# Introduction to GUI programming with PyGTK and Glade

This tutorial will give you a brief introduction into create a GUI application for Python using PyGTK and Glade, include how to install PyGTK and Glade.

This tutorial uses a 32 bit installation of Python 2.7 I am also assuming you are installing everything in default locations.

This tutorial is an adaptation of Latty’s Python GUI tutorial which can be found here:

<http://www.overclock.net/t/342279/tutorial-using-python-glade-to-create-a-simple-gui-application>

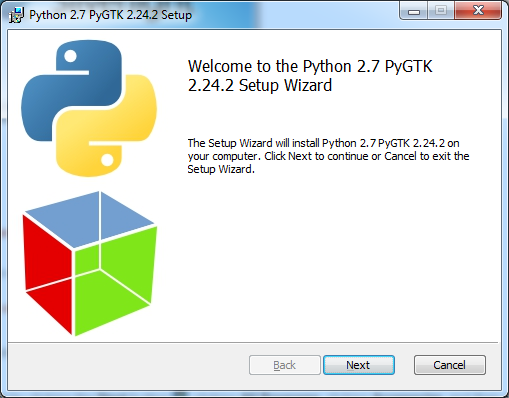
## Installing PyGTK and Glade

PyGTK has an all in one installer that will install the Python implementation of the GTK+ toolkit as well as Glade interface designer. A link to the download page is provided below. For this tutorial we are installing version 2.24.2 of PyGTK.

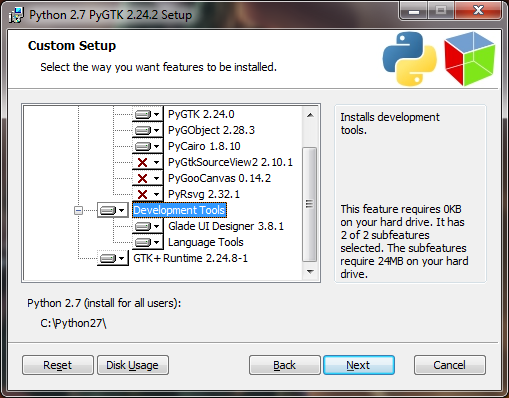
<http://www.pygtk.org/downloads.html>



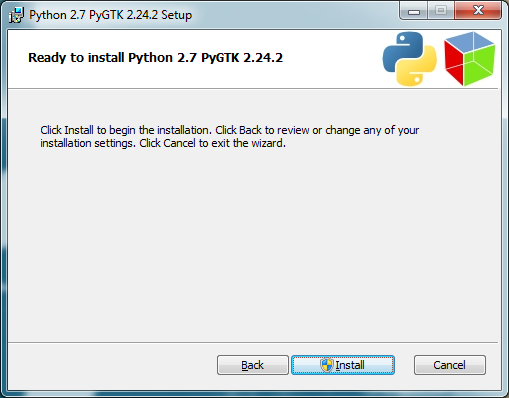
Download and run the installer.



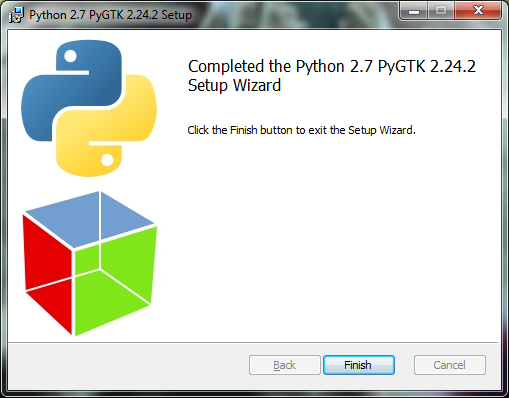
Click Next.



On the custom setup window make sure you have selected to install the **Development Tools**, make sure it has also selected the right folder for your Python installation. Click Next.



Click Install.

