PyQt5 Basics Tutorial

By Sam Poirier / darthmorf

Contents

Introduction

PyQt5 is, in my opinion a much better GUI module than the python default; tKinter, however it does require a much greater programming knowledge and confidence than tkinter. You will likely need to be able to debug silent crashes (due to the C++ backend of PyQt – errors there aren't piped back to Python properly) and have some knowledge of OOP and a basic understanding of lambda functions.

Despite these extra requirements, it does allow for a much more professional looking project and is much more powerful.

For this guide I will be using python 3.7 but it should work for most versions of Python after 3.2.

There are two sites that have PyQt related docs on them:

http://pyqt.sourceforge.net/Docs/PyQt5/ and

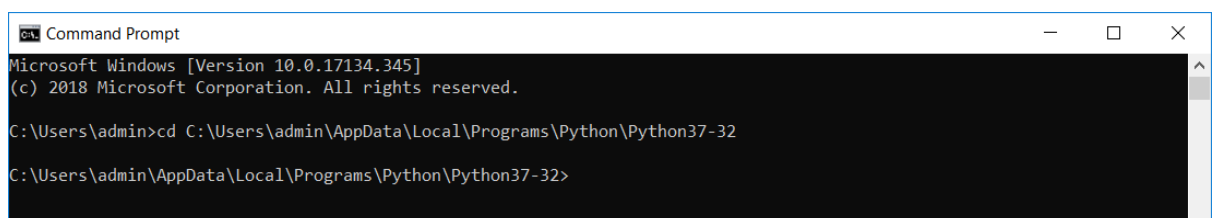https://doc.qt.io/qt-5/reference-overview.html

The second is a much better site but is for the C++ version (Qt) so requires you to translate the code there into Python. Make sure for both that you're on the PyQt5 version!

If you have any suggestions for this tutorial feel free to make an issue.

Installing PyQt5

1. Navigate to the python install directory. Mine is located at:
   'C:\Users\admin\AppData\Local\Programs\Python\Python37-32'
   There are multiple possible locations python could be installed to, so make sure yours is correct.

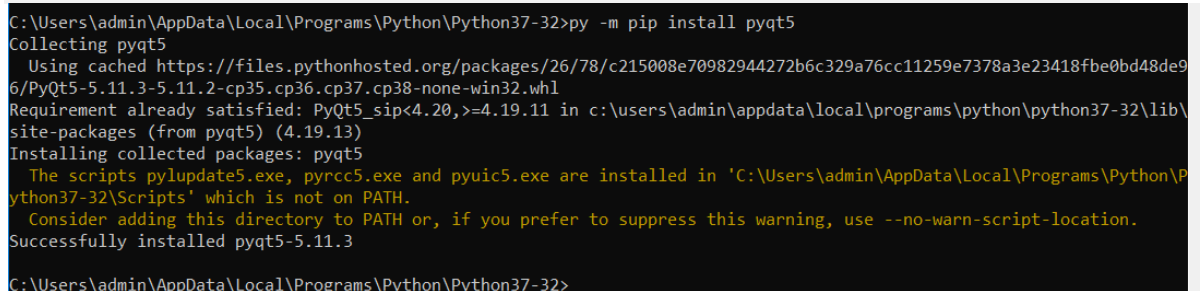2. Open a command prompt and cd to your python install directory:
   ```
   cd <your python path>
   ```


3. Use pip to install PyQt5:
   ```
   py -m pip install pyqt5
   ```


   The output should look something like this – the important part is the line 'Successfully installed pyqt5-x.x.x'

4. You can test that PyQt has been installed correctly by quickly booting up a python shell and attempting to import it. Enter:
   `python`
   then
   `import PyQt5` (once the shell opens)

```
C:\Users\admin\AppData\Local\Programs\Python\Python37-32>python
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import PyQt5
>>>
```

If you get no errors, it is installed correctly!

