PyQt: Getting started with PyQt and Qt Designer

This tutorial is aimed at beginners just starting out with PyQt/PySide and Qt Designer, it will cover very basic usage of PyQt in combination with Qt Designer.

The tutorial will guide you, step by step, towards creating a very simple app that lists all files in the selected directory.

Prerequisites

You need PyQt and Qt Designer installed, and of course python.

I'll be using PyQt4 with python 2.7.10 but there are no major differences between PyQt and PySide or python 3 versions of those, so if you already have PyQt5 or PySide installed there is no need to downgrade/switch.

If you don't have anything installed you can get PyQt for Windows here:

http://www.riverbankcomputing.com/software/pyqt/download

It comes with Qt Designer bundled.

For OS X you can download the PyQt via homebrew:

```
$ brew install pyqt
```

And QtCreator (which contains Qt Designer) here: http://download.qt.io/archive/qt/4.8/4.8.6/

On Linux the packages required are probably in your distro repositories, if you're on Ubuntu/Debian you can run:

```
$ apt-get install python-qt4 pyqt4-dev-tools qt4-designer
```

After you're done installing requirements on your operating system open terminal/command prompt and make sure you can run `pyuic4` command it should show:

```
$ pyuic4
Error: one input ui-file must be specified
```

If you get "command not found" or something along those lines try googling on how to solve it for your operating system and pyqt version.

If you're on windows you most likely don't have `C:\Python27\Scripts` (replace `27` with your python version) in your `PATH`.

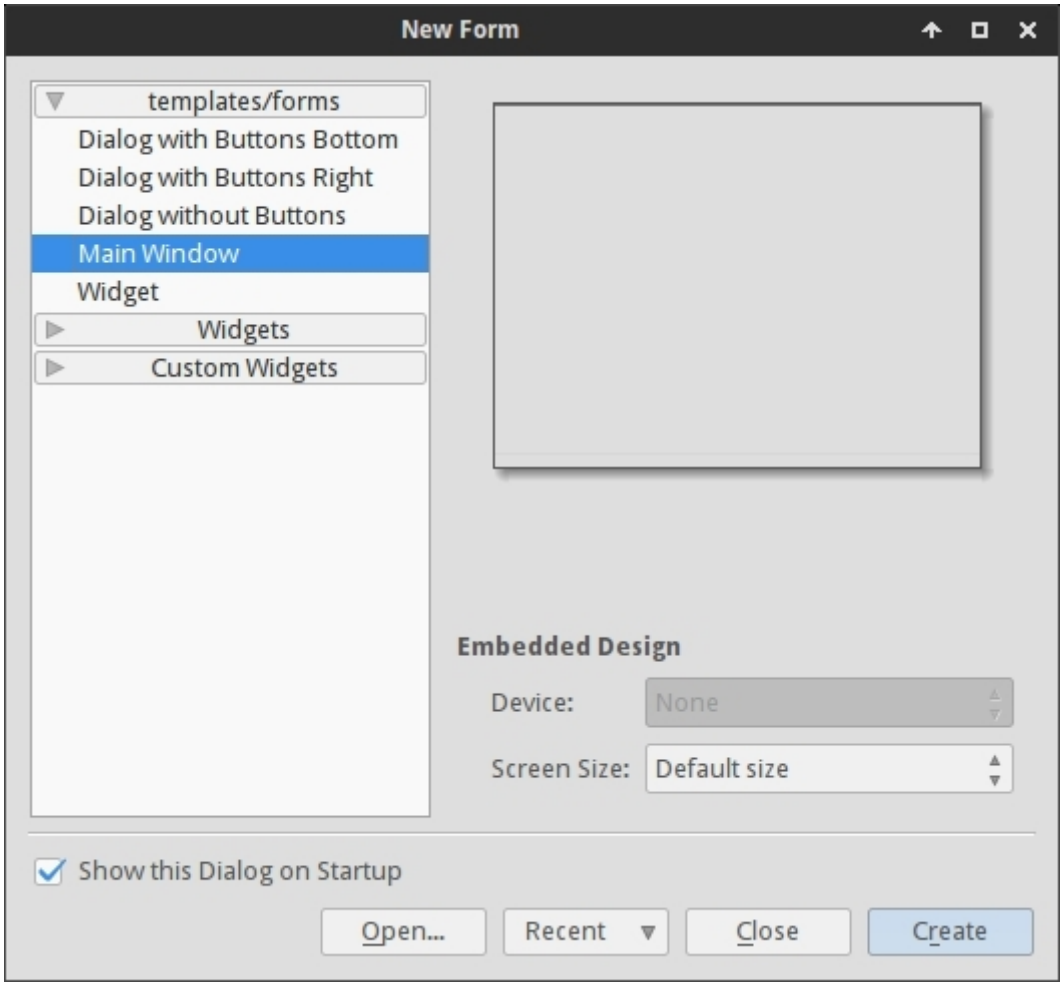
To see how to solve that look at this SO thread

Design

Basics

Now that we've got everything that we need installed let's first start with simple design.

Open up Qt Designer and you should see a new form dialog, pick a "Main Window" and click "Create"



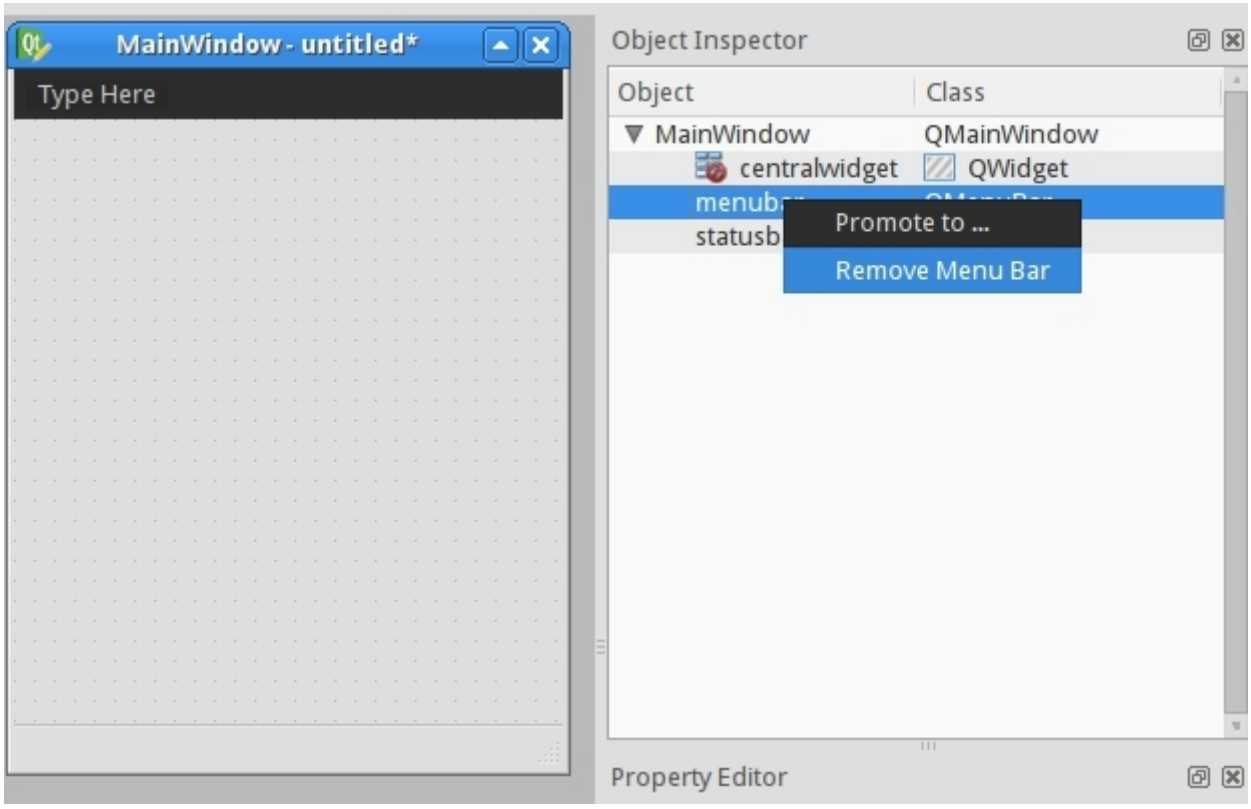
Qt Designer New Form Window

After that you should get a new form that you can resize, drop objects from widget box on etc. Make yourself familiar with the interface, it's pretty simple.