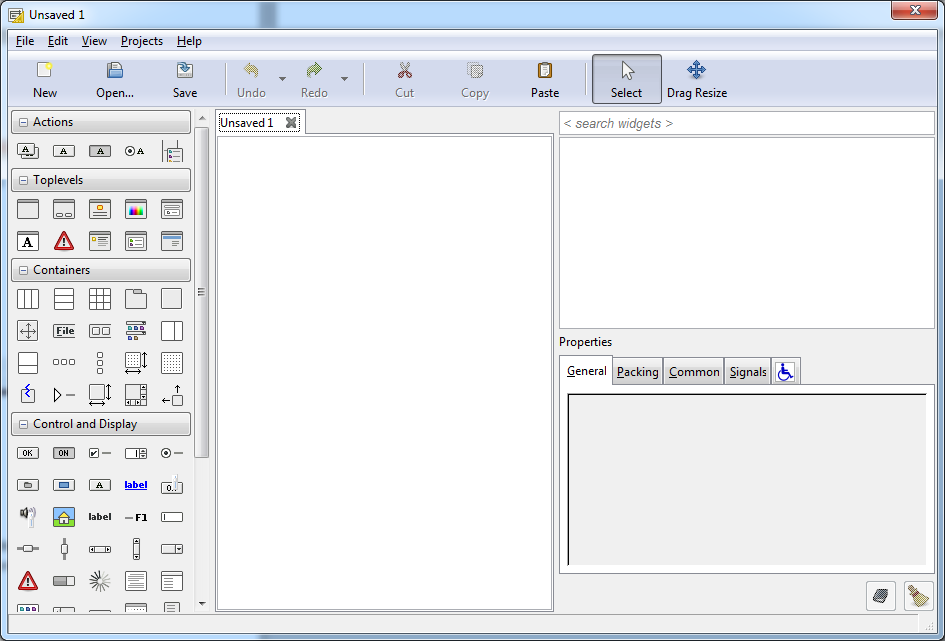


Click Finish.

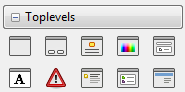
## Quick Sanity Check

The first thing we will be doing is creating a one button application in Glade. We are doing this for two reasons, one to make sure everything has installed correctly and secondly to give you understanding of how to link your code to the GUI.

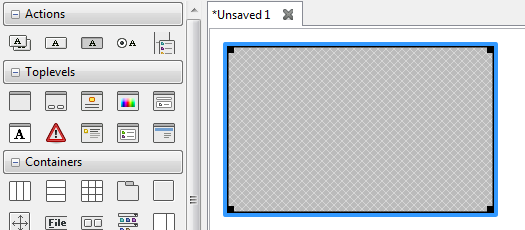
Open up **Glade UI Designer**. You should see a screen like this.



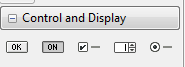
The first thing we need to do is create a **blank window** that will hold our GUI. The blank window is found in the **Toplevels** group in the **pallet box**. Clicking on Window will create a blank window in the main view.



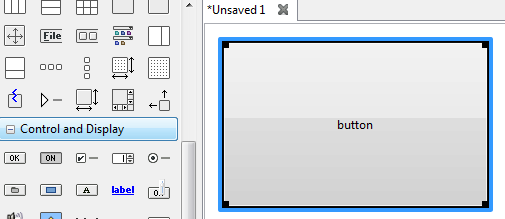
Window



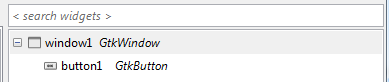
Next we want to add a **button**. Buttons can be found in the **Control and Display** group in the pallet box. Click on the button then click on the blank window in the main view.



Button



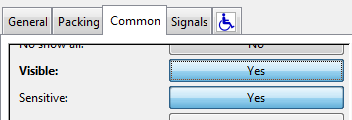
Our GUI is complete, we all need to do now is change some properties before we start coding. If you are having trouble selecting between the window and the button you can use the inspector in the top right to select different elements of the GUI.



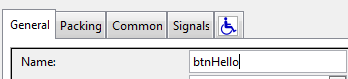
For the window we want to change the name of the window to mainWindow. This is found under the General tab of the properties box.



We also want to set mainWindow visibility to yes. This is found under the Common tab of the properties box.

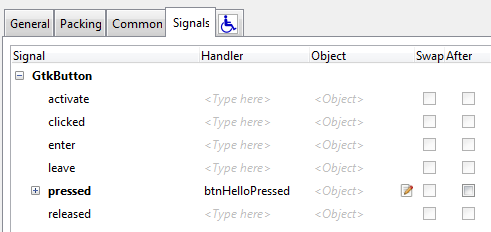


For the button window we want to change the name to btnHello

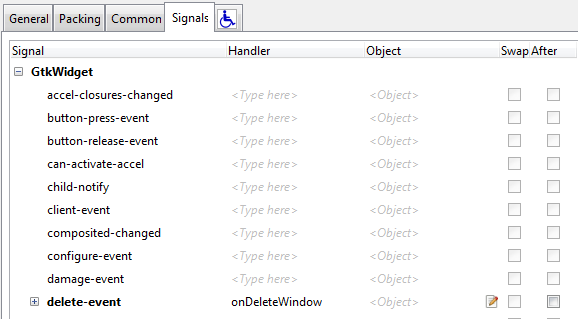


We also want to change the Signals for the btnHello and mainWindow. Signals are actions that our GUI listen out for, code will be executed when the signal is heard.

For btnHello we are going to create a pressed signal. The handler will be called **btnHelloPressed**



For mainWindow we are going to create a signal for when the window is deleted. This event is called delete-event. Give it a handler of **onDeleteWindow**.