PyQt Test App – Hardcoding UI

There are two main ways to create GUIs in PyQt – the first is to manually program in each element, and the second is to load an xml file containing the layout of the window. For this first example, we will start with the first method.
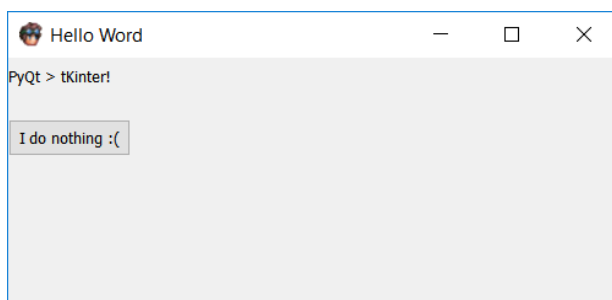

```python
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import * # Import all PyQt5 Widgets

class MainWindow(QMainWindow): # Create a 'mainWindow' class that inherits from
the PyQt equivalent
    def __init__(self, *args):
        super().__init__(*args) # Pass the window arguments to the Qt Class - we
don't need it
        self.resize(500, 200) # Set the window size
        self.setWindowTitle("Hello Word")
        self.setWindowIcon(QIcon("icon.png")) # Load an icon from the same
directory

        self.label = QLabel("PyQt > tKinter!", self) # Create a new label element
        self.button = QPushButton("I do nothing :(", self) # Create a new button
        self.button.move(0, 50) # Move the button 50 pixels down
        self.show() # Show the window


app = QApplication(sys.argv) # Create a PyQt app
mainWindow = MainWindow() # Initialise the MainWindow
sys.exit(app.exec_()) # Wait for the app to close, then close the program
```

This code contains a class inheriting from the PyQt5 QMainWindow class with its own button and label – more types can be found on the docs. The window can be resized and acts and looks like a proper window. It has an icon. It doesn't do anything yet, however. Let's make the label change when we press the button!

The first thing is to create a simple function to update the text of the label. We'll take the new label text as a parameter, and update the button text too:

```python
def updateLabelText(self, text):
        self.label.setText(text)
        self.button.setText("I did stuff :D")
```

We also need to connect the signal we get from the button clicked to this 'updateLabelText' function. We will pass the 'button.clicked.connect' function a lambda function which calls the function we just created above. *(I think I said 'function too many times in that sentence).*

```python
self.button.clicked.connect(lambda: self.updateLabelText("Button was clicked!"))
```

Our code overall now looks like this:

(See 'TestApp.py')

```python
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import * # Import all PyQt5 Widgets

class MainWindow(QMainWindow): # Create a 'mainWindow' class that inherits from
the PyQt equivalent
    def __init__(self, *args):
        super().__init__(*args) # Pass the window arguments to the Qt Class - we
don't need it
        self.resize(500, 200) # Set the window size
        self.setWindowTitle("Hello Word")
        self.setWindowIcon(QIcon("icon.png"))

        self.label = QLabel('PyQt > tKinter!', self) # Create a new label element
        self.button = QPushButton("I do stuff :)", self) # Create a new button
        self.button.move(0, 50) # Move the button 50 pixels down
        self.button.clicked.connect(lambda: self.updateLabelText("Button was
clicked!"))
        self.show() # Show the window

    def updateLabelText(self, text):
        self.label.setText(text)
        self.button.setText("I did stuff :D")


app = QApplication(sys.argv) # Create a PyQt app
mainWindow = MainWindow() # Initialise the MainWindow
sys.exit(app.exec_()) # Wait for the app to close, then close the program
```
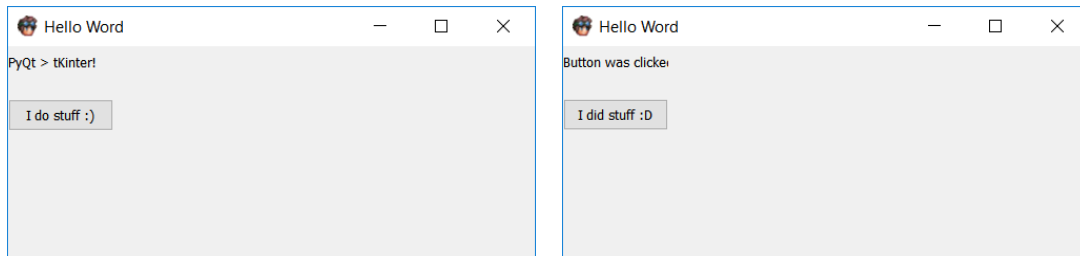
When we run this, we are greeted with the first screen, which updates to the second once the button is clicked:
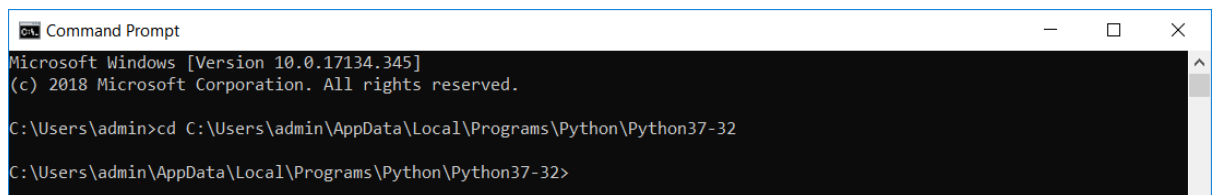
As you can see, the label does not auto-expand to fit the new size. Perhaps now is a good time to set up the designer to create a more advanced UI.
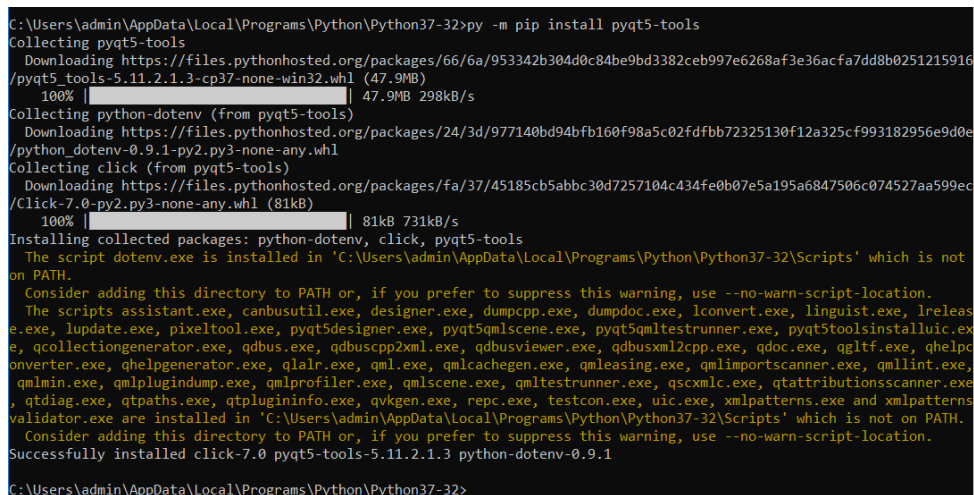
Installing the PyQt5 Designer

1. Once again, open a command prompt and cd to your python install directory:
   (Mine is 'C:\Users\admin\AppData\Local\Programs\Python\Python37-32')
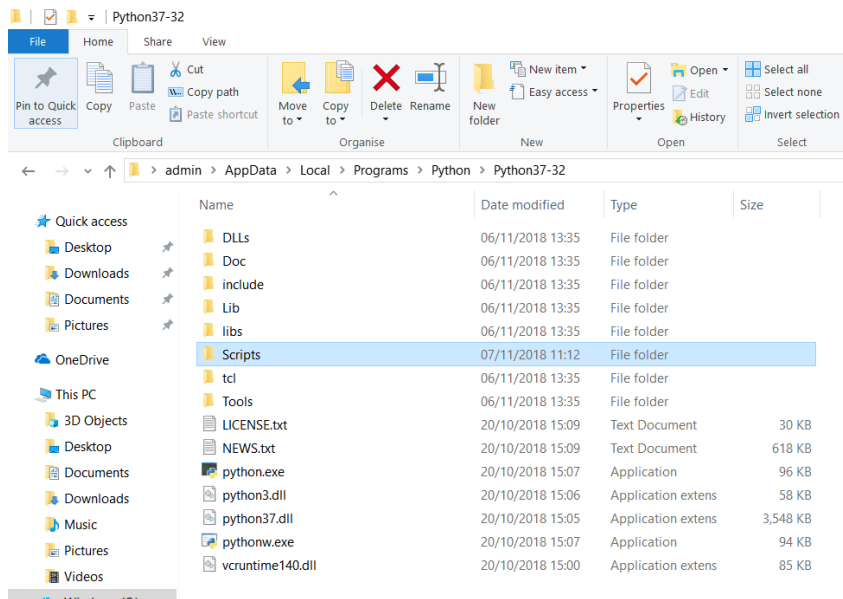
```
cd <your python path>
```

