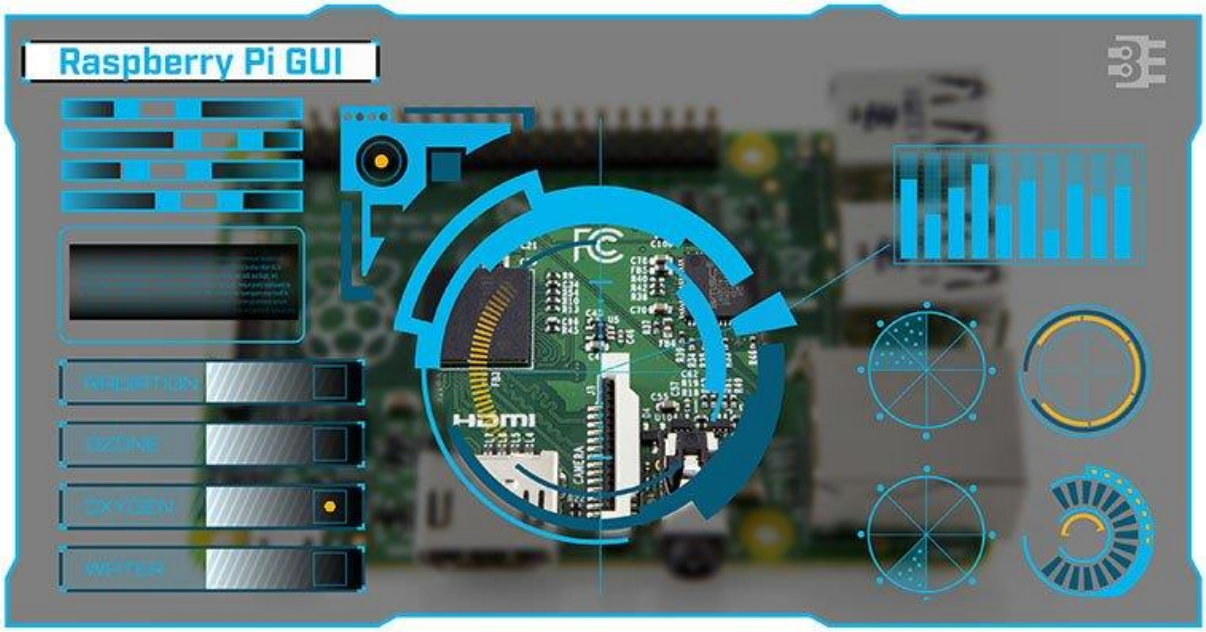
Raspberry Pi GUI Tutorial

Use Qt and Python to make an easy to use Pi App

by James Lewis https://www.baldengineer.com/raspberry-pi-gui-tutorial.html



My favorite Raspberry Pi add-on is the PiTFT from Adafruit. With it, you easily get a Raspberry Pi GUI interface and touch screen. The PiTFT software install is just a few things and it is good to go.

Image from adafruit.com

This screen is what I needed in my IoT project. The Pi+Screen will act as the primary controller for all of my things. The problem is I didn't know much about writing GUI applications in Linux. So what could I do to create a Raspberry Pi GUI?

Python is popular in Pi projects, so I decided to stick with it and find out what GUI toolkits are ready to go. "Ready to go" means they install easily on Raspian and work well on the Pi.

Here is how I got Qt5 for Python up and running to create a Raspberry Pi GUI.

Raspberry Pi GUI Libraries

If you start with this article on dice.com, you'll find five potential python toolkits.

Not listed there is TkInter, which is the most used Python toolkit. If you have Python running on your system, you have TkInter. I decided not to use it though because the widgets have a very dated look to them.

The other toolkit I considered was wxPython. The UI widgets have a modern feel, though they not entirely native looking. Also plenty of documentation exists. However, I had some trouble getting wxGlade running, which is a WYSIWYG GUI editor.

This lead me to Qt.

Why Qt for Raspberry Pi GUI

In college, I used Qt for my senior design project: an electronic notebook. It was a PC with a 10" touch screen. You could write on it, and it'd save what you write, about ten years before Apple did it.

Having a little bit of familiarity with Qt, I looked into its Python bindings and was happy. I decided on the PyQt binding from Riverbank to go along with Qt5.

Note about licensing PyQt and Qt

PyQt is available as GPL or Commercial License while Qt is available under LGPL or Commercial license. I know even the mere mention of "commercial" will cause some people to turn away. And that's fine. Qt is modern, and it's free (since I'm not developing anything commercial.)

If you want "pure" open source, then you might be happier with wxPython.

Development and Production

While the Raspberry Pi is a perfectly capable computer, I found developing on a 3.2" TFT a bit cumbersome. So I wanted to use my laptop "for development" and then deploy the code to my Pi when done.

In my case, I found it pretty easy to duplicate the Python/Qt environment on both.

Installing Python3

It wasn't clear to me if Qt5 worked with the python2.7. I ran into problems and decided to use python3 instead.

