on the c# gtk documation how to use more features from within IronPython.

# Appendix A

# Book Text Licenses

See http://creativecommons.org/licenses/by-sa/3.0/legalcode

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

1. "Adaptation" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License. 2. "Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined below) for the purposes of this License. 3. "Creative Commons Compatible License" means a license that is listed at http://creativecommons.org/compatiblelicenses that has been

approved by Creative Commons as being essentially equivalent to this License, including, at a minimum, because that license: (i) contains terms that have the same purpose, meaning and effect as the License Elements of this License; and, (ii) explicitly permits the relicensing of adaptations of works made available under that license under this License or a Creative Commons jurisdiction license with the same License Elements as this License. 4. "Distribute" means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership. 5. "License Elements" means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, ShareAlike. 6. "Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License. 7. "Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast. 8. "Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work. 9. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation. 10. "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images. 11. "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

1. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections; 2. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified."; 3. to Distribute and Publicly Perform the Work including as incorporated in Collections; and, 4. to Distribute and Publicly Perform Adaptations. 5.

For the avoidance of doubt: 1. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; 2. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and, 3. Voluntary License Schemes. The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

1. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(c), as requested. 2. You may Distribute or Publicly Perform an Adaptation only under the terms of: (i) this License; (ii) a later version of this License with

the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-ShareAlike 3.0 US)); (iv) a Creative Commons Compatible License. If you license the Adaptation under one of the licenses mentioned in (iv), you must comply with the terms of that license. If you license the Adaptation under the terms of any of the licenses mentioned in (i), (ii) or (iii) (the "Applicable License"), you must comply with the terms of the Applicable License generally and the following provisions: (I) You must include a copy of, or the URI for, the Applicable License with every copy of each Adaptation You Distribute or Publicly Perform; (II) You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License; (III) You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform; (IV) when You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License. 3. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv) , consistent with Ssection 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties. 4. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

1. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License. 2. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

1. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License. 2. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License. 3. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable. 4. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent. 5. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements

# Appendix B

# Source Code Lisence

when the facility is invoked), then you may convey a copy of the modified version:

a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

# Appendix C

# PyGTK and Windows

## C.1 Install GTK and Glade

First thing you need before you start using PyGTK on Windows is to have GTK+ installed.

Go to the gtk web site (`http://www.gtk.org`) download section and download the GTK bundle for windows. Then you will need to set your path to include the location of gtk.

- Control Panel -> System -> Advanced tab -> Environment Variables -> Select "Path"

Now for the path select it for your current user or your system wide path and add the "bin" location from where you put your GTK bundle.

## C.2 Install PyGTK

Go to the PyGTK website (`http://www.pygtk.org`) download page. At the top of the page are the downloads for Windows. You will need all three and need to install them in this order:

- PyGObject
- PyCairo
- PyGTK

You may also want to download the GTK+ Preference Tool. You should be able to find it at http://sourceforge.net/projects/gtk-win/files/. This tool will allow you to set the GTK theme on your Windows user account. At this point you should have a PyGTK development environment on your computer.

If you want you can also install Win32 extensions for python from http://sourceforge.net/projects/pywin32/files/.

## C.3 Icons Not Displaying

Some win32 distributions of GTK+ have icons set to not display. This is a configuration option in the gtkrc theme files. One option to fix this and make sure that icons display for buttons and all other widgets is to place the following code in the main PyGTK file of your application.

```
if sys.platform=="win32":
 gtk.settings_get_default().set_long_property("gtk-button-images", True, "main")
```

The other option is to do the following:

1. Open the C:\GTK\share\themes\MS-Windows\gtk-2.0\gtkrc file

2. And change the line “gtk-button-images = 0” to “gtk-button-images = 1”

# Appendix D

# Stock Icons

gtk.STOCK_ABOUT Available in GTK+ 2.6 and above.

gtk.STOCK_ADD

gtk.STOCK_APPLY

gtk.STOCK_BOLD

gtk.STOCK_CANCEL

gtk.STOCK_CDROM

gtk.STOCK_CLEAR

gtk.STOCK_CLOSE

gtk.STOCK_COLOR_PICKER Available in GTK+ 2.2 and above.

gtk.STOCK_CONVERT

gtk.STOCK_CONNECT Available in GTK+ 2.6 and above.

gtk.STOCK_COPY

gtk.STOCK_CUT

gtk.STOCK_DELETE

gtk.STOCK_DIALOG_AUTHENTICATION Available in GTK+ 2.4 and above.

gtk.STOCK_DIALOG_ERROR

gtk.STOCK_DIALOG_INFO

gtk.STOCK_DIALOG_QUESTION

gtk.STOCK_DIALOG_WARNING

gtk.STOCK_DIRECTORY Available in GTK+ 2.6 and above.

gtk.STOCK_DISCONNECT Available in GTK+ 2.6 and above.

gtk.STOCK_DND

gtk.STOCK_DND_MULTIPLE

gtk.STOCK_EDIT Available in GTK+ 2.6 and above.

gtk.STOCK_EXECUTE

gtk.STOCK_FILE Available in GTK+ 2.6 and above.

gtk.STOCK_FIND

gtk.STOCK_FIND_AND_REPLACE

gtk.STOCK_FLOPPY

gtk.STOCK_FULLSCREEN Available in GTK+ 2.8 and above.

