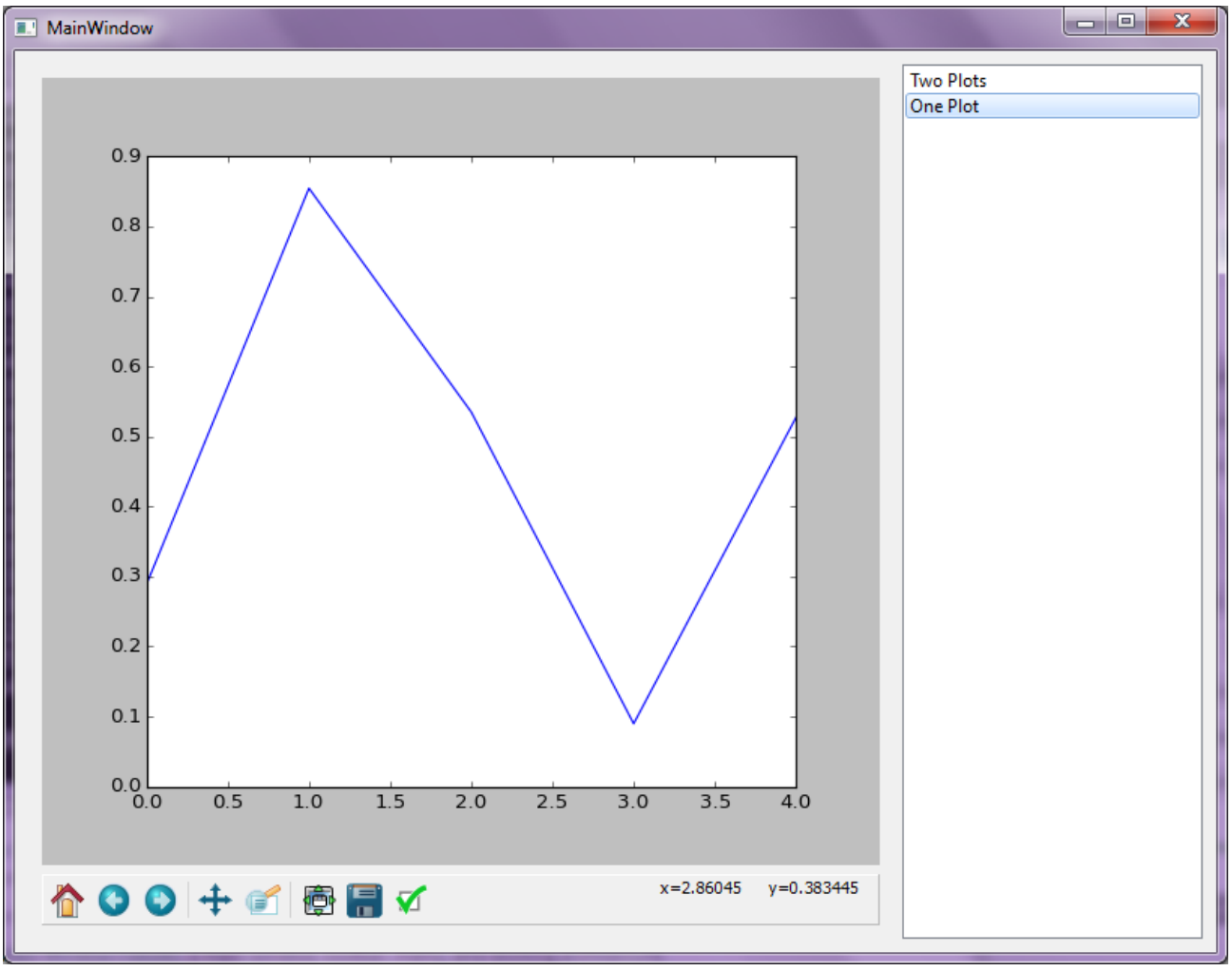
Building a Matplotlib GUI with Qt Designer: Part 1

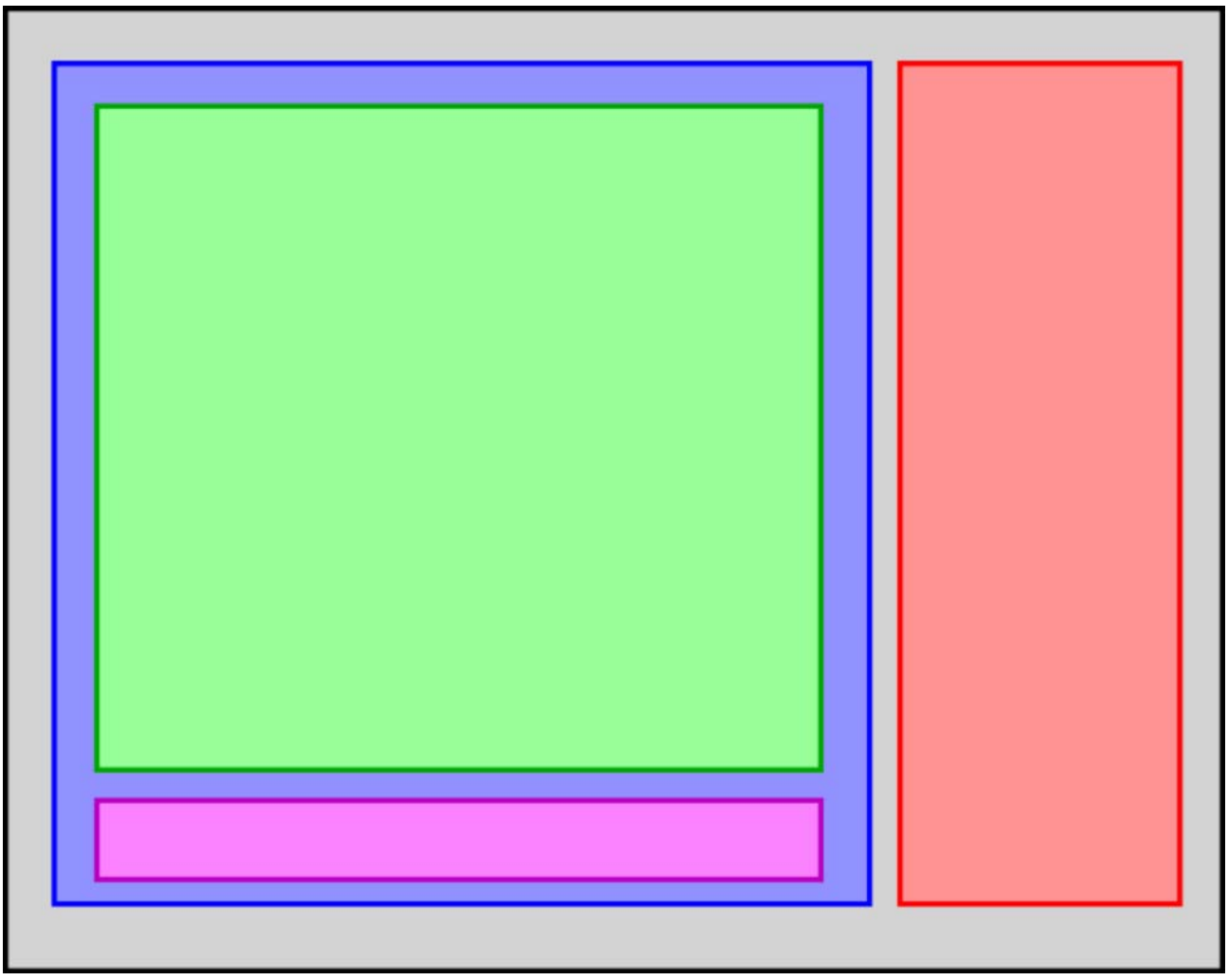
Designer is a graphical tool for building complex Qt4 GUI applications. In this post, I will use Designer to construct a simple GUI application, and in the following posts, I'll use Python, matplotlib, and PyQt4 to add the necessary application logic to display an interactive data plot and a plot selection list. The final result is shown below.



One thing to keep in mind, Designer's purpose is to help with GUI construction and widget layout. Application logic must be added in a separate step. That means that ultimately this example will differ from other Qt embedding examples in that we will end up with two files: the first, "window.ui", defines our overall GUI design, and the second, "custommpl.py", contains the custom application logic. A zip package of the completed files can be downloaded here.

Designing the Layout

A first step is to analyze the look of the GUI and decide how you will implement this with Designer. Although there are a number of ways to construct this layout, we will define a model that includes five major elements, with a generic graphical display shown below: 1) the main window holds everything together (black), 2) a list of figures (red), 3) a matplotlib container (blue), 4) a data plot (green), and 5) a navigation toolbar (purple).

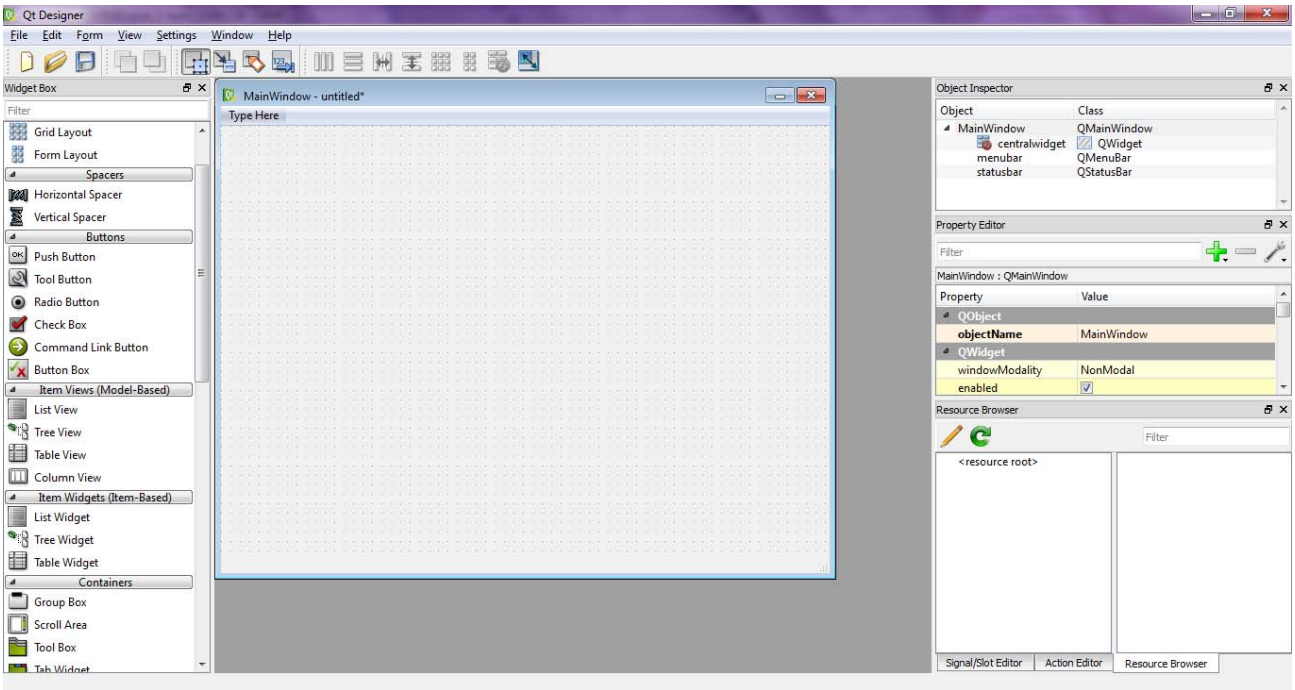


There are two major layouts to consider in our application. First, the matplotlib container and figure list are *horizontally aligned* inside of the main window, with the figure list taking up less overall space. Second, the data plot and toolbar are *vertically aligned* inside of the matplotlib container.

With this list of widgets and layouts in hand, let's try to create this application framework in Designer.

The Designer Window

Upon starting Designer, you will be presented with a creation dialog. Select "Main Window" and click "Create". You should now see something very close to the screen shot shown below. The "Main Window" that we've just created is going to act as an overall application container widget, in other words the black box in the generic figure above.



This Designer window has four major elements that will be important for this tutorial.

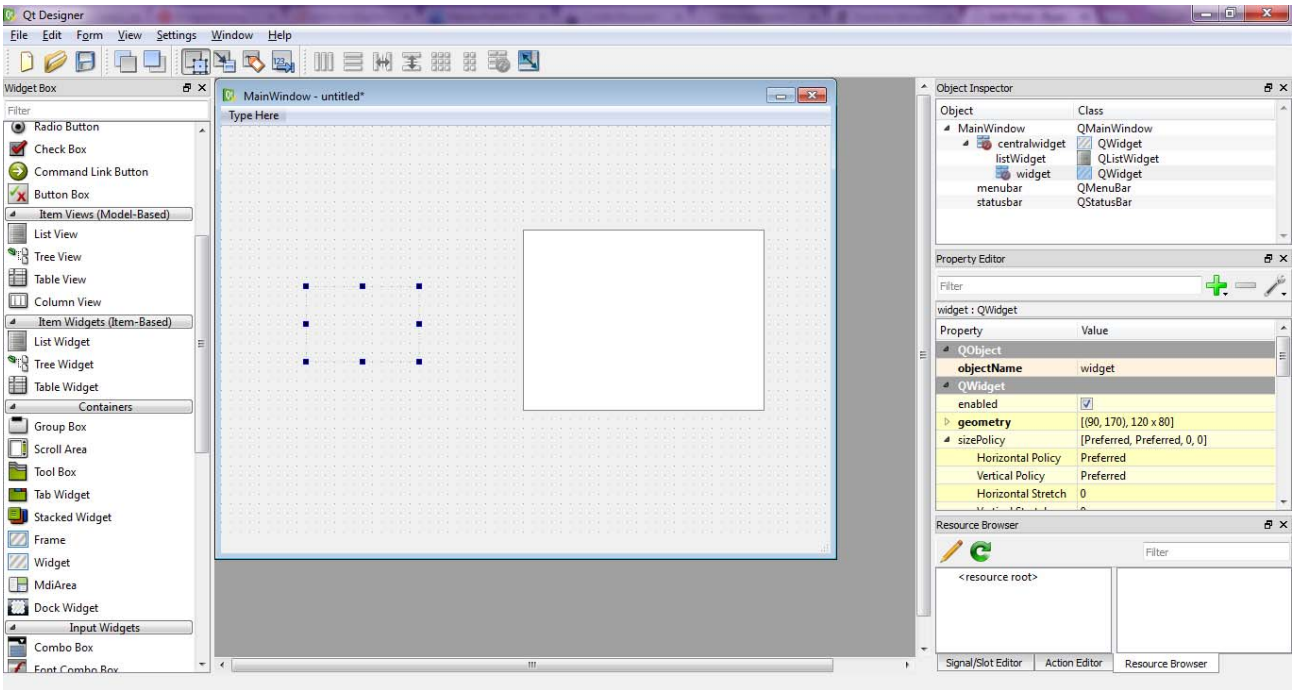
1. The blank, untitled "MainWindow" instance in the center of the screen is our application window. We will add widgets to this window to create our GUI layout. At this point, adjust the

size of this window until it is approximately the size that you'd like to see for your final application.

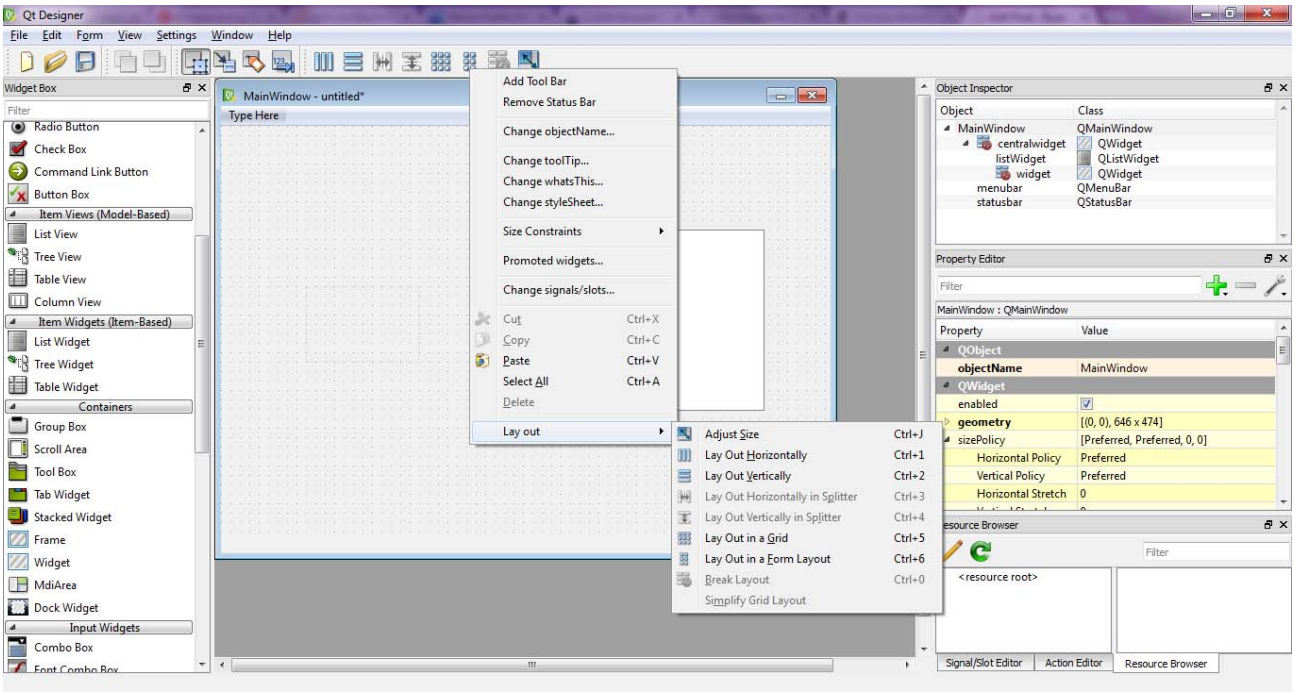
2. The "Widget Box" (left) contains all of the default Qt widgets that we can add to our MainWindow. To add a widget, click and drag the item into the MainWindow.
3. The "Object Inspector" (top right) presents a tree of all the widgets that are currently present in our MainWindow. Use this to select and modify individual widgets. The Object indentation level indicates which widgets are contained inside other widgets. For example, the MainWindow widget currently contains centralwidget, menubar, and statusbar widgets.
4. The "Property Editor", just below the "Object Inspector", is an editable list of properties for the currently selected widget.

Building the Framework

To start building our application, let's add the two widgets that should be contained in the MainWindow: the figure list and the matplotlib container. To do this, drag and drop a "List Widget" (under Item Widgets) to the right side of the MainWindow and a generic Widget (under Containers) to the left hand side. These widgets will have some default size, which will not look correct.



Manual adjustment of the widgets size and position is unnecessary. This is done automatically by applying the appropriate layout to the MainWindow container widget. Right click on the MainWindow and select “Lay Out”. You can select either “Lay Out Horizontally” or “Layout in a Grid”, both of which will have the same effect.



This stretches both widgets until they completely fill the window; unfortunately, they will not be the correct shape. Size adjustments are made by changing several widget properties. First of all, select the List Widget (either by clicking on it in the MainWindow or selecting it from the Object Inspector) and set the following properties in the Property Editor:

- sizePolicy->Horizontal Policy->Maximum
- maximumSize->Width->200

Next, select the matplotlib container widget and set this property:

- sizePolicy->Horizontal Policy->Expanding

At this point, our application window should look very much like our final product. However, we still need to setup the contents of our matplotlib container widget.

Adding the data plot and toolbar to the matplotlib container is best done as part of the application logic, but a vertical alignment should be enforced for the widgets contained in the matplotlib container. Unfortunately, a little hack is necessary to set the layout inside our matplotlib container. First of all, drag and drop a temporary widget (it doesn't matter what you choose) into the matplotlib container. Now right click on the matplotlib container and set the "Lay Out" to "Lay Out Vertically". With the layout set, you can delete the temporary widget from the plotting widget container.

Naming the Widgets

As a final step, we are going to change the names of some of the application elements. Ultimately, this application framework is going to be compiled into a Python class, and each of the widget and layout elements will be added as attributes of this class. The default attribute names are very generic "widget", "listWidget", etc., and renaming these widgets in Designer changes the attribute names as defined inside the main window class. This is not strictly necessary, but it will probably make it easier to keep track of the elements in a much larger GUI application. The widget names are shown in the "Object" column of the "Object Inspector". To change the names, simply double-click the name and type in the new one. For this application, make the following name changes:

- listWidget->mplfigs
- widget->mplwindow

The layouts are also named attributes of our application, and it will be useful to change at least the name of the vertical alignment layout of "mplwindow". Select the "mplwindow" widget in the Object Inspector, and scroll through the properties until you find the "Layout" section. Change the layoutName property from "verticalLayout" to "mplvl".

Saving the Application

With the application design complete, save this project as “window.ui”. UI files are language-agnostic XML representations of our GUI. If you need to make changes to your layout in the future, reopen the UI file with Designer and make the necessary changes. In the next post, we’ll go over how to programmatically extract out our MainWindow class from the UI file and create a subclass with the appropriate logic.

PyQt4 ships with a command line utility script called `pyuic4`, which converts UI files into Python module files. Below is an example of the command line invocation of this script.

```
1 $ pyuic window.ui > window.py
```

This is useful if you want to see how the MainWindow class and its attributes are defined in Python code; however, this step is unnecessary. *Be Warned!* Designer can not open Python module files, so you should never delete the UI file. Also, do not directly modify the resulting Python module. If you make layout changes to the UI file, the `pyuic4` conversion process will overwrite your modified window module. To add additional logic to your MainWindow class, write a separate module file that defines a subclass of the MainWindow object. (See the next post.)

February 19, 2015 Ryan Python GUI, Matplotlib, PyQt, Qt, Qt Designer