Mac (aka PC) Instructions

Whether you're on a Mac, Windows, or Linux, get python3 and Qt5 installed. I don't have much to say about Windows or Linux but on the Mac, install python3 and Qt5 with homebrew.

Simple command:

```
1brew install python3
2brew install qt5
```

If you know all the directories, you can probably get away without re-installing these. However, I like that with the homebrew install; they can find each other.

Next you need PyQt5, which installation varies by operation system.

On Mac, use homebrew:

```
1brew install pyqt5
```

On Linux, check your package manager. The PyQt5 download page has binaries for Windows.

Raspberry Pi Instructions

Assuming you're running Raspian Jessie, you can install all three with apt-get:

```
1sudo apt-get install python3 python3-pyqt5
```

I'm no apt-get expert, but "python3-pyqt5" might be enough to drag along python3.

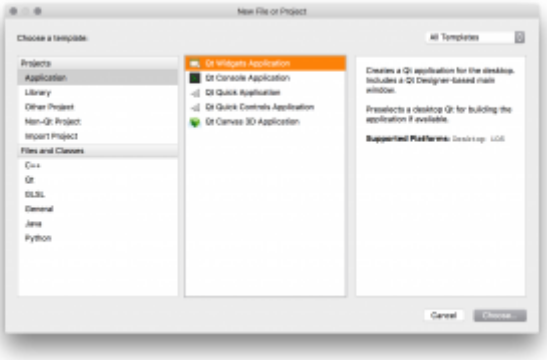
Download QtCreator

On the PC you plan to develop from, download and install QtCreator. You can download that from Qt, after answering some questions about how you plan to use it.

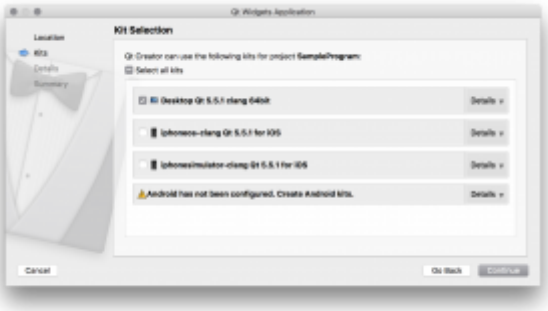
Believe it or not, I think getting python3, Qt5, and PyQt5 installed is the hardest part. After that, it's just a matter of copy/paste to get a GUI up and running.

Making a GUI in QtCreator

Start a new project in QtCreator and select "Desktop Application."

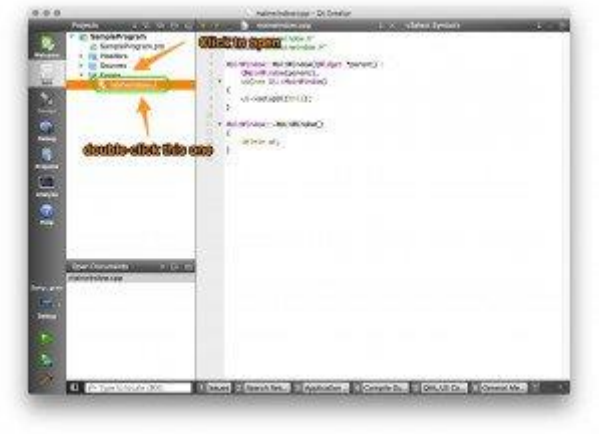


Where you save it, is up to you. (*Note, OSX when selecting a directory, the "New Project" window gets hidden. It's behind the initial QtCreator window.*) On kit selection, I have been sticking with "Desktop Qt 5.5.1 clang 64bit." No need to select the others. QtCreator is going to try and create a C++ program, but the only thing we are interested in is the MainWindow.ui file. So double-click that to open the GUI Editor.



Last you need to name the class for your program. For this tutorial, leave it as "MainWindow." That way the name will match up with the Python stuff later.

Finally, select none for version control and QtCreator going to create a C++ framework.



The only thing we are interested in is the MainWindow.ui file. So double-click that to open the GUI Editor. At this point, you're going to add the GUI elements to create whatever GUI you want. I'm not going to go into depth about the different widgets or how they work. Plenty of better tutorials on those.

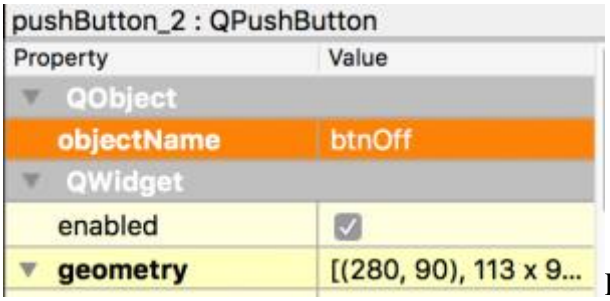
Instead, here are the critical things to know when creating your Raspberry Pi GUI.

1. Set the Window to your Screen Size
2. Pay attention to the widget name
3. Set defaults for elements

1. Window Size / Screen Size

In my case, I am using the 3.5" PiTFT. Using screen means I have 460 by 320 pixels available. This resolution is taking into account LXDE's menubar. So I made sure to set my Window to that size.

2. Widget Name



Here's where my hardware background shows. I'm not sure what the right naming convention to use for widget elements. However, 25 years ago when I was forced to learn VisualBasic, I learned the "xxxNamey" convention. In this case, "xxx" is a three letter abbreviation for the widget style and "y" is the numeric element. So for a button it might be "btnOK1" or "btnRedOn1" and a Text Box would be "txtName1".

If you know a better method, let me know. I encourage this one because it makes the code read easier later on.

Don't stick to the default widget names QtCreator provides.

In this code I created two buttons "On" and "Off" named "btnOn" and "btnOff" respectively.

3. Set Defaults

Even if you plan for your program/script to fill in values in the GUI, make sure you set appropriate defaults. For example "PushButton" is a label text for the button. You CAN change in your code, but why not set it now?

Create the GUI Code

Download Tutorial Code

Once you have drawn your GUI elements, it is time to generate the matching Python code. Now, don't worry if your GUI isn't "done." You can repeat the following steps as many times as you want.

In QtCreator, just save your GUI. It will update the MainWindow.ui file. In a terminal (or command shell for Windows), we're going to make use of the pyuic5 utility. On Mac and Unix, here's your command:

```
1pyuic5 mainwindow.ui > mainwindow_auto.py
```

Name the result whatever you want, I keep the name mostly the same. Each time you update the GUI in QtCreator, you'll want to run pyuic5 to update the "auto.py" file.

Okay, you're asking, are we done yet? Do we have a GUI? Yes and no. This code will run and create a GUI, but that's not quite enough yet.

Add Basic Python Code

Create a new Python file in the same directory as your mainwindow_auto.py, with this initialization code. Not that it matters really, but I name my main python a very clever name: “main.py”. I know, right?

```
1 # always seem to need this
2 import sys
3
4 # This gets the Qt stuff
5 import PyQt5
6 from PyQt5.QtWidgets import *
7
8 # This is our window from QtCreator
9 import mainwindow_auto
10
11# create class for our Raspberry Pi GUI
12class MainWindow(QMainWindow, mainwindow_auto.Ui_MainWindow):
13 # access variables inside of the UI's file
14 def __init__(self):
15 super(self.__class__, self).__init__()
16 self.setupUi(self) # gets defined in the UI file
17
18# I feel better having one of these
19def main():
20 # a new app instance
21 app = QApplication(sys.argv)
22 form = MainWindow()
23 form.show()
24 # without this, the script exits immediately.
25 sys.exit(app.exec_())
26
27# python bit to figure how who started This
28if __name__ == "__main__":
29 main()
```

With just those bits, you can run your new Python script and have basic GUI functions. Of course buttons and other widgets won’t know what to do yet, but you can click them.

```
1python3 main.py
```

And here’s my GUI:

Add Handlers

Remember when I said to pay attention to the widget names you create? This bit of code is why. Now we need to create handlers in the Python code for each button and widget. In your main.py file, you need to add some functions to your MainWindow class.

```
1 # create class for our Raspberry Pi GUI
2 class MainWindow(QMainWindow, mainwindow_auto.Ui_MainWindow):
3  # access variables inside of the UI's file
4
5  ### functions for the buttons to call
6  def pressedOnButton(self):
7  print ("Pressed On!")
8
9  def pressedOffButton(self):
10 print ("Pressed Off!")
11
12 def __init__(self):
13 super(self.__class__, self).__init__()
14 self.setupUi(self) # gets defined in the UI file
```

```
15
16 ### Hooks to for buttons
17 self.btnOn.clicked.connect(lambda: self.pressedOnButton())
18 self.btnOff.clicked.connect(lambda: self.pressedOffButton())
```

Replace the MainWindow class from the minimal example before with this code. When it runs a message will be printed on the console with each button press.

In this case, I'm only going to provide button examples. With a little bit of Google-fu, you can find example code for other widget types.

Running on startup

Okay if you've already configured the Pi to boot into X-Windows when it starts up, you might want to know how to make your python program run. Well first make sure your "main.py" and "mainwindow_auto.py" files are in the same directory. In my case, I leave them in my "pi" user's home directory.

Edit this file:

```
1nano /home/pi/.config/lxsession/LXDE-pi/autostart
```

And add this line:

```
1@/usr/bin/python3 /home/pi/main.py
```

Now issue a sudo reboot and it should load. (Note, you can use the "which" command to verify the location of the python3 binary.)

Download The Entire project

In case you want to download my Qt and Python code, here is a download button.

Download Tutorial Code

Conclusion

As I said, I think the most difficult thing in this tutorial is getting all of the pieces installed—in two places.

That isn’t to say creating the proper handlers for the widgets is easy. However, there are tons (and tons) of help on how to build Qt programs—even if it the code is in C++.