

PyQt5 Basics Tutorial

By Sam Poirier / darthmorf

Contents

Introduction	2
Installing PyQt5	2
PyQt Test App – Hardcoding UI.....	3
Installing the PyQt5 Designer.....	5
PyQt Test App – Loading UI.....	7
PyQt Test App – Second Window	11
PyQt Test App – Layouts and window Scaling	13
PyQt Test App – Dynamically Loading Widget Sections	15

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

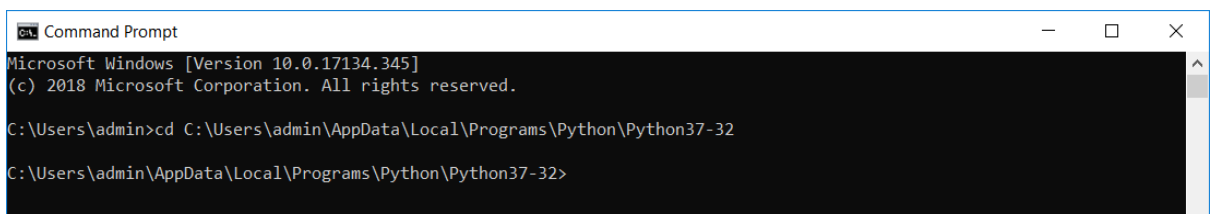
<https://doc.qt.io/qt-5/reference-overview.html>

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>`

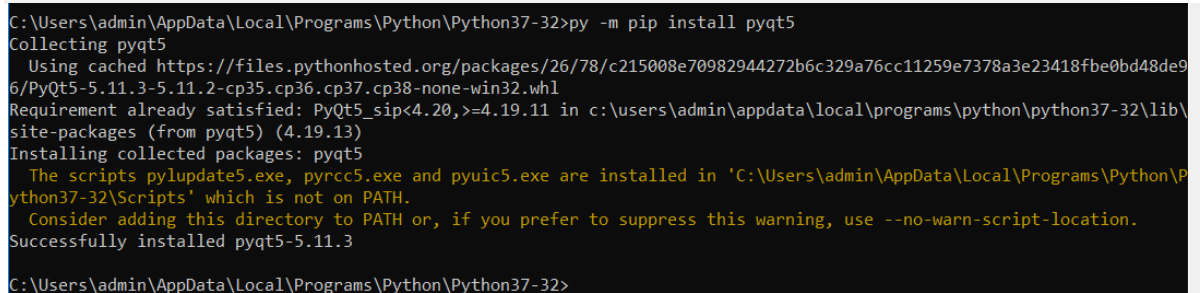


```
Command Prompt
Microsoft Windows [Version 10.0.17134.345]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\admin>cd C:\Users\admin\AppData\Local\Programs\Python\Python37-32
C:\Users\admin\AppData\Local\Programs\Python\Python37-32>
```

3. Use pip to install PyQt5:

```
py -m pip install pyqt5
```



```
C:\Users\admin\AppData\Local\Programs\Python\Python37-32>py -m pip install pyqt5
Collecting pyqt5
  Using cached https://files.pythonhosted.org/packages/26/78/c215008e70982944272b6c329a76cc11259e7378a3e23418fbd0bd48de96/PyQt5-5.11.3-5.11.2-cp35.cp36.cp37.cp38-none-win32.whl
Requirement already satisfied: PyQt5_sip<4.20,>=4.19.11 in c:\users\admin\appdata\local\programs\python\python37-32\lib\site-packages (from pyqt5) (4.19.13)
Installing collected packages: pyqt5
  The scripts pylupdate5.exe, pyrcc5.exe and pyuic5.exe are installed in 'C:\Users\admin\AppData\Local\Programs\Python\Python37-32\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed pyqt5-5.11.3

C:\Users\admin\AppData\Local\Programs\Python\Python37-32>
```

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:

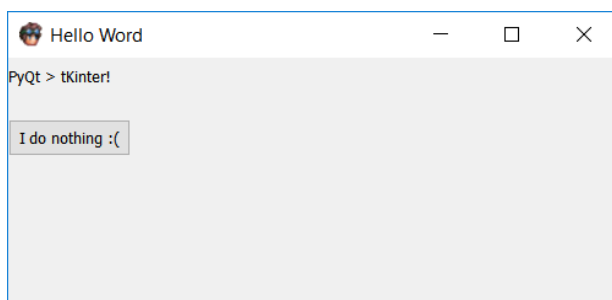
```
python
then
import PyQt5 (once the shell opens)
```

```
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.