We are done with the interface save the interface and call it **test.glade**

Once you have saved the interface open up a Python script editor of your choice. I use IDLE. Save your Python script in the same folder as test.glade.

The first thing we are going to do is load the GUI. The first line of code should be:



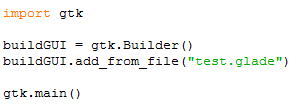
This will give us access to all code need to get our GUI to work. Next we need to build our GUI from the file that Glade has created, gtk has provided us with an object that can do this called Builder. The next two lines we are going to create a builder object and then have it create our GUI from test.glade.



Next we want to tell the GUI to run. This can be done using the following line of code:



This is what your code should now look like:

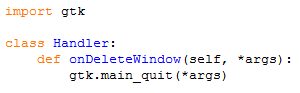


Save the code and run it. Your GUI should appear somewhere on the screen.



Just a word of note, at the moment our program doesn’t know how to quite. If you click on the X button the window will disappear but the program will keep on running. Depending on your code this may cause some interesting memory leaks. Let’s fix this problem.

We are now going to create some events to happen when someone triggers a signal. We are going to define a class called handler which will have functions for all the signals we want. The first function will be the window delete signal so we can exit the program.



Two things to note, one I have put it under the import statements because I like to define my events before I build the GUI. Also the name of the function is the EXACT NAME of the handler we set in the mainWindow properties.

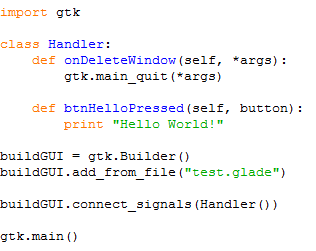
Now let’s make the even for the button pressed. In this example we are going to print “Hello World!” to the console. I know how original.



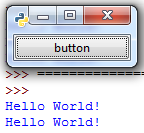
Now that we have created our events lets connect them to the signals. We can do that with the following line of code:



The final code should look like this.



Save and run the code.



It should be working.

## Adding program

This part of the tutorial assumes you have down the sanity check.

We are now going to create a small program that adds two numbers together. First open up **Glade UI Designer** and create a blank window.

You may have noticed form the sanity check that if we drag a widget into the window it takes up the entire window. We don’t want that to happen, to get around this problem we are going to create **containers**. Containers are used to hold widgets and allow you to influence where widgets will appear on your interface. There are many different types of containers and you will have to do some experimenting to get the lay out you want.

The first thing we are going to create a vertical box to separate our window into three sections. Click on the vertical box tool and then click on your window. We want to hold three items.



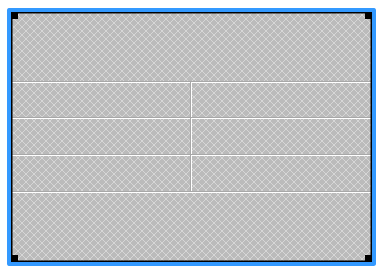
Vertical box

We want to create a table container in the middle vertical box. We want it to have 2 columns and three rows.

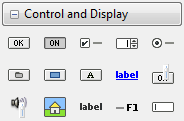


Table

Your window should now look like this.

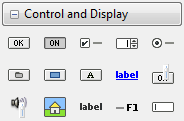


In the top vertical box and in the first column of the middle vertical box we want to add labels.



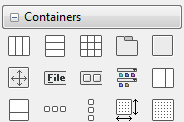
Label

In all three rows of the second column of the middle vertical box we want to add a text entry.



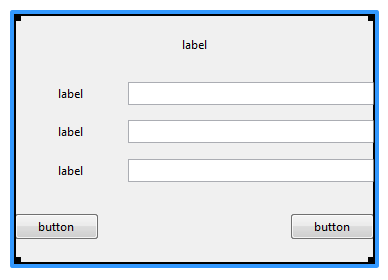
Text Entry

In the bottom vertical container we want to add a Horizontal Button Box to hold two buttons.

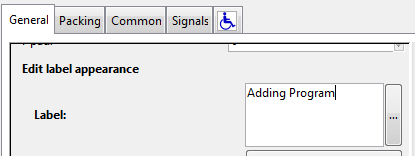


Horizontal Button Box

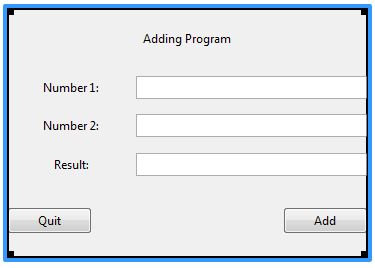
We then want to add two buttons to the horizontal button box. Your interface should now look like this.



To change the text a label or button displays change the label property for that label or button.



Your final interface should look like this.



Now all we have to do is change the properties for some of the widgets as well as create some signals.