2. Use pip to install the PyQt5 tools:

```
py -m pip install pyqt5-tools
```
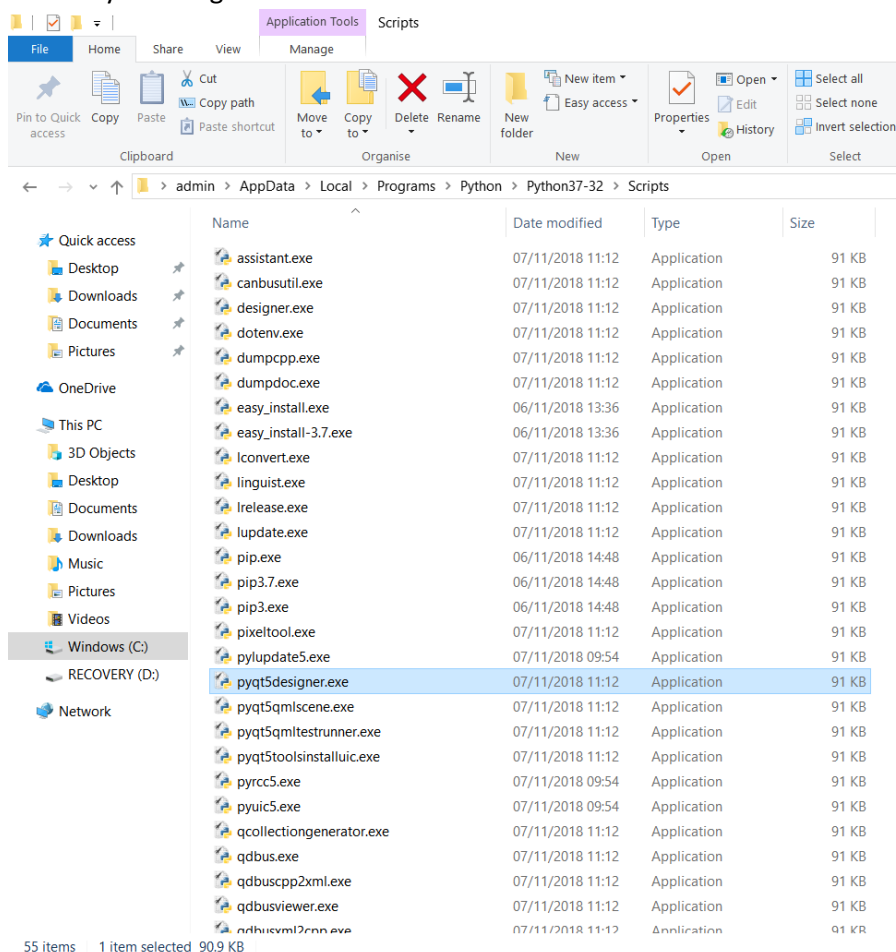
Once it's done it should look something like this – look for the 'Successfully installed' line.
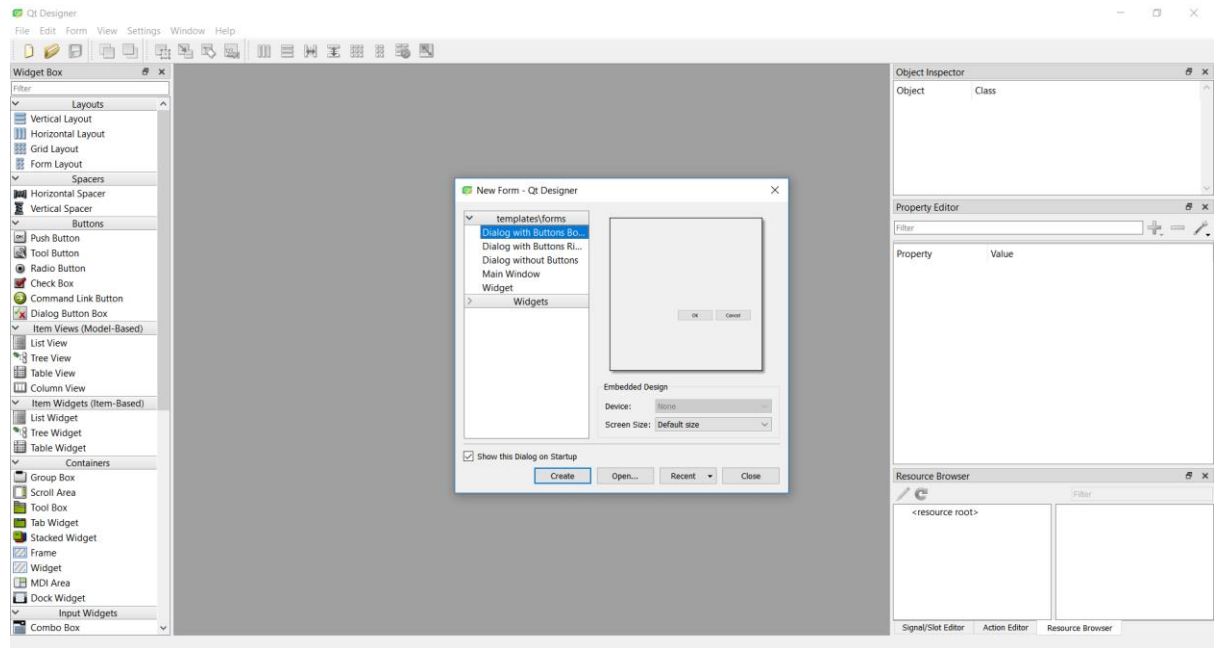
3. Go back to the python install directory, and open the 'scripts' subfolder:

4. Inside, look for 'pyqt5designer.exe' – you might want to create a shortcut to this app as you will likely be using it lots!

5. Open it up, and you should be greeted with a window like this:
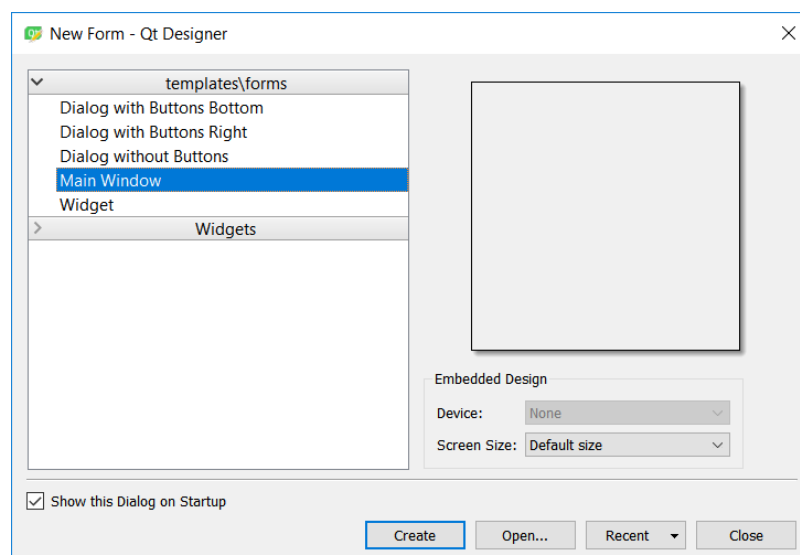

If so, you're ready to start designing an advanced UI!