gtk.STOCK_GOTO_BOTTOM

gtk.STOCK_GOTO_FIRST

gtk.STOCK_GOTO_LAST

gtk.STOCK_GOTO_TOP

gtk.STOCK_GO_BACK

gtk.STOCK_GO_DOWN

gtk.STOCK_GO_FORWARD

gtk.STOCK_GO_UP

gtk.STOCK_HARDDISK Available in GTK+ 2.4 and above

gtk.STOCK_HELP

gtk.STOCK_HOME

gtk.STOCK_INDENT Available in GTK+ 2.4 and above.

gtk.STOCK_INDEX

gtk.STOCK_INFO Available in GTK+ 2.8 and above.

gtk.STOCK_ITALIC

gtk.STOCK_JUMP_TO

gtk.STOCK_JUSTIFY_CENTER

gtk.STOCK_JUSTIFY_FILL

gtk.STOCK_JUSTIFY_LEFT

gtk.STOCK_JUSTIFY_RIGHT

gtk.STOCK_LEAVE_FULLSCREEN Available in GTK+ 2.8 and above.

gtk.STOCK_MEDIA_FORWARD Available in GTK+ 2.6 and above.

gtk.STOCK_MEDIA_NEXT Available in GTK+ 2.6 and above.

gtk.STOCK_MEDIA_PAUSE Available in GTK+ 2.6 and above.

gtk.STOCK_MEDIA_PLAY RTL version is Available in GTK+ 2.6 and above.

gtk.STOCK_MEDIA_PREVIOUS Available in GTK+ 2.6 and above.

gtk.STOCK_MEDIA_RECORD Available in GTK+ 2.6 and above.

gtk.STOCK_MEDIA_REWIND Available in GTK+ 2.6 and above.

gtk.STOCK_MEDIA_STOP Available in GTK+ 2.6 and above.

gtk.STOCK_MISSING_IMAGE

gtk.STOCK_NETWORK Available in GTK+ 2.4 and above.

gtk.STOCK_NEW

gtk.STOCK_NO

gtk.STOCK_OK

gtk.STOCK_OPEN

gtk.STOCK_PASTE

gtk.STOCK_PREFERENCES

gtk.STOCK_PRINT

gtk.STOCK_PRINT_PREVIEW

gtk.STOCK_PROPERTIES

gtk.STOCK_QUIT

gtk.STOCK_REDO

gtk.STOCK_REFRESH

gtk.STOCK_REMOVE

gtk.STOCK_REVERT_TO_SAVED

gtk.STOCK_SAVE

gtk.STOCK_SAVE_AS

gtk.STOCK_SELECT_COLOR

gtk.STOCK_SELECT_FONT

gtk.STOCK_SORT_ASCENDING

gtk.STOCK_SORT_DESCENDING

gtk.STOCK_SPELL_CHECK

gtk.STOCK_STOP

gtk.STOCK_STRIKETHROUGH

gtk.STOCK_UNDELETE

gtk.STOCK_UNDERLINE

gtk.STOCK_UNDO

gtk.STOCK_UNINDENT Available in GTK+ 2.4 and above.

gtk.STOCK_YES

gtk.STOCK_ZOOM_100

gtk.STOCK_ZOOM_FIT

gtk.STOCK_ZOOM_IN

gtk.STOCK_ZOOM_OUT

# Bibliography

[1] 2006-03-02, John Finlay, Version 2.5, PyGTK 2.0 Tutorial, http://pygtk.org/pygtk2tutorial/

[2] 2007-07-31, http://www.cairographics.org/pycairo/tutorial/

[3] Accessed 2008-08-31,Michael Urman, http://www.tortall.net/mu/wiki/CairoTutorial

[4] Accessed 2008-08-31,Michael Urman,http://www.tortall.net/mu/wiki/PyGTKCairoTutorial

[5] Accessed 2008-08-31, Lawrence Oluyede,http://pygtk.org/articles/cairo-pygtk-widgets/cairo-pygtk-widgets.htm

[6] http://pygtk.org/articles/cairo-pygtk-widgets/cairo-pygtk-widgets2.htm

[7] http://www.mono-project.com/FAQ:_Gtk#Should_I_hook_up_to_events.2C_or_override_methods_to_create_a_custom_widget.3F

[8] 2008-04-16, http://blog.sontek.net/2008/04/16/printing-in-gtk/

[9] 2008-04-15, John Anderson, PyGTK 2.0 Reference Manual http://pygtk.org/docs/pygtk/

[10] Accessed 2008-08-31, Cairo 1.6.5 API Reference Manualhttp://www.cairographics.org/manual/cairo-Image-Surfaces.html

[11] http://standards.freedesktop.org/desktop-entry-spec/latest/ar01s05.html

[12] http://standards.freedesktop.org/menu-spec/menu-spec-1.0.html

[13] http://pygstdocs.berlios.de/pygst-tutorial/seeking.html

[14] http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer/html/GstElement.html

[15] 2008-01-23, Thomas Heller, Internet Explorer with PyGTK, http://www.mail-archive.com/comtypes-users@lists.sourceforge.net/msg00084.html

[16] 2007-09-23, Giuliani Vito Ivan, http://zeta-puppis.com/2007/09/23/an-introduction-to-pyclutter-part-one/

[17] Accessed 2009-02-18, gettext - Multilingual internationalization services, http://docs.python.org/library/gettext.html

[18] Accessed 2009-02-18, learningpython.com,http://www.learningpython.com/2006/12/03/translating-your-pythonpygtk-application/

# Index