```
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:

```
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').*

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

Our code overall now looks like this:

(See 'TestApp.py')

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

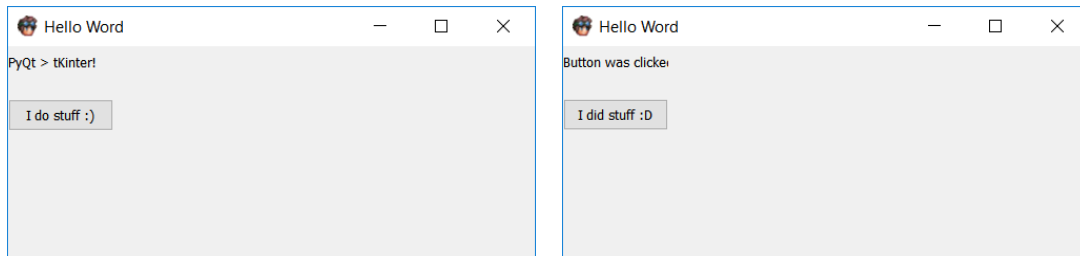
class MainWindow(QMainWindow): # Create a 'mainWindow' class that inherits from
the PyQt equivalent
    def __init__(self, *args):
        super().__init__(*args) # Pass the window arguments to the Qt Class - we
don't need it
        self.resize(500, 200) # Set the window size
        self.setWindowTitle("Hello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

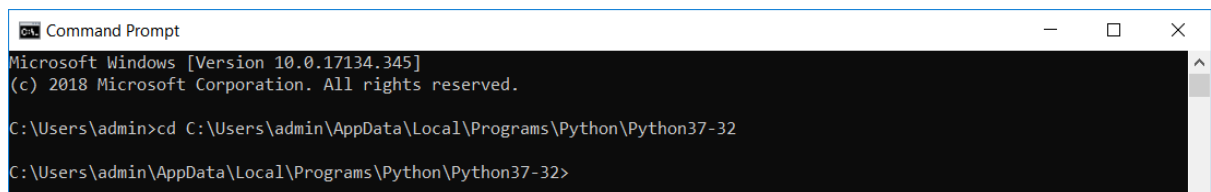


As you can see, the label does not auto-expand to fit the new size. Perhaps now is a good time to set up the designer to create a more advanced UI.

Installing the PyQt5 Designer

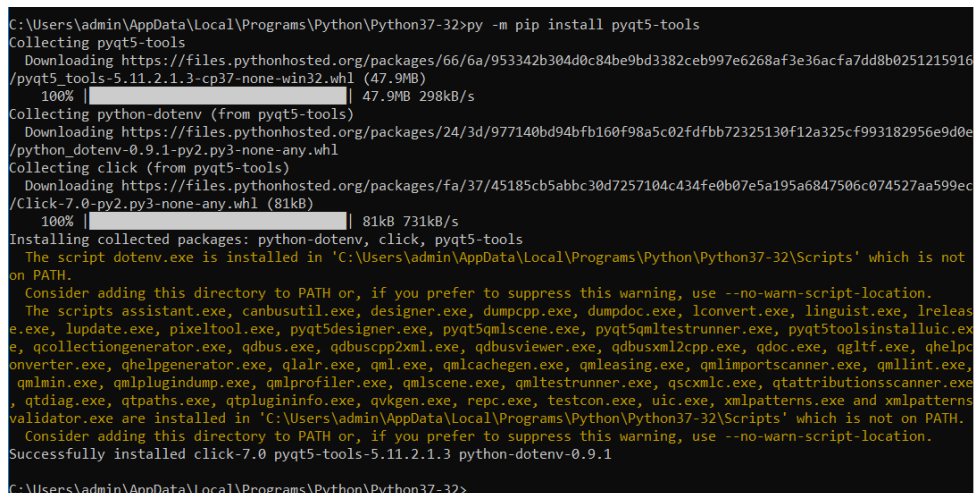
1. Once again, open a command prompt and cd to your python install directory:
(Mine is 'C:\Users\admin\AppData\Local\Programs\Python\Python37-32')

```
cd <your python path>
```



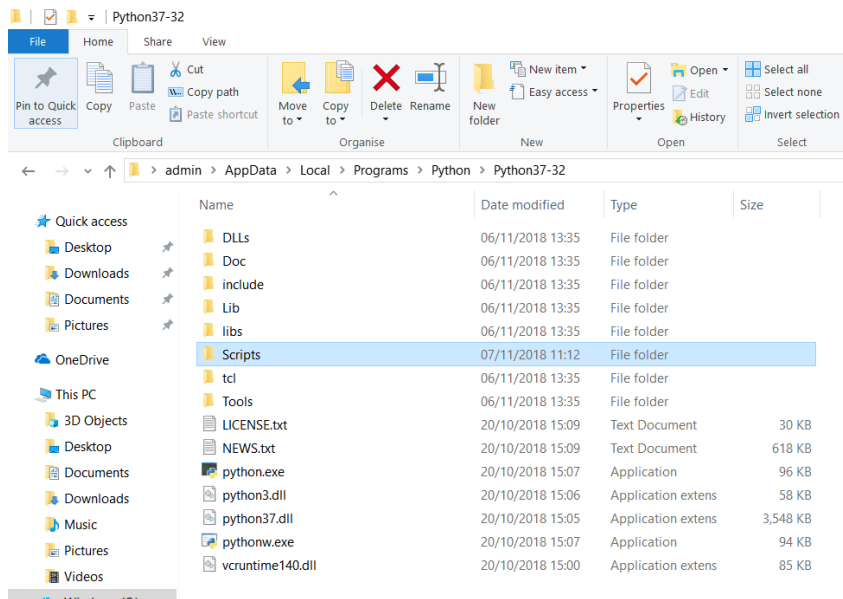
2. Use pip to install the PyQt5 tools:

```
py -m pip install pyqt5-tools
```

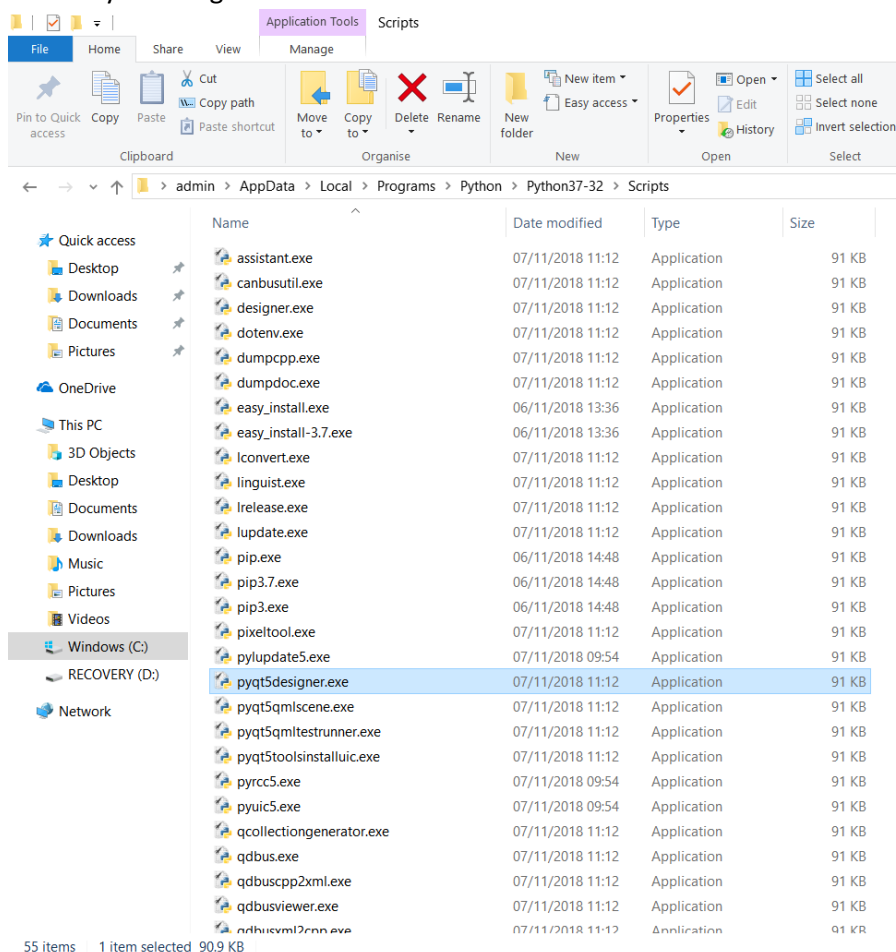


Once it's done it should look something like this – look for the 'Successfully installed' line.

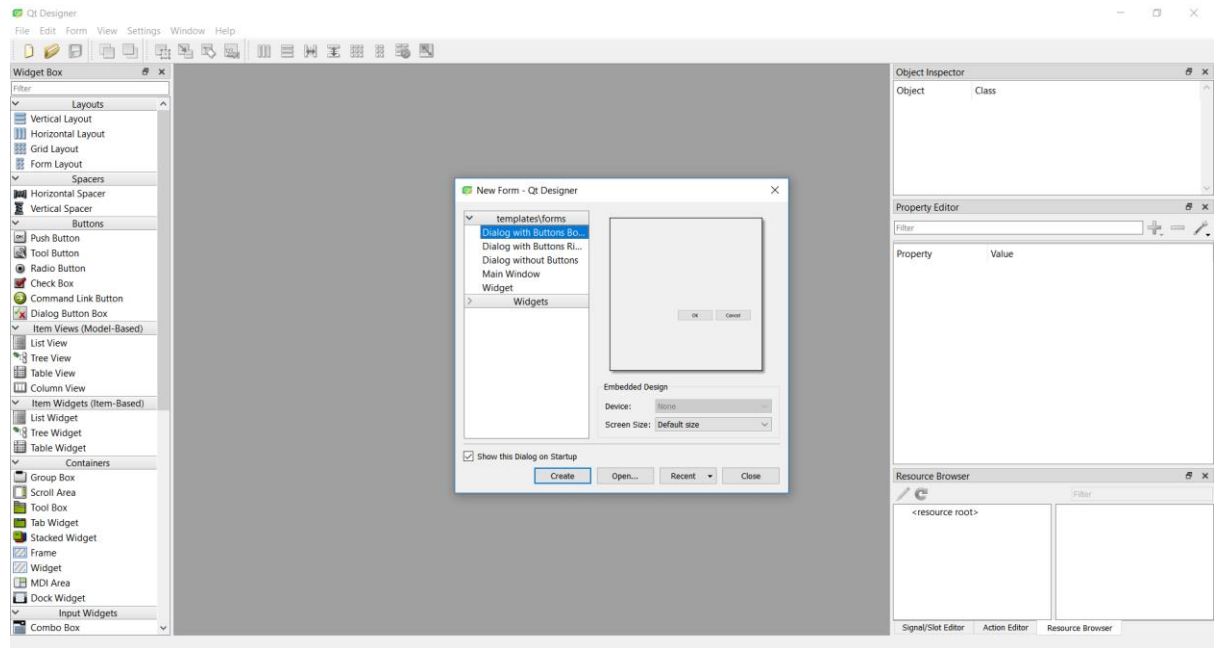
3. Go back to the python install directory, and open the 'scripts' subfolder:



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:

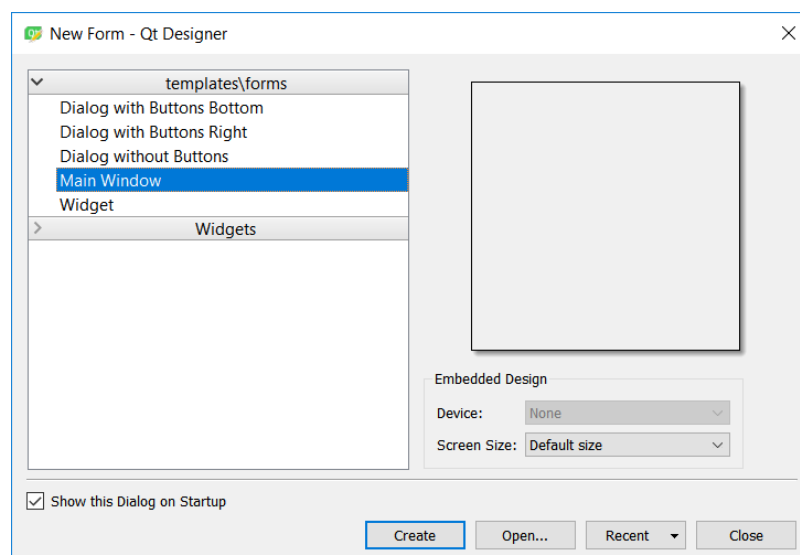


If so, you're ready to start designing an advanced UI!

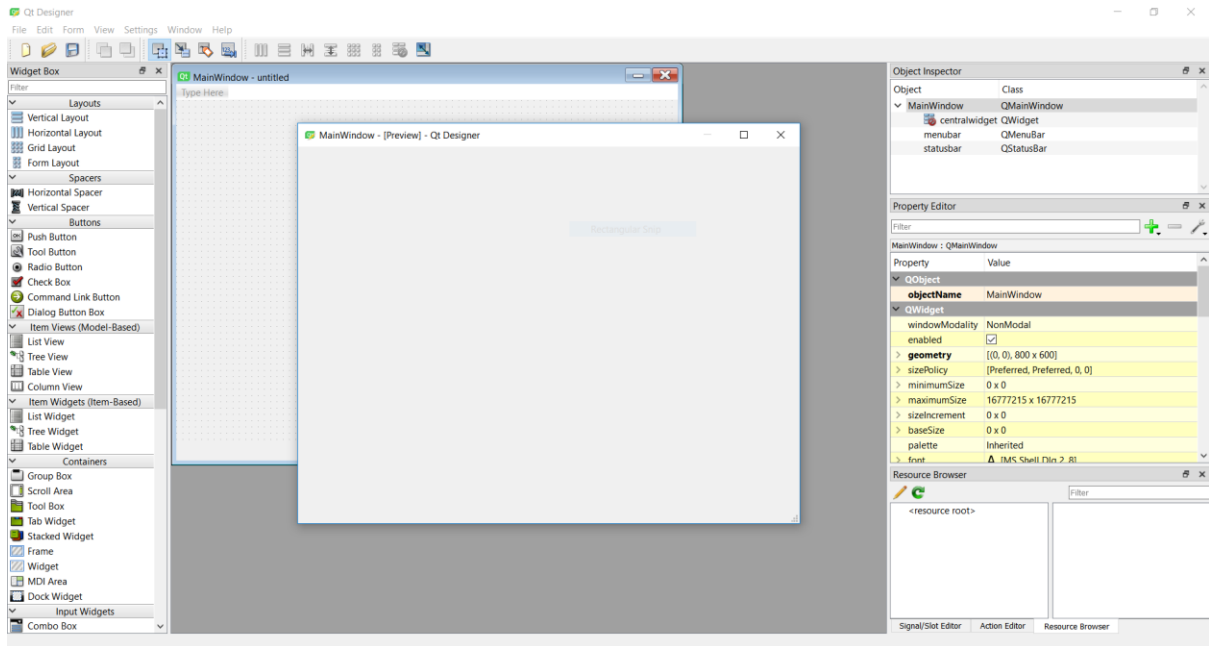
PyQt Test App – Loading UI

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.

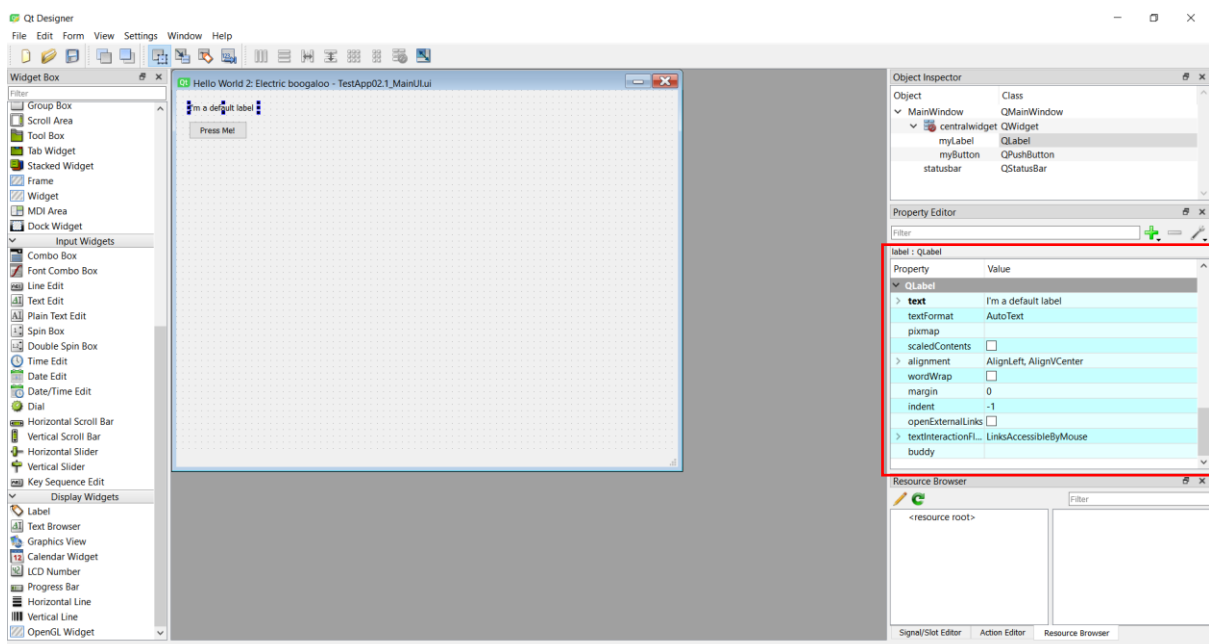
We're creating the main window currently, so that is what we will select:



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 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:

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

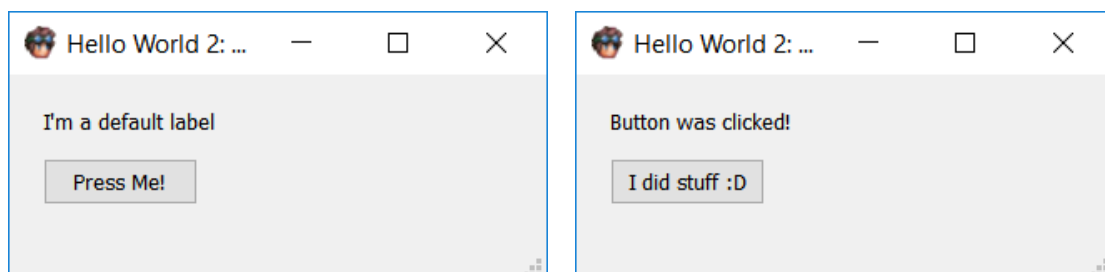
class MainWindow(QMainWindow): # Create a 'mainWindow' class that inherits from
the PyQt equivalent
    def __init__(self, *args):
        super().__init__(*args) # Pass the window arguments to the Qt Class - we
don't need it
        loadUi("TestApp_MainUI.ui", self) # Load UI layout from file

        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')

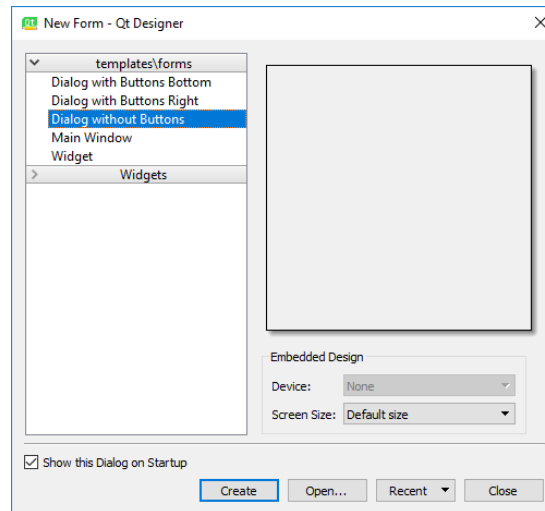
app = QApplication(sys.argv) # Create a PyQt app
mainWindow = MainWindow() # Initialise the MainWindow
sys.exit(app.exec_()) # Wait for the app to close, then close the program
```

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:



PyQt Test App – Second Window

In order to create a second window, we need to create a dialog without buttons:



For the purposes of this tutorial all I've added to it is a label as it doesn't need to do anything. I've also added a second button to the main window which we will use to open the dialog.

(See 'TestApp_MainUI.ui' and 'TestApp_Dialog.ui')

```
<?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>163</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 class="QPushButton" name="openDialogButton">
<property name="geometry">
<rect>
<x>20</x>
<y>88</y>
<width>109</width>
<height>26</height>
</rect>
</property>
<property name="text">
<string>Open A new Dialog</string>
</property>
</widget>
</widget>
<widget class="QStatusBar" name="statusbar">
</widget>
<resources/>
<connections/>
</ui>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>popupDialog</class>
<widget class="QDialog"
name="popupDialog">
<property name="geometry">
<rect>
<x>0</x>
<y>0</y>
<width>163</width>
<height>56</height>
</rect>
</property>
<property name="windowTitle">
<string>Popup! Boo!</string>
</property>
<property name="modal">
<bool>true</bool>
</property>
<widget class="QLabel" name="label">
<property name="geometry">
<rect>
<x>14</x>
<y>13</y>
<width>127</width>
<height>20</height>
</rect>
</property>
<property name="text">
<string>Hello there!</string>
</property>
</widget>
</widget>
<resources/>
<connections/>
</ui>
```

The first thing to do is to create a class for our Dialog:

```
class PopupDialog(QDialog):
    def __init__(self, *args):
        super().__init__(*args)
        loadUi("TestApp_Dialog.ui", self) # Load UI layout from file
        self.setModal(True) # If a dialog is modal, it means you cannot interact
        with the main window while it is open
        self.show()
```

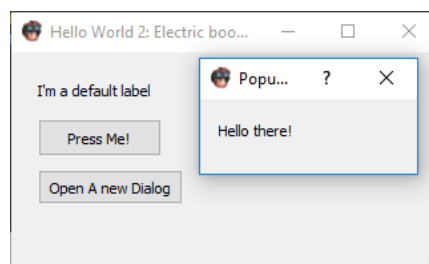
We also need to create a function to open the dialog in the MainWindow class:

```
def openPopupDialog(self):
    self.popupDialog = PopupDialog(self)
```

and connect this function to the second button signal in order to open it:

```
self.openDialogButton.clicked.connect(lambda: self.openPopupDialog())
```

When running this code and pressing the button, you can see that a new dialog appears!



Our code now looks like this:

```
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.openDialogButton.clicked.connect(lambda: self.openPopupDialog())
        self.show() # Show the window

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

    def openPopupDialog(self):
        self.popupDialog = PopupDialog(self)

class PopupDialog(QDialog):
```

```
def __init__(self, *args):
    super().__init__(*args)
    loadUi("TestApp_Dialog.ui", self) # Load UI layout from file
    self.setModal(True) # If a dialog is modal, it means you cannot interact
    with the main window while it is open
    self.show()

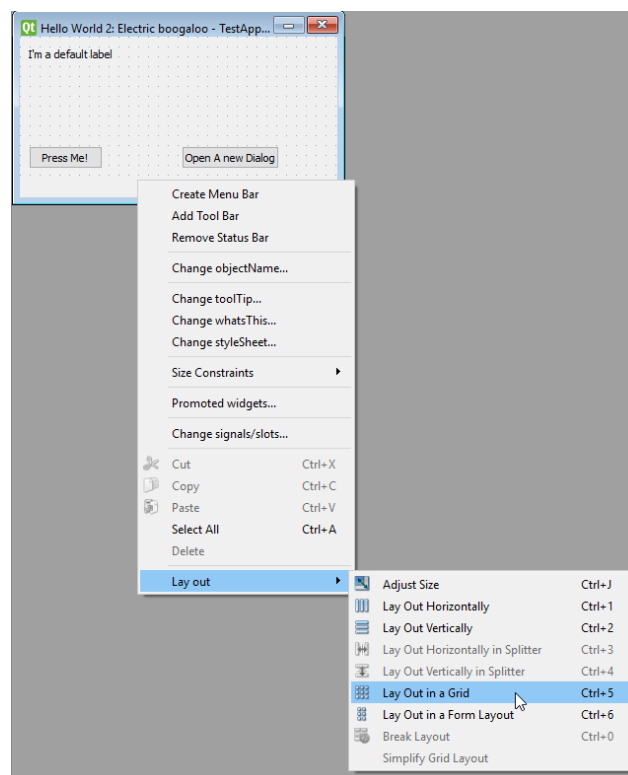
app = QApplication(sys.argv) # Create a PyQt app
mainWindow = MainWindow() # Initialise the MainWindow
sys.exit(app.exec_()) # Wait for the app to close, then close the program
```

PyQt Test App – Layouts and window Scaling

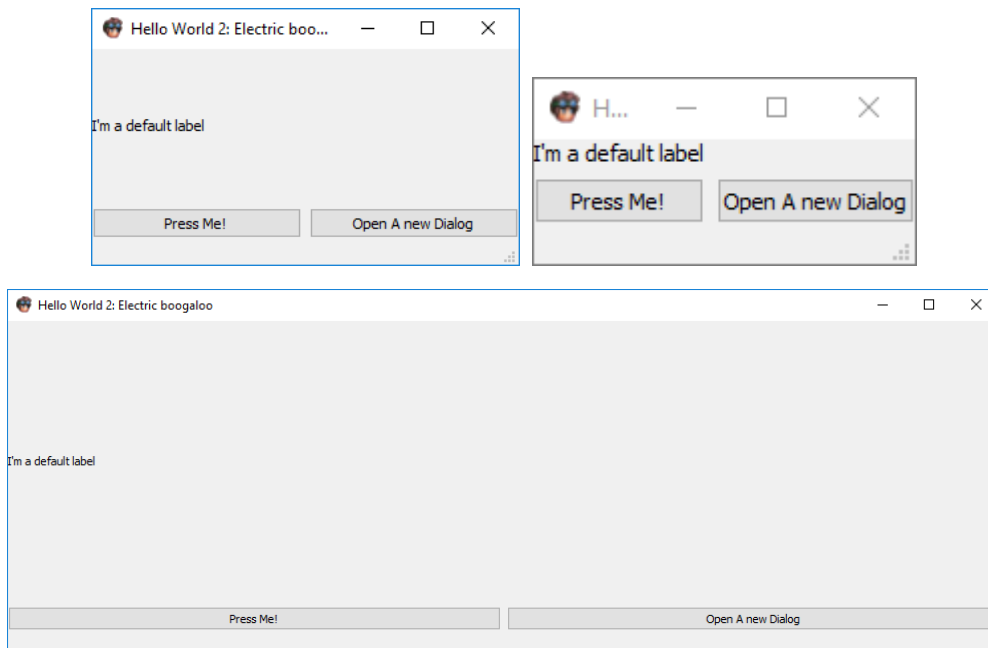
Say we want our window to scale the elements properly when it's resized. In tKinter, you would use a grid layout, where the window is divided up into a grid. Each element takes up a number of squares in the grid, and as the window is scaled up, so are the squares, and anything inside them.

Luckily, PyQt supports grids too! They're probably the easiest of the layout managers in PyQt (The other main one is a 'form layout'. It's even possible to make your own, but that's out of scope for this tutorial).

To enable a grid layout in the window, right click and go to Lay Out > Lay Out in a Grid:



Now, if we run the program, you can see the elements scale properly:



```
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.openDialogButton.clicked.connect(lambda:
        self.openPopupDialog())
        self.show() # Show the window

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

    def openPopupDialog(self):
        self.popupDialog = QDialog(self)

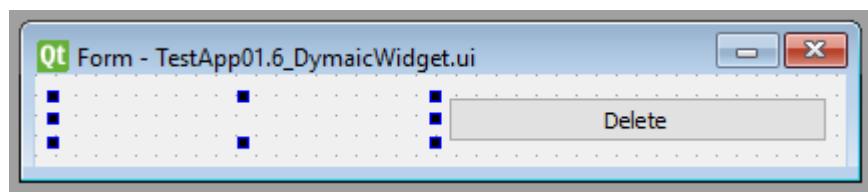
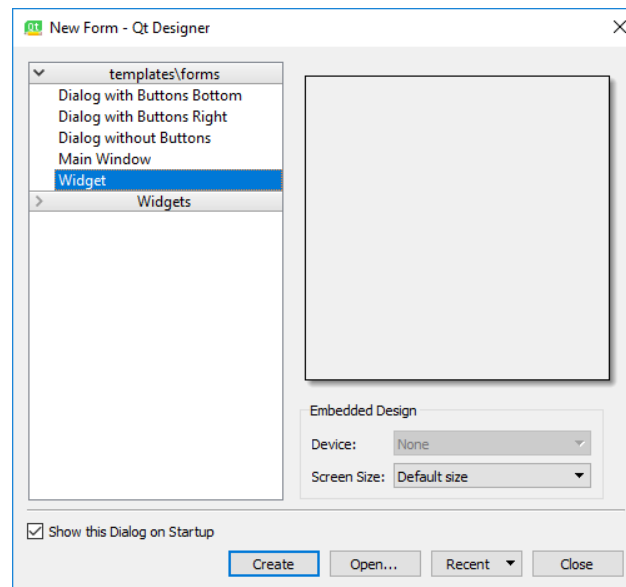
class PopupDialog(QDialog):
    def __init__(self, *args):
        super().__init__(*args)
        loadUi("TestApp_Dialog.ui", self) # Load UI
        layout from file
        self.setModal(True) # If a dialog is modal,
        it means you cannot interact with the main window
        while it is open
        self.show()

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
```

```
<?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>163</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">
<layout class="QGridLayout" name="gridLayout">
<item row="0" column="0">
<widget class="QLabel" name="myLabel">
<property name="text">
<string>I'm a default label</string>
</property>
</widget>
</item>
<item row="1" column="0">
<widget class="QPushButton" name="myButton">
<property name="text">
<string>Press Me!</string>
</property>
</widget>
</item>
<item row="1" column="1">
<widget class="QPushButton" name="openDialogButton">
<property name="text">
<string>Open A new Dialog</string>
</property>
</widget>
</item>
</layout>
</widget>
<widget class="QStatusBar" name="statusbar"/>
</widget>
<resources/>
<connections/>
</ui>
```

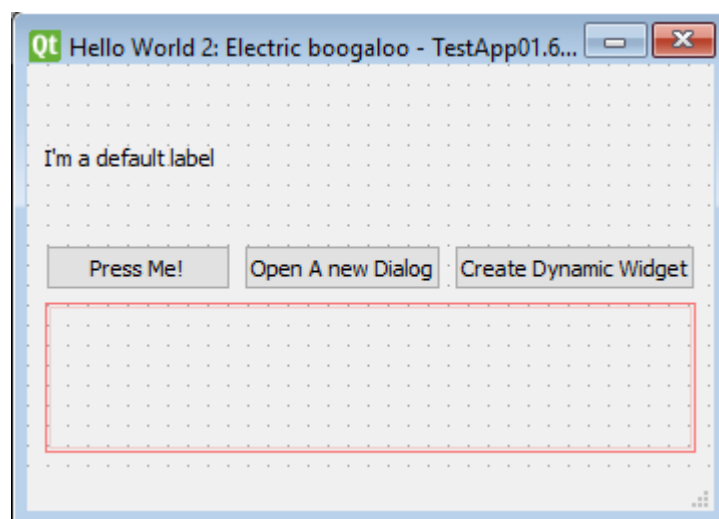
PyQt Test App – Dynamically Loading Widget Sections

This time, we want to make a new widget:



I've made a widget with a label and a button ('infoLabel' and 'deleteButton').

I have also added a button and a boxLayout ('dynamicWidgetButton' and 'dynamicWidgetLayout') element to the main window:



The aim of this is to be able to press the 'Create Dynamic Widget' button, which will create a new instance of the dynamic widget, displaying its own number, inside the boxLayout. Each widget will have a button on it to delete it.

To start with, lets implement the class for the custom dynamic dialog:

```
class DynamicWidget(QWidget):
    def __init__(self, *args, text):
        super().__init__(*args)
        loadUi("TestApp_DynamicWidget.ui", self)
        self.infoLabel.setText(text)
        self.deleteButton.clicked.connect(lambda: self.delete())

    def delete(self):
        self.setParent(None)
```

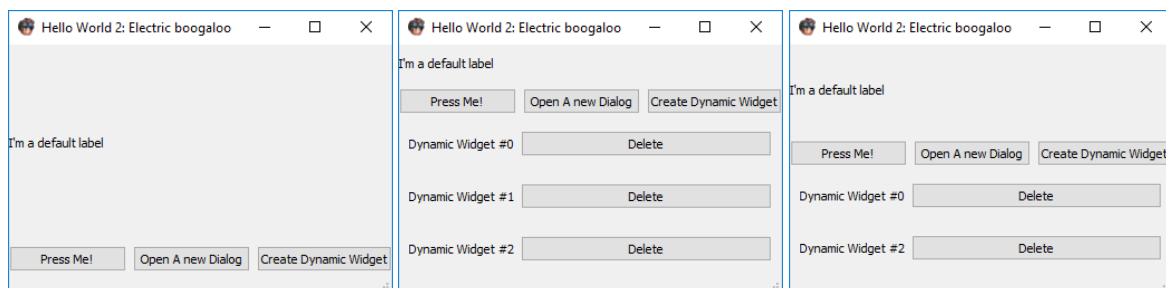
This class takes a parameter ('text') which it sets to the label contents and has a button that deletes itself by utilising the python memory manager; when it unassigns its own parent, the object is no longer referenced anywhere in memory and is therefore cleaned up/deleted.

Next, we will write the function to create this widget. This will be a property of the MainWindow class:

```
def createDynamicWidget(self):
    displayText = "Dynamic Widget #" + str(self.dynamicLayout.count()) # Get
the number widget this is
    dynamicWidget = DynamicWidget(self, text=displayText) # Create the widget
    self.dynamicLayout.addWidget(dynamicWidget) # Add the widget to the layout
```

and of course, hook up the function to the button click event:

```
self.dynamicWidgetButton.clicked.connect(lambda: self.createDynamicWidget())
```



The first screenshot shows the app on launch. The second is after the 'Create Dynamic Widget' button is clicked three times and the third is when the 'Delete' button is pressed on the middle widget.