PyQt Test App – Loading UI

First, we will create a similar UI inside the designer. It operates via drag and drop, and the best way to get used to it is probably to fiddle with it yourself. There are a few important points to cover first however.

We're creating the main window currently, so that is what we will select:

This immediately greets us with an empty window, which if we press ctrl+r we can see a preview of:
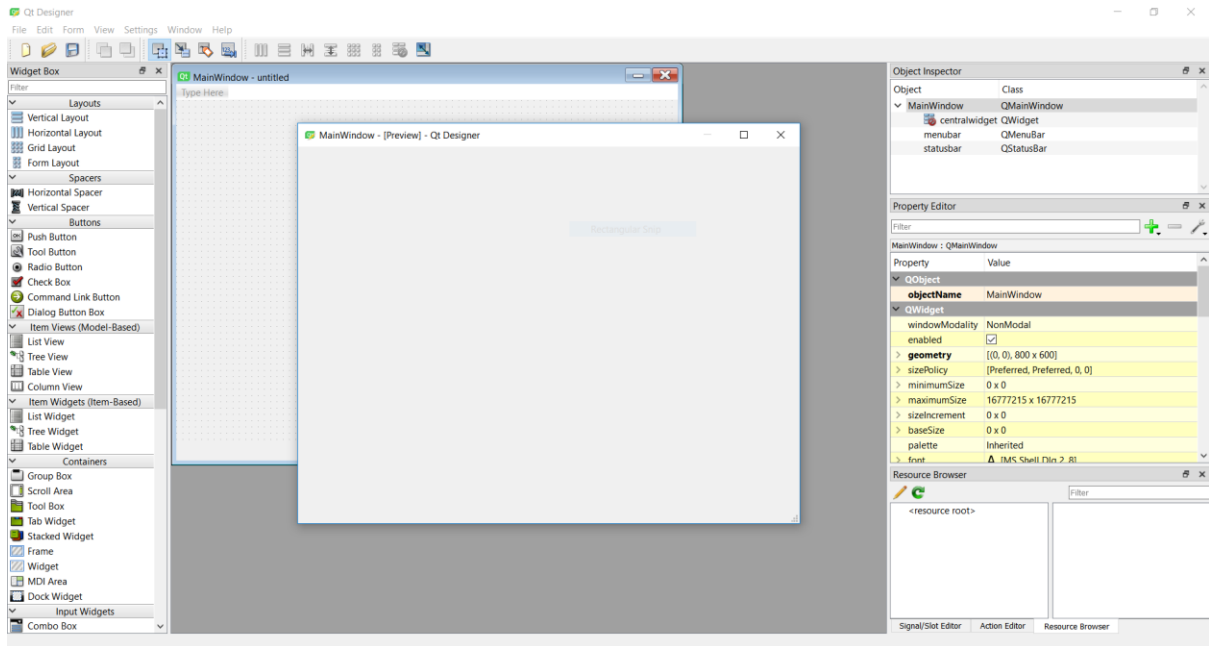
I have now drag-and-dropped a new label and button element onto the page and deleted the menubar at the top as we will not be using it. I have also updated the display text and object name by clicking on the element and updating the values in the 'Object Properties' tab on the right (highlighted in red). It is important to update the objectName to something that makes sense, as we will use that to reference this element in code (I chose myButton and myLabel as they do not do much yet so it's hard to come up with a description).
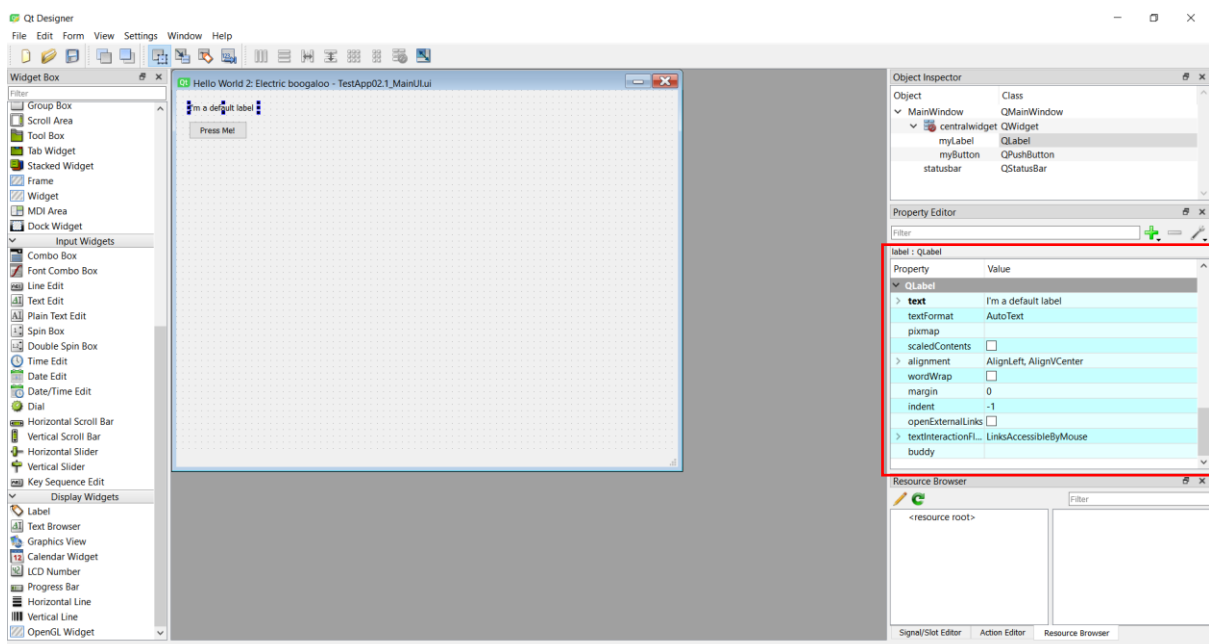
This program saves the files with a '.ui' file extension – this is slightly misleading as they are actually just XML files containing the window data. Mine looks like this:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>322</width>
    <height>123</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Hello World 2: Electric boogaloo</string>
  </property>
  <property name="windowIcon">
   <iconset>
    <normaloff>icon.png</normaloff>icon.png</iconset>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QPushButton" name="myButton">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>50</y>
      <width>93</width>
      <height>28</height>
     </rect>
    </property>
    <property name="text">
     <string>Press Me!</string>
    </property>
   </widget>
   <widget class="QLabel" name="myLabel">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>20</y>
      <width>111</width>
      <height>16</height>
     </rect>
    </property>
    <property name="text">
     <string>I'm a default label</string>
    </property>
   </widget>
  </widget>
  <widget class="QStatusBar" name="statusbar"/>
 </widget>
 <resources/>
 <connections/>
</ui>
```

Now, to implement this XML into our code, we need to import the PyQt library that allows us to load UI files:

```
from PyQt5.uic import loadUi # Import all PyQt5 Widgets
```

Then, remove the code creating the widgets, setting the window size, the icon etc as all of that is inside the UI file. Replace it with this:

```
loadUi("TestApp_MainUI.ui", self) # Load UI layout from file
```

I have also updated all instances of 'button' and 'label' to 'myButton' and 'myLabel' as that's what we named them in the designer:

(see TestApp03.py)

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.uic import loadUi # Import all PyQt5 Widgets

class MainWindow(QMainWindow): # Create a 'mainWindow' class that inherits from
the PyQt equivalent
    def __init__(self, *args):
        super().__init__(*args) # Pass the window arguments to the Qt Class - we
don't need it
        loadUi("TestApp_MainUI.ui", self) # Load UI layout from file

        self.myButton.clicked.connect(lambda: self.updateLabelText("Button was
clicked!"))
        self.show() # Show the window

    def updateLabelText(self, text):
        self.myLabel.setText(text)
        self.myButton.setText('I did stuff :D')


app = QApplication(sys.argv) # Create a PyQt app
mainWindow = MainWindow() # Initialise the MainWindow
sys.exit(app.exec_()) # Wait for the app to close, then close the program
```

As you can see, this code is much more concise, and performs the same function that it did before when we were manually creating the widgets: