Object Oriented Programming II

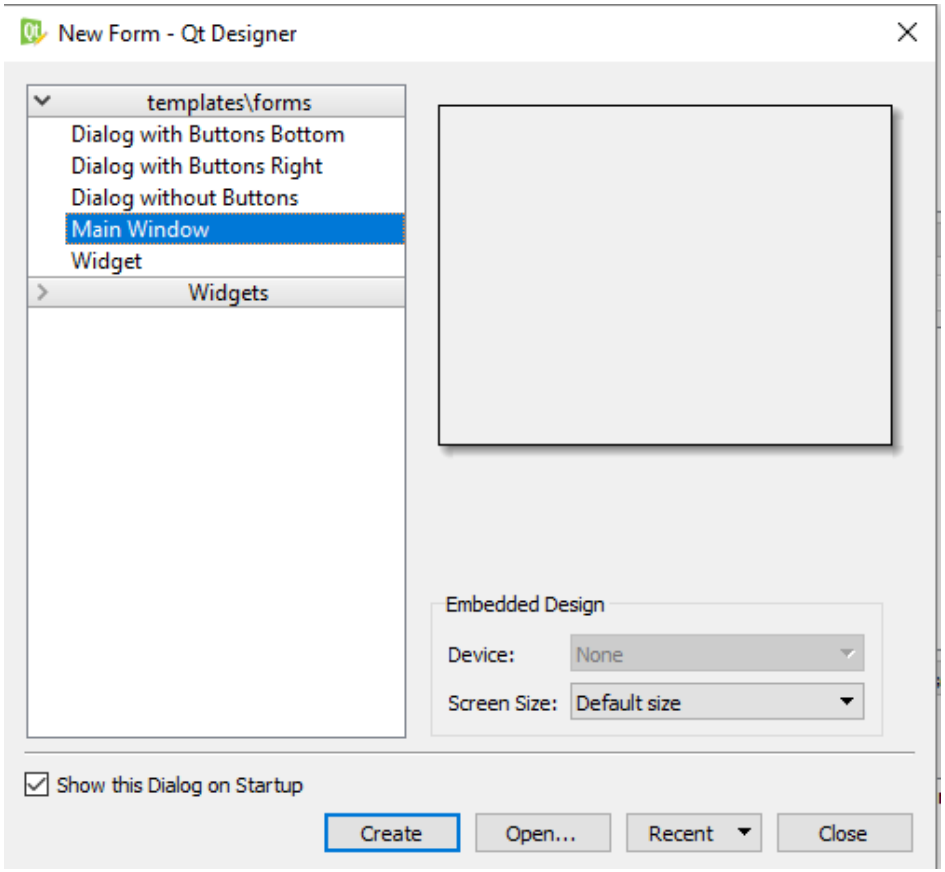
Dr. Burak Kaleci

April 5, 2019

Content

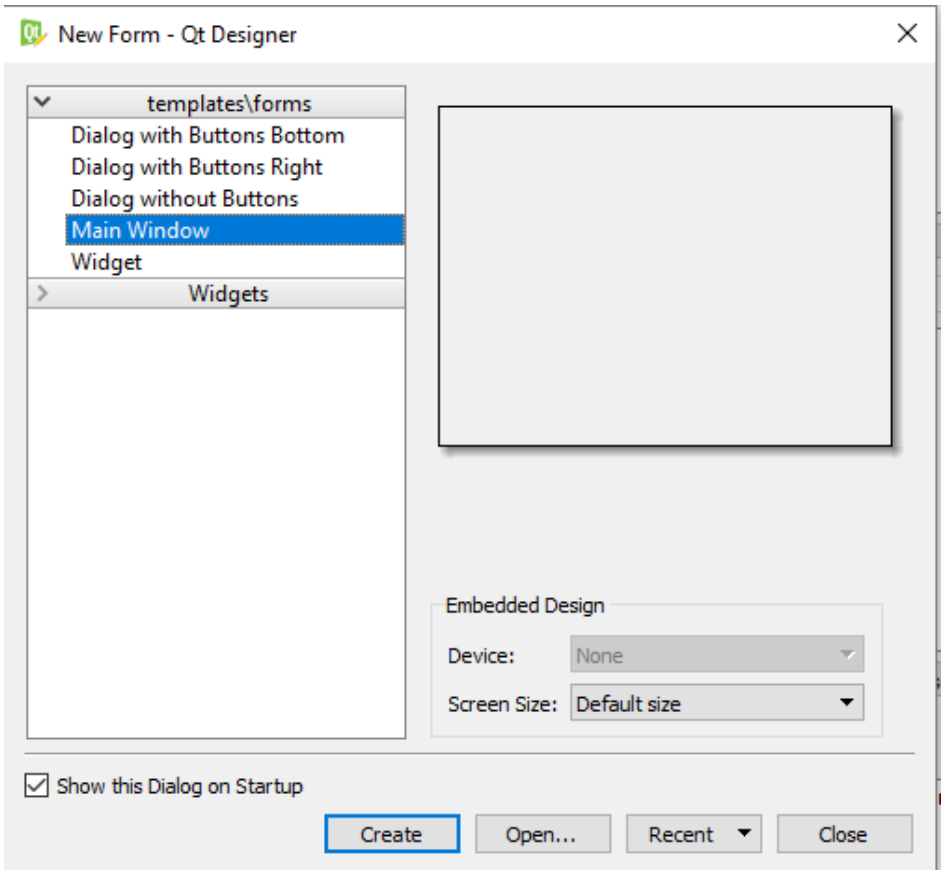
PyQt5 designer

Introduction



- Open designer.exe and you will see a dialog asking you about the form template you want.
- There are five templates available:
- Dialog with Buttons Bottom: Creates a form with OK and Cancel buttons at the bottom right of the form.
- Dialog with Buttons Right: Creates a form with OK and Cancel buttons at the top right of the form.
- Dialog without Buttons: Creates a blank form.
- Main Window: Creates a window with a menu bar and a toolbar and inherited from QMainWindow.

Introduction



- Widget: Creates a widget which is inherited from QWidget class, unlike the Dialogs templates which inherit from QDialog class.
- QWidget is the base class for all GUI elements in the PyQt5.
- QDialog is used for asking the user about something, like asking the user to accept or reject something or maybe asking for an input and is based on QWidget.
- QMainWindow is the bigger template where you can place your toolbar, menu bar, status bar, and other widget and it doesnt have a built-in allowance for buttons like those in QDialog.

Load .ui VS convert .ui to .py

- Open PyQt5 designer, and choose Main Window template and click create button.
- Then from the file menu, click save; PyQt5 designer will export your form into XML file with .ui extension.
- Now, in order to use this design, you have two ways:
  - Loading the .ui file in your Python code.
  - Converting the .ui file to a .py file using pyuic5.

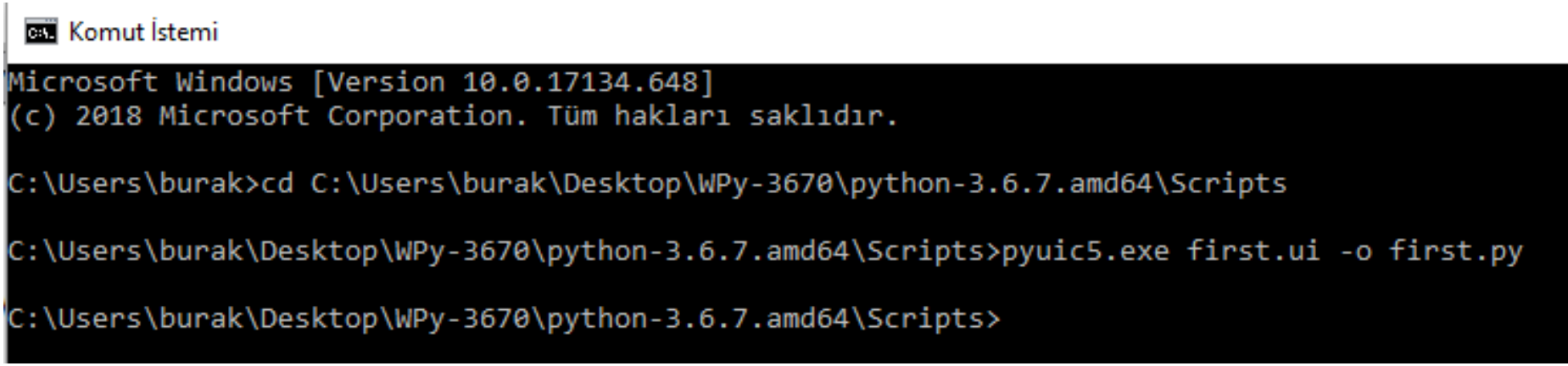
Loading the .ui file in your Python code

- To load the .ui file in your Python code, you can use the loadUI() function from uic like this:

```
1 from PyQt5 import QtWidgets, uic
2 import sys
3
4 app = QtWidgets.QApplication([])
5 win = uic.loadUi("first.ui") #specify the location of your .ui file
6 win.show()
7 sys.exit(app.exec())
```

Converting the .ui file to a .py file using pyuic5

- Now, lets try the second way by converting the .ui file to a Python code:



- Yes! A new file was created with the name first.py. Now, let's import that file to show our window.

Converting the .ui file to a .py file using pyuic5

- If you run this code, you should see the same window again as we did in the first method.

```
1 from PyQt5 import QtWidgets
2
3 from first import Ui_MainWindow  # importing our generated file
4
5 import sys
6
7 class mywindow(QtWidgets.QMainWindow):
8
9     def __init__(self):
10
11         super(mywindow, self).__init__()
12
13         self.ui = Ui_MainWindow()
14
15         self.ui.setupUi(self)
16
17 app = QtWidgets.QApplication([])
18
19 application = mywindow()
20
21 application.show()
22
23 sys.exit(app.exec())
```

Qt Designer's Editing Modes

- Qt Designer provides four editing modes:
  - In **Edit mode**, we can change the appearance of the form, add layouts, and edit the properties of each widget. To switch to this mode, press F3. This is Qt Designer's default mode.
  - In **Signals and Slots mode**, we can connect widgets together using Qt's signals and slots mechanism. To switch to this mode, press F4.
  - In **Buddy Editing Mode**, buddy widgets can be assigned to label widgets to help them handle keyboard focus correctly.
  - In **Tab Order Editing Mode**, we can set the order in which widgets receive the keyboard focus.

Edit Mode

- In the Widget Editing Mode, objects can be dragged from the main window's widget box to a form, edited, resized, dragged around on the form, and even dragged between forms.
- Object properties can be modified interactively, so that changes can be seen immediately.
- The editing interface is intuitive for simple operations, yet it still supports Qt's powerful layout facilities.
- Objects are added to the form by dragging them from the main widget box and dropping them in the desired location on the form.
- Once there, they can be moved around simply by dragging them, or using the cursor keys.
- Pressing the Ctrl key at the same time moves the selected widget pixel by pixel, while using the cursor keys alone make the selected widget snap to the grid when it is moved.

Edit Mode

- **Selecting Objects:** Objects on the form are selected by clicking on them with the left mouse button.
- When an object is selected, resize handles are shown at each corner and the midpoint of each side, indicating that it can be resized.
- When a widget is selected, normal clipboard operations such as cut, copy, and paste can be performed on it. All of these operations can be done and undone, as necessary.
- All of the above actions (apart from cloning) can be accessed via both the Edit menu and the form's context menu. These menus also provide functions for laying out objects as well as a Select All function to select all the objects on the form.

Edit Mode

- **Drag and Drop:** Qt Designer makes extensive use of the drag and drop facilities provided by Qt.
- Widgets can be dragged from the widget box and dropped onto the form.
- Widgets can also be "cloned" on the form: Holding down Ctrl and dragging the widget creates a copy of the widget that can be dragged to a new position.
- Qt Designer allows selections of objects to be copied, pasted, and dragged between forms. You can use this feature to create more than one copy of the same form, and experiment with different layouts in each of them.

Edit Mode

- **The Property Editor** always displays properties of the currently selected object on the form.
- The available properties depend on the object being edited, but all of the widgets provided have common properties such as objectName, the object's internal name, and enabled, the property that determines whether an object can be interacted with or not.
- **The Object Inspector** displays a hierarchical list of all the objects on the form that is currently being edited.
- To show the child objects of a container widget or a layout, click the handle next to the object label.
- Each object on a form can be selected by clicking on the corresponding item in the Object Inspector.

Signals and Slots Mode

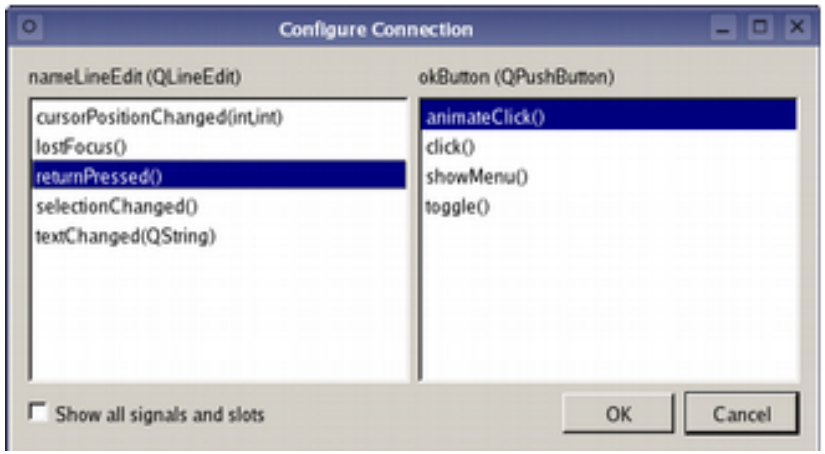
- In Qt Designer's signals and slots editing mode, you can connect objects in a form together using Qt's signals and slots mechanism.
- Both widgets and layouts can be connected via an intuitive connection interface, using the menu of compatible signals and slots provided by Qt Designer.
- When a form is saved, all connections are preserved so that they will be ready for use when your project is built.
- **Connecting Objects:**To begin connecting objects, enter the signals and slots editing mode by opening the Edit menu and selecting Edit Signals/Slots, or by pressing the F4 key.

Signals and Slots Mode

- To make a connection, press the left mouse button and drag the cursor towards the object you want to connect it to.
- As you do this, a line will extend from the source object to the cursor. If the cursor is over another object on the form, the line will end with an arrow head that points to the destination object.
- This indicates that a connection will be made between the two objects when you release the mouse button.
- You can abandon the connection at any point while you are dragging the connection path by pressing Esc.

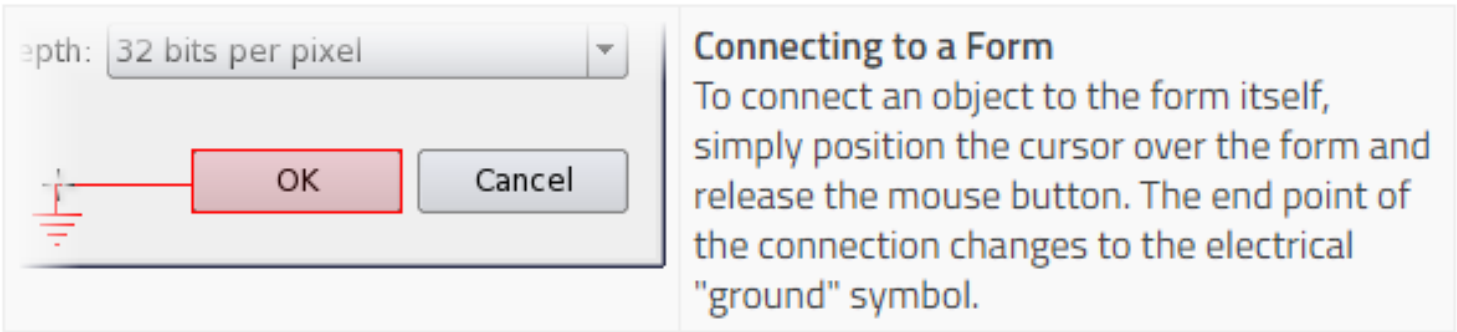
Signals and Slots Mode

- The **Configure Connection** dialog (below) is displayed, showing signals from the source object and slots from the destination object that you can use.
- To complete the connection, select a signal from the source object and a slot from the destination object, then click OK. Click Cancel if you wish to abandon the connection.
- If the Show all signals and slots checkbox is selected, all available signals from the source object will be shown. Otherwise, the signals and slots inherited from QWidget will be hidden.



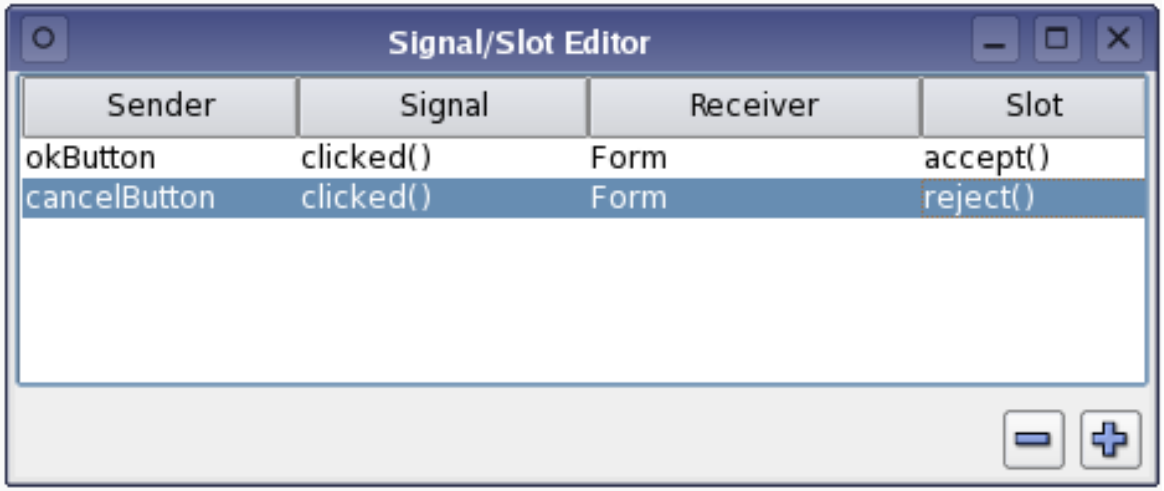
Signals and Slots Mode

- You can make as many connections as you like between objects on the form; it is possible to connect signals from objects to slots in the form itself. As a result, the signal and slot connections in many dialogs can be completely configured from within Qt Designer.



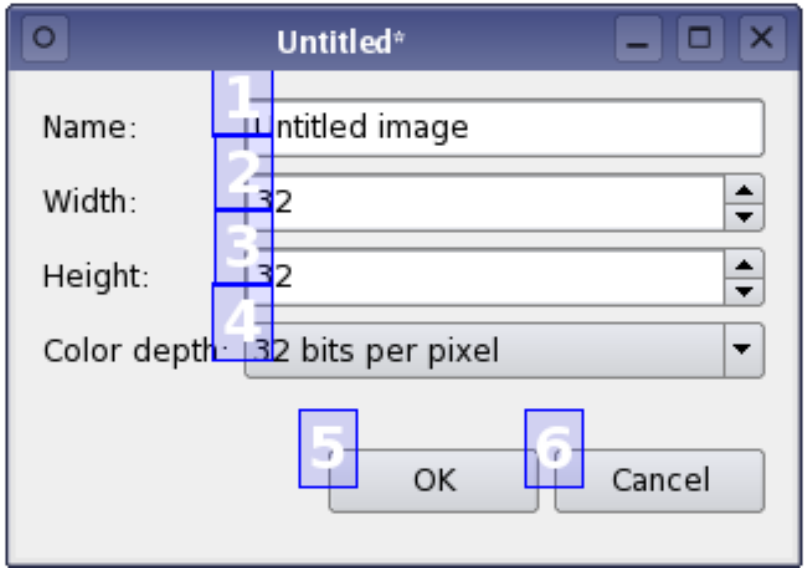
Signals and Slots Mode

- **The Signal/Slot Editor:** The signal and slot used in a connection can be changed after it has been set up.
- When a connection is configured, it becomes visible in Qt Designer's signal and slot editor where it can be further edited.
- You can also edit signal/slot connections by double-clicking on the connection path or one of its labels to display the Connection Dialog.



Tab Order Editing Mode

- Many users expect to be able to navigate between widgets and controls using only the keyboard.
- Qt lets the user navigate between input widgets with the Tab and Shift+Tab keyboard shortcuts.
- The default tab order is based on the order in which widgets are constructed.
- Although this order may be sufficient for many users, it is often better to explicitly specify the tab order to make your application easier to use.



Tab Order Editing Mode

- To enter tab order editing mode, open the Edit menu and select Edit Tab Order. In this mode, each input widget in the form is shown with a number indicating its position in the tab order.
- So, if the user gives the first input widget the input focus and then presses the tab key, the focus will move to the second input widget, and so on.
- The tab order is defined by clicking on each of the numbers in the correct order. The first number you click will change to red, indicating the currently edited position in the tab order chain. The widget associated with the number will become the first one in the tab order chain. Clicking on another widget will make it the second in the tab order, and so on.
- If you make a mistake, simply double click outside of any number or choose Restart from the form's context menu to start again.

Using Layouts in Qt Designer

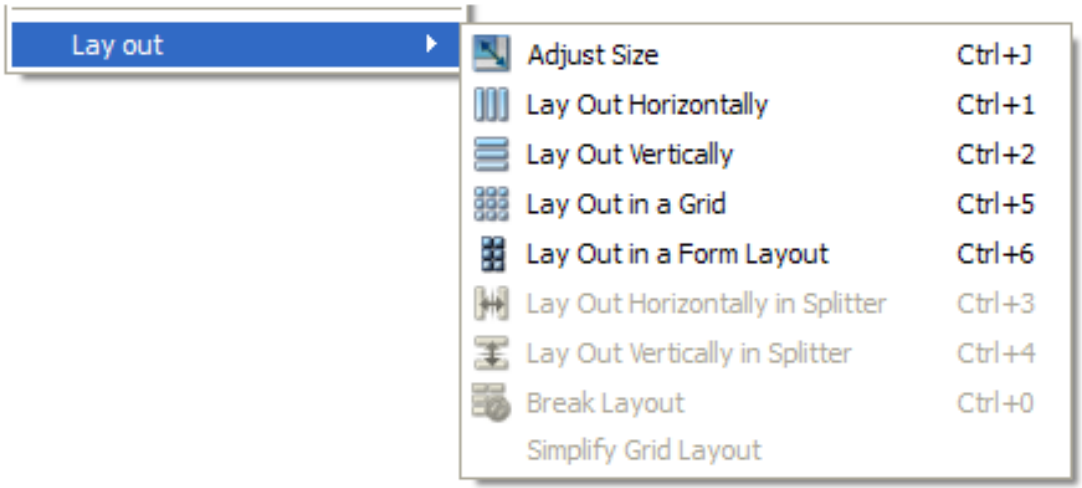
- Before a form can be used, the objects on the form need to be placed into layouts.
- This ensures that the objects will be displayed properly when the form is previewed or used in an application.
- Placing objects in a layout also ensures that they will be resized correctly when the form is resized.
- Once widgets have been inserted into a layout, it is not possible to move and resize them individually because the layout itself controls the geometry of each widget within it.
- Layouts can be nested to form a hierarchy. For example, to achieve a typical dialog layout with a horizontal row of buttons, the dialog elements can be laid out using a vertical box layout with a horizontal box layout containing the buttons at the bottom.
- To break a layout, press Ctrl+0 or choose Break Layout from the form's context menu, the Form menu or the main toolbar.

Using Layouts in Qt Designer

- The form's top level layout can be set by clearing the selection (click the left mouse button on the form itself) and applying a layout.
- A top level layout is necessary to ensure that your widgets will resize correctly when its window is resized.
- To check if you have set a top level layout, preview your widget and attempt to resize the window by dragging the size grip.
- Top level layouts are not visible as separate objects in the Object Inspector. Their properties appear below the widget properties of the main form, container widget, or page of a container widget in the Property Editor.

Using Layouts in Qt Designer

- To apply a layout, you can select your choice of layout from the toolbar shown on the left, or from the context menu shown below.

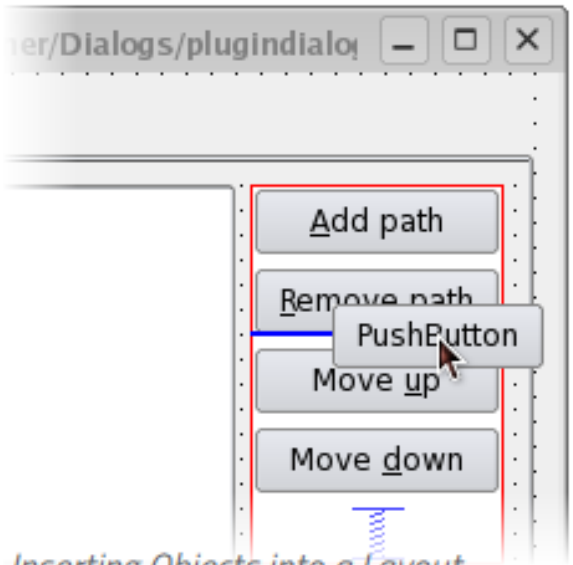


Using Layouts in Qt Designer

- Layout objects are created by applying a layout to a group of existing objects. This is achieved by selecting the objects that you need to manage and applying one of the standard layouts using the main toolbar, the Form menu, or the form's context menu.
- The layout object is indicated by a red frame on the form and appears as an object in the Object Inspector. Its properties (margins and constraints) are shown in the Property Editor.
- The layout object can be selected and placed within another layout along with other widgets and layout objects to build a layout hierarchy.

Using Layouts in Qt Designer

- Objects can be inserted into an existing layout by dragging them from their current positions and dropping them at the required location. A blue cursor is displayed in the layout as an object is dragged over it to indicate where the object will be added.



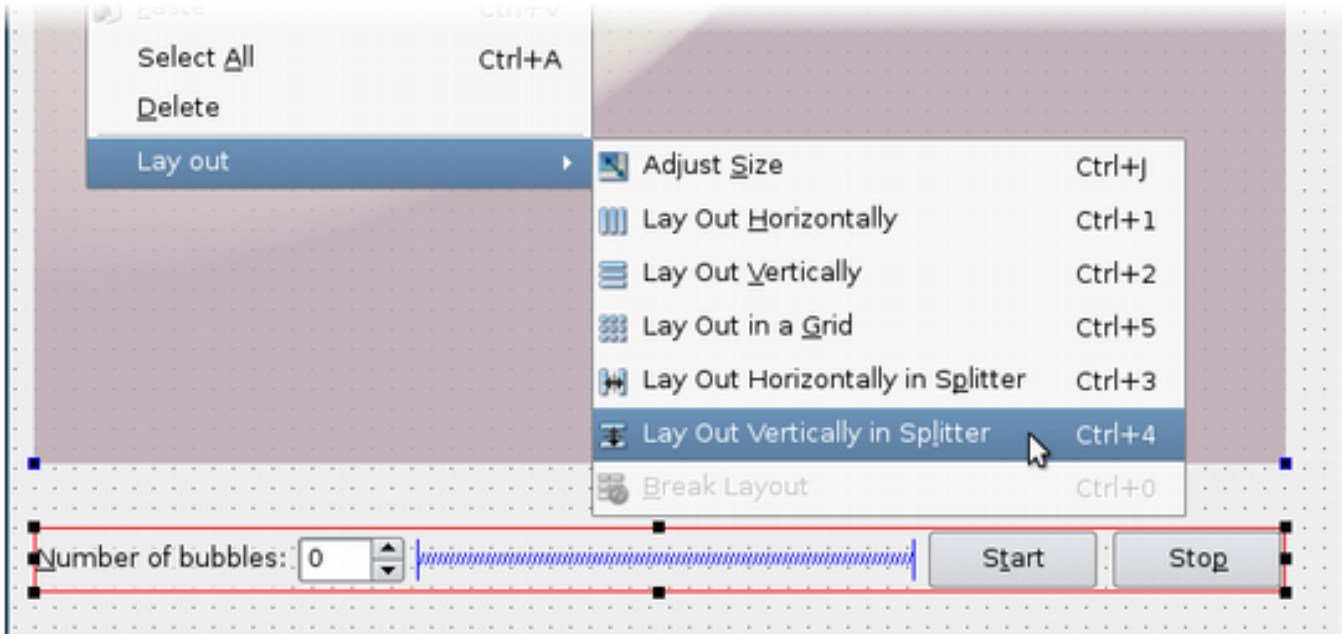
Inserting Objects into a Layout

Using Layouts in Qt Designer

- The simplest way to arrange objects on a form is to place them in a horizontal or vertical layout.
- Horizontal layouts ensure that the widgets within are aligned horizontally; vertical layouts ensure that they are aligned vertically.
- Horizontal and vertical layouts can be combined and nested to any depth. However, if you need more control over the placement of objects, consider using the grid layout.
- Complex form layouts can be created by placing objects in a grid layout.
- This kind of layout gives the form designer much more freedom to arrange widgets on the form, but can result in a much less flexible layout.
- However, for some kinds of form layout, a grid arrangement is much more suitable than a nested arrangement of horizontal and vertical layouts.

Using Layouts in Qt Designer

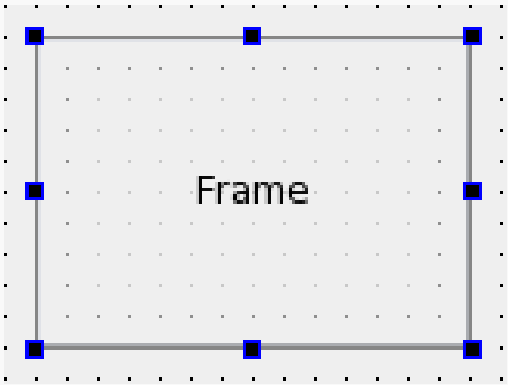
- Another common way to manage the layout of objects on a form is to place them in a splitter. These splitters arrange the objects horizontally or vertically in the same way as normal layouts, but also allow the user to adjust the amount of space allocated to each object.



Using Containers in Qt Designer

- Container widgets provide high level control over groups of objects on a form. They can be used to perform a variety of functions, such as managing input widgets, providing paged and tabbed layouts, or just acting as decorative containers for other objects.
- Stacked widgets, tab widgets, and toolboxes are handled specially in Qt Designer. Normally, when adding pages (tabs, pages, compartments) to these containers in your own code, you need to supply existing widgets, either as placeholders or containing child widgets.
- In Qt Designer, these are automatically created for you, so you can add child objects to each page straight away.
- Each container typically allows its child objects to be arranged in one or more layouts. The type of layout management provided depends on each container, although setting the layout is usually just a matter of selecting the container by clicking it, and applying a layout.

Using Containers in Qt Designer

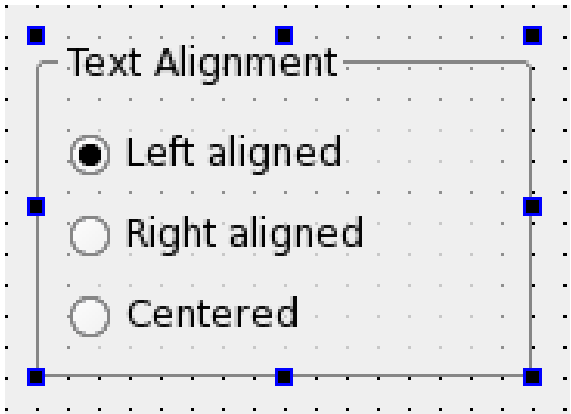


Frames

Frames are used to enclose and group widgets, as well as to provide decoration. They are used as the foundation for more complex containers, but they can also be used as placeholders in forms.

The most important properties of frames are `frameShape`, `frameShadow`, `lineWidth`, and `midLineWidth`. These are described in more detail in the QFrame class description.

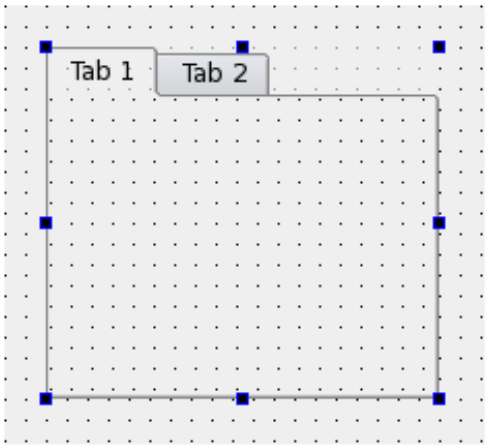
Using Containers in Qt Designer



Group Boxes

Group boxes are usually used to group together collections of checkboxes and radio buttons with similar purposes. Among the significant properties of group boxes are `title`, `flat`, `checkable`, and `checked`. These are demonstrated in the Group Box example, and described in the QGroupBox class documentation. Each group box can contain its own layout, and this is necessary if it contains other widgets. To add a layout to the group box, click inside it and apply the layout as usual.

Using Containers in Qt Designer



Tab Widgets

Tab widgets allow the developer to split up the contents of a widget into different labelled sections, only one of which is displayed at any given time. By default, the tab widget contains two tabs, and these can be deleted or renamed as required. You can also add additional tabs.

To delete a tab:

- Click on its label to make it the current tab.
- Select the tab widget and open its context menu.
- Select **Delete Page**.

To add a new tab:

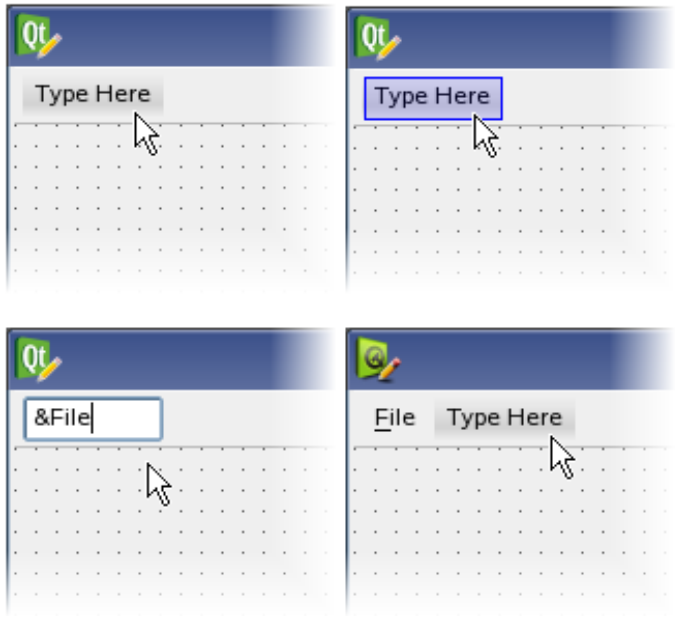
- Select the tab widget and open its context menu.
- Select **Insert Page**.
- You can add a page before or after the *current* page. *Qt Designer* will create a new widget for that particular tab and insert it into the tab widget.
- You can set the title of the current tab by changing the `currentTabText` property in the **Property Editor**.

Creating Main Windows in Qt Designer

- Qt Designer can be used to create user interfaces for different purposes, and it provides different kinds of form templates for each user interface.
- The main window template is used to create application windows with menu bars, toolbars, and dock widgets.
- Create a new main window by opening the File menu and selecting the New Form... option, or by pressing Ctrl+N. Then, select the Main Window template. This template provides a main application window containing a menu bar and a toolbar by default – these can be removed if they are not required.
- If you remove the menu bar, a new one can be created by selecting the Create Menu Bar option from the context menu, obtained by right-clicking within the main window form.
- An application can have only one menu bar, but several toolbars.

Creating Main Windows in Qt Designer

- Menus are added to the menu bar by modifying the Type Here placeholders. One of these is always present for editing purposes, and will not be displayed in the preview or in the finished window.
- Once created, the properties of a menu can be accessed using the Property Editor, and each menu can be accessed for this purpose via the The Object Inspector.
- Existing menus can be removed by opening a context menu over the label in the menu bar, and selecting Remove Menu 'menu_name'.

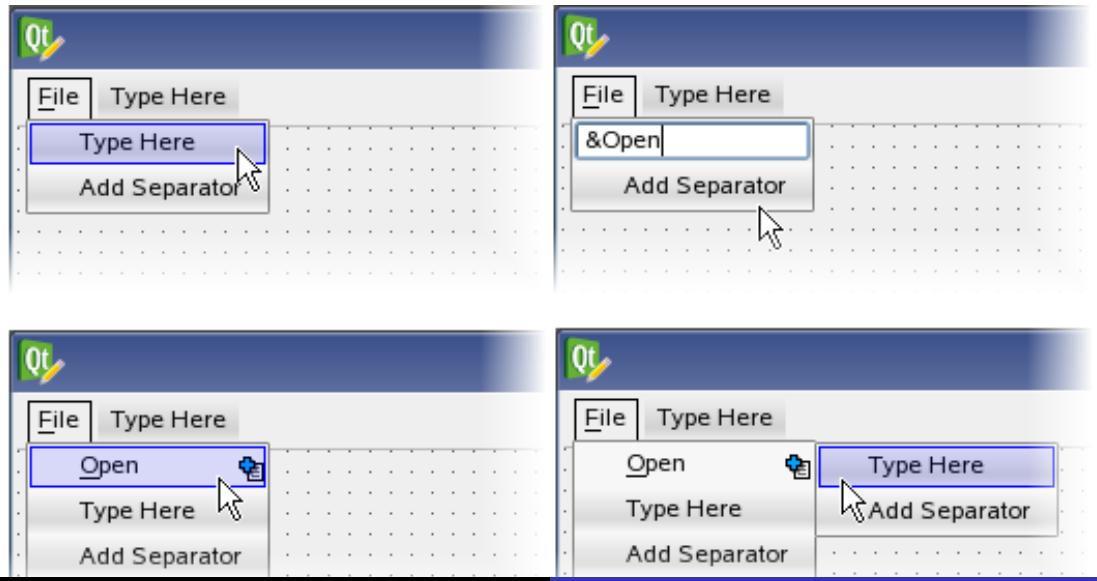


Creating Main Windows in Qt Designer

- Double-click the placeholder item to begin editing. The menu text, displayed using a line edit, can be modified.
- Insert the required text for the new menu. Inserting an ampersand character (&) causes the letter following it to be used as a mnemonic for the menu.
- Press Return or Enter to accept the new text, or press Escape to reject it. You can undo the editing operation later if required.

Creating Main Windows in Qt Designer

- Menus can also be rearranged in the menu bar simply by dragging and dropping them in the preferred location. A vertical red line indicates the position where the menu will be inserted.
- Menus can contain any number of entries and separators, and can be nested to the required depth. Adding new entries to menus can be achieved by navigating the menu structure in the usual way.