Once we got that we'll resize our main window a bit, since we don't need it that large, and we'll also remove the automatically added menu and status bar since we don't plan on using them in this tutorial.

All the form elements that your design has, and their hierarchy, are listed (by default) on the right side of the Qt Designer window under "Object Inspector". You can easily remove objects by right clicking on them in that window or just by selecting them on your main form and pressing DEL key on your keyboard.

For now we'll just resize our form and delete menu and status bar.



Object Inspector Removing Items

Once we do that we have an (almost) empty form. The only object still left is "centralwidget" but we need it so we won't change anything about it.

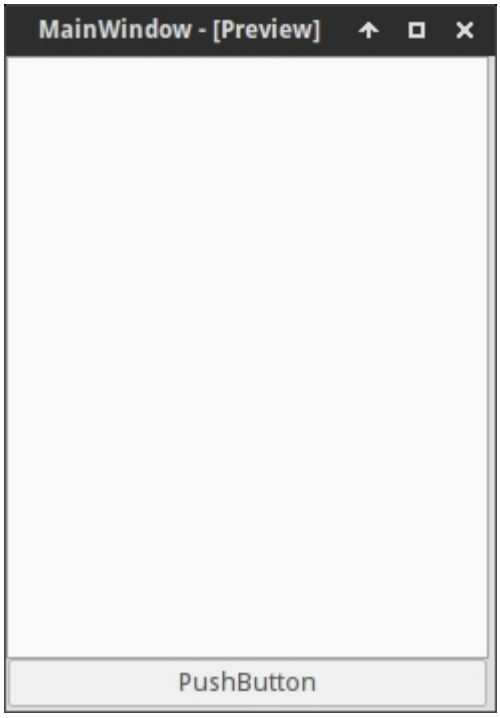
Now drag and drop from the "Widget Box" in Qt Designer a "List Widget" (not List View) widget and a "Push Button", drop them anywhere on the main form.

Layouts

Instead of using fixed positions and sizes of the elements in your application you should be using layouts. Fixed positions and sizes will look good on your end (at least until you resize the window) but you can never be sure that they'll look exactly the same on other machines and/or operating systems.

Layouts are basically containers for your widgets which will keep them in certain positions in relation to other elements, they'll also resize as the main window size changes. There are many features that you can set for both widgets and layouts but I won't go in depth a lot about them.

Let's design our form first without using layouts. Drag and resize the list and button on the main form so that they look something like this:



Fixed Design Preview

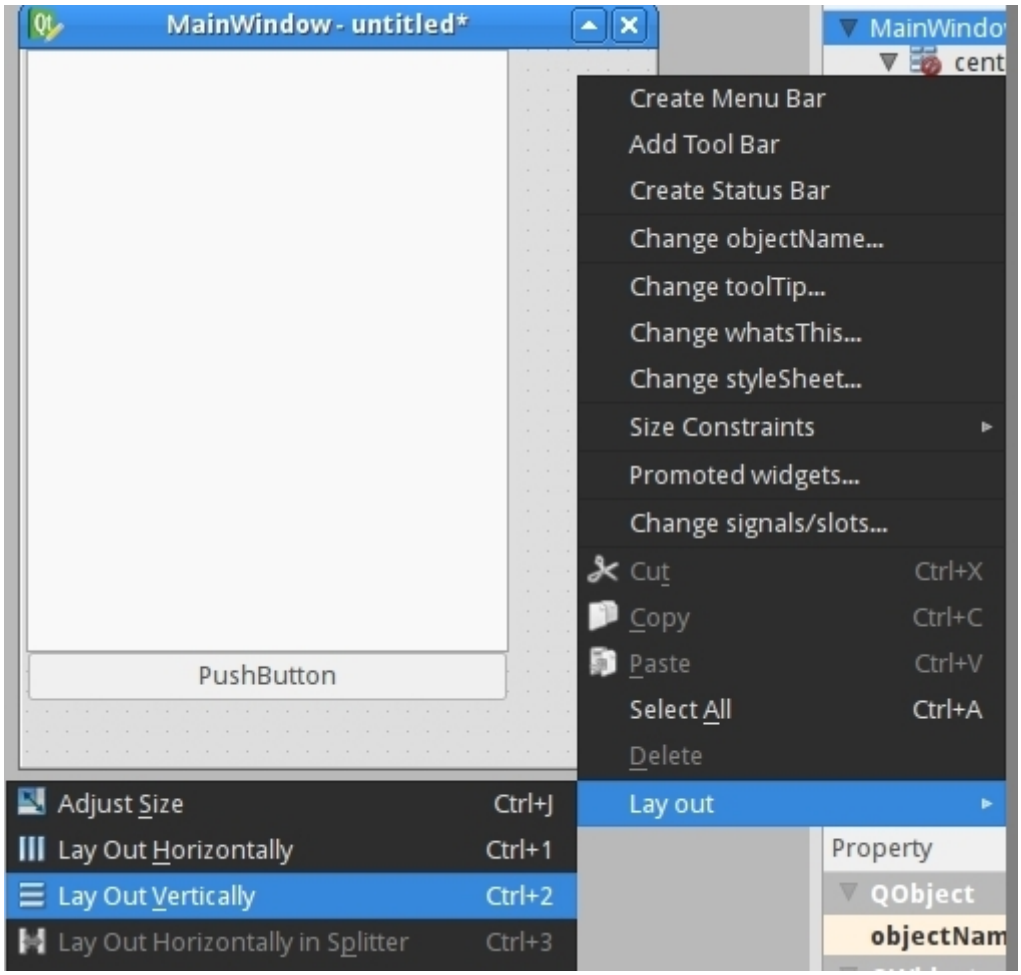
Now in Qt Designer menu click "View" then pick the "Preview" option, you should get the something like the screenshot above. It looks good right? But watch what happens when we increase our window size:

Fixed Design Resized

Our objects stay in same positions and have same sizes even though the main form changed size, and even though the button is almost invisible. That's why you should use layouts more often than not when designing things. Of course in some instances you will want fixed width, or minimum/maximum width of an object. But generally speaking you should be using layouts in your application.

The main window already supports having layouts, so we don't need to add any new ones to our form. Simply right click on the "Main Window" in "Object Inspector" and pick

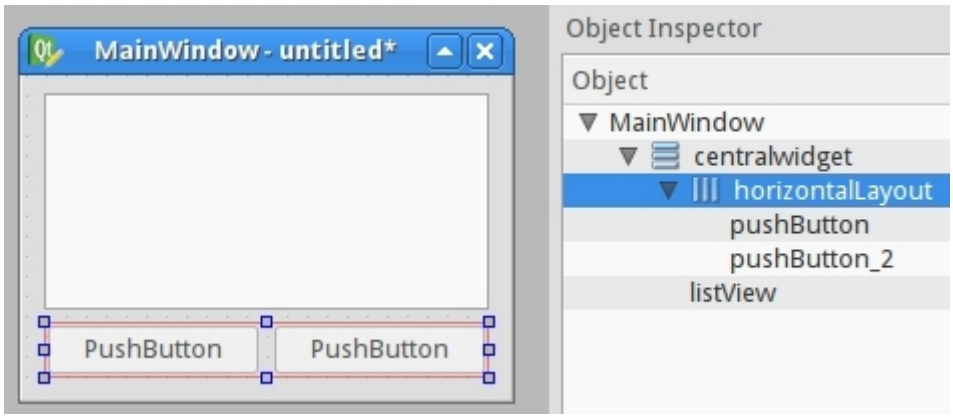
"Layout">"Lay out vertically". You can also right click in the empty space on your main form and choose the same option like this:



Main Window Change Layout

Your elements should be in the same order they were when you changed the layout, but in case they aren't you can simply drag and drop them to the location you want.

Since we've used vertical layout all elements we add will be in vertical order. You can combine layouts to get the desired look you want. For **example**: Horizontal layout with 2 buttons in Vertical layout will look like this:



Layout Inside Layout Example

If you have trouble placing an element by dragging it in main form you can also drag and drop it in Object Inspector window.

Finishing Touches

Now that we've used vertical layout our elements are aligned properly. The only thing left to do is (optional, but recommended) change name of the elements and the text they display.

In simple app like this with only a list and a button element changing names isn't mandatory since it's easy to use anyway, but properly naming elements is something that you should get used to doing.