For the text entry next to label Number 1 and label Number 2:

|  |  |
| --- | --- |
| *Property* | *Value* |
| Name | *txtNumber1* |

|  |  |
| --- | --- |
| *Property* | *Value* |
| Name | *txtNumber2* |

For the text entry to label Result

|  |  |
| --- | --- |
| *Property* | *Value* |
| Name | *txtResult* |
| Sensitivity | *No* |

Sensitivity changes the text entry to read only, it can be found under the common tab.

For the two buttons:

|  |  |
| --- | --- |
| *Property* | *Value* |
| Name | *btnQuit* |

|  |  |
| --- | --- |
| *Property* | *Value* |
| Name | *btnAdd* |

Change the name of the window to mainWindow and set visibility to yes.

For mainWindow we are going to create a signal for when the window is deleted. This event is called delete-event. Give it a handler of **onDeleteWindow**.

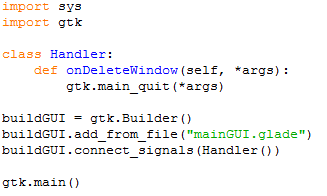
For btnQuit we are going to create a pressed signal. The handler will be called **btnQuitPressed**

For btnAdd we are going to create a pressed signal. The handler will be called **btnAddPressed**

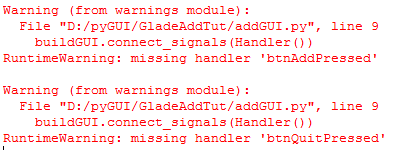
Save your interface.

Once you have saved the interface open up a Python script editor of your choice.. Save your Python script in the same folder as the .glade file.

The first thing we are going to do is create an even to quit the program and load the GUI just to make sure everything works.



When you run the program you will get warning messages about missing handlers, you can ignore these as we just making sure the interface loads.

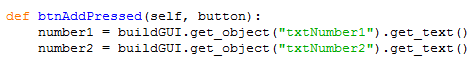


Once we have checked to see it works we can now make our interface do something.

The function btnQuitPressed will be simple as we are just exiting the program.



The function to add the numbers is a little more complicated. The first thing we need to do is get the numbers from the text entry. I will show you the code then give you an explanation of what is happening.



What’s happen in line 2 is we are first declaring a variable called number1, we are assigning number1 the value of the text in txtNumber1. This is a twostep process, the *buildGUI.get\_oibject* part of the statement allows us to get access to any widgets in our GUI, it uses the widgets name to find the match. The next part *.get\_text()* allows us to get the text of a text entry.

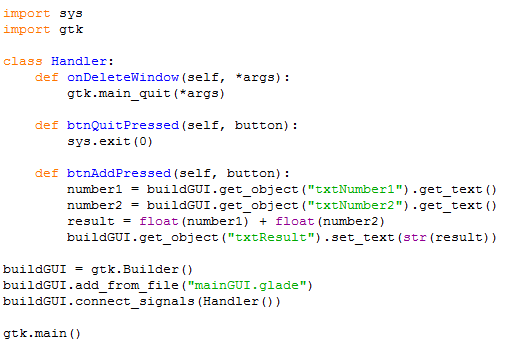
Next we are going to add the numbers together and store them in a variable called result. As the get\_text() function returns a text data type we will need to cast the numbers to float.



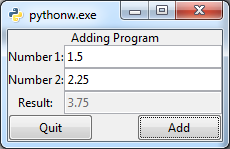
Lastly we now need to update the value of txtResult. Make sure we cast our number back to a string.



This is what the final script should look like.



Make sure you test and run your program.

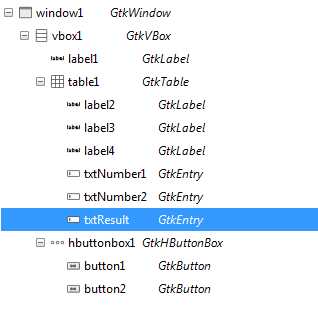


As an extension add three more buttons to subtract, multiple and divide. Make sure you add a guard so the user can’t divide by zero.

Just be aware, with the example I have shown you the code for the program is tightly coupled with the code of the GUI. This can be seen as bad design as I if I change the code of the program I will have to change the code of the GUI and if I change the GUI I have to change the code of the program. The original tutorial I linked shows a better example with the GUI code isolated form the program code.

## Further notes

A nice feature about Glade is that the inspector window gives you the name of the objects as referenced by the GTK+ reference guide. This is help because if we want to know what widgets can do we can just look at the reference guide to find out. I have provided links to the PyGTK reference guide as well as GTK+ you should always look at the Python implementation first.



PyGTK reference guide:

<https://developer.gnome.org/pygtk/stable/>

GTK+ reference guide:

<https://developer.gnome.org/gtk3/stable/>