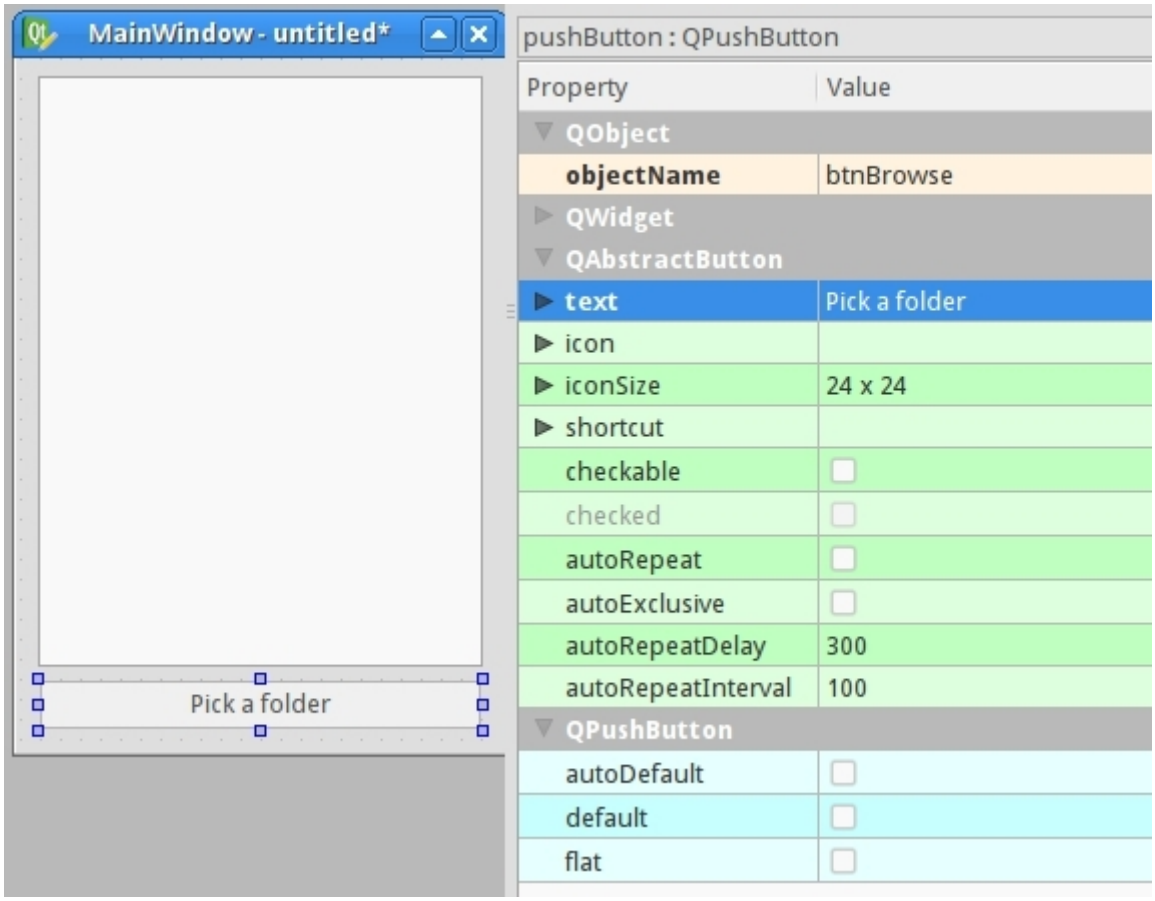
We can change most of the element properties that we'd normally change in the "Property Editor" part of the Qt Designer.

Hint: You can resize, remove, or add elements to the Qt Designer interface that you use often to improve your work-flow. You can add hidden/closed parts of the interface via "View" menu option. I personally only have Object Inspector and Property Editor on the right side of Qt Designer.

Click on the button you added to the form. Now in Property Editor you should see all properties associated with that element, at this time we're just interested in "objectName" and "text" in "QAbstractButton" section. You can minimize the sections in Property Editor by clicking the title of the section.

Change "objectName" to "btnBrowse" and "text" to "Pick a folder".

It should look like this:



Button Property Editor

Object name of the list is "listWidget" which is good enough for in this case.

Save the design as `design.ui` in your project folder which can be wherever you want.

Design to Python code

While it's possible to use `.ui` files directly from python code, I found the method of converting the `.ui` code to python file that we can import and use, easier.

To convert the file we use `pyuic4` command from terminal/command prompt that should have been installed when you've installed PyQt. If you get an error please Google on how to use the command `pyuic` on your operating system.

To convert the design file to python code saved as `design.py`, use `cd` command to change to the directory holding the `design.ui` file and simply run:

```
$ pyuic4 design.ui -o design.py
```

If you want to specify full path for either input or output file you can do that like this:

```
$ pyuic4 path/to/design.ui -o output/path/to/design.py
```

Writing the code

Now that we have the `design.py` file with the necessary design part of the application we can create our main application code and logic.

Create a file `main.py` in the same folder as your `design.py` file.

Using the design

For the application we'll need the following python modules imported:

```
from PyQt4 import QtGui
import sys
```

We also need the design code we created in the previous steps so add this too:

```
import design
```

Since the design file will be completely overwritten each time we change something in the design and recreate it we will not be writing any code in it, instead we'll create a new class e.g. `ExampleApp` that we'll combine with the design code so that we can use all of its features, like this:

```
class ExampleApp(QtGui.QMainWindow, design.Ui_MainWindow):
    def __init__(self, parent=None):
        super(ExampleApp, self).__init__(parent)
        self.setupUi(self)
```

In that class we'll interact with the GUI elements, add connections and everything else we need. But first we'll need to initialize that class on our code startup, we'll handle the class instance creation and other stuff in our `main()` function:

```
def main():
    app = QtGui.QApplication(sys.argv)
    form = ExampleApp()
    form.show()
    app.exec_()
```

And to execute that main function we'll use well known:

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

In the end our whole `main.py` file looks like this (with short explanations of the code):

```
from PyQt4 import QtGui # Import the PyQt4 module we'll need
import sys # We need sys so that we can pass argv to QApplication

import design # This file holds our MainWindow and all design related things
              # it also keeps events etc that we defined in Qt Designer

class ExampleApp(QtGui.QMainWindow, design.Ui_MainWindow):
    def __init__(self):
        # Explaining super is out of the scope of this article
        # So please google it if you're not familar with it
        # Simple reason why we use it here is that it allows us to
        # access variables, methods etc in the design.py file
        super(self.__class__, self).__init__()
        self.setupUi(self)  # This is defined in design.py file automatically
                            # It sets up layout and widgets that are defined


def main():
    app = QtGui.QApplication(sys.argv)  # A new instance of QApplication
    form = ExampleApp()                 # We set the form to be our ExampleApp (design)
    form.show()                         # Show the form
    app.exec_()                         # and execute the app


if __name__ == '__main__':              # if we're running file directly and not importing it
    main()                              # run the main function
```

Running that will bring up our app running completely from python code!

But clicking button isn't doing anything, so we need to implement those features ourselves.

Implementing functions

(All of the following code is written inside the `ExampleApp` class)

Let's start with the "Pick a folder" button.

To connect a button event, such as clicked, to a function we use the following code:

```
self.btnBrowse.clicked.connect(self.browse_folder)
```

And add it to the `__ini__` method of our `ExampleApp` class so that it's set up when the application starts.

Explanation of that line of code:

`self.btnBrowse` - btnBrowse is the name of the object we defined in Qt Designer. `self` is self exaplanatory and means that it belongs to current class.

`clicked` - the event we want to connect. Various elements have various events, for example list widgets have `itemSelectionChanged` etc.

`connect()` - used to specify with what we want to connect it with. In our example:

`self.browse_folder` - simply a function name that we'll write inside our `ExampleApp` class:

For getting the directory browser dialog we can use the built in `QtGui.QFileDialog.getExistingDirectory` method like this:

```
directory = QtGui.QFileDialog.getExistingDirectory(self,"Pick a folder")
```

If the user picks a directory the `directory` variable will be equal to absolute path of the selected directory, otherwise it's `None`. To avoid running our code any further if the user cancels the folder browse dialog we'll use `if directory:` statement.

To list the directory contents we'll need to add `os` to our imports:

```
import os
```

and to get current file list we can use `os.listdir(path)`.

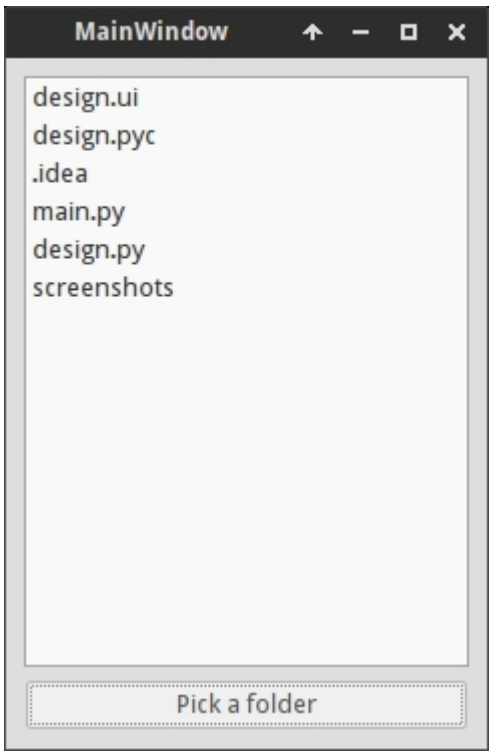
For adding items to the listWidget we use `addItem()` method on it, and to clear all existing items simply use `self.listWidget.clear()`

In the end our `browse_folder` function looks something like this:

```
def browse_folder(self):
    self.listWidget.clear()
    directory = QtGui.QFileDialog.getExistingDirectory(self,
                                                       "Pick a folder")

    if directory:
        for file_name in os.listdir(directory):
            self.listWidget.addItem(file_name)
```

Now you can run your app by typing `python main.py` and you should get the layout you desgined and picking the folder will populate list with folder items.



Finished App After Selecting Folder

Finished main.py

```
from PyQt4 import QtGui  # Import the PyQt4 module we'll need
import sys  # We need sys so that we can pass argv to QApplication

import design  # This file holds our MainWindow and all design related things

# it also keeps events etc that we defined in Qt Designer
import os  # For listing directory methods


class ExampleApp(QtGui.QMainWindow, design.Ui_MainWindow):
    def __init__(self):
        # Explaining super is out of the scope of this article
        # So please google it if you're not familar with it
        # Simple reason why we use it here is that it allows us to
        # access variables, methods etc in the design.py file
        super(self.__class__, self).__init__()
        self.setupUi(self)  # This is defined in design.py file automatically
        # It sets up layout and widgets that are defined
        self.btnBrowse.clicked.connect(self.browse_folder)  # When the button is pressed
                                                            # Execute browse_folder function

    def browse_folder(self):
        self.listWidget.clear() # In case there are any existing elements in the list
        directory = QtGui.QFileDialog.getExistingDirectory(self,
                                                           "Pick a folder")
        # execute getExistingDirectory dialog and set the directory variable to be equal
        # to the user selected directory

        if directory: # if user didn't pick a directory don't continue
            for file_name in os.listdir(directory): # for all files, if any, in the directory
                self.listWidget.addItem(file_name)  # add file to the listWidget


def main():
    app = QtGui.QApplication(sys.argv)  # A new instance of QApplication
    form = ExampleApp()  # We set the form to be our ExampleApp (design)
    form.show()  # Show the form
    app.exec_()  # and execute the app
```

```
if __name__ == '__main__':  # if we're running file directly and not importing it
    main()  # run the main function
```

That's the basic logic of using Qt Designer and PyQt to design and develop a GUI application.

You can safely edit your design and re-run the `pyuic4` command without fear of your main code getting lost or overwritten.

Final notes

All the code in this example is done in the main UI thread, which is often a bad idea - especially if you have something that takes time to finish. For example if the `listdir` function took long time our UI would be frozen the whole time. In one of the future tutorials I'll cover PyQt threading, this one is focused on just the bare basics without too much unnecessary things. But please keep in mind that anything that doing stuff in main UI thread is generally a bad idea.

All the code and files used can be found in this gist: https://gist.github.com/Nikola-K/615def9e197d5db04bef

Useful links:

Python Inheritance SO thread: http://stackoverflow.com/questions/576169/understanding-python-super-with-init-methods

PyQt reference guide: http://pyqt.sourceforge.net/Docs/PyQt4/

PyQt class reference: http://pyqt.sourceforge.net/Docs/PyQt4/classes.html

Written by

Nikola Kovacevic

is a developer specializing in Python but his talents go beyond that. He's done freelance consulting/development for 3 years and gives back to the open source world through teaching and code.

Published 27 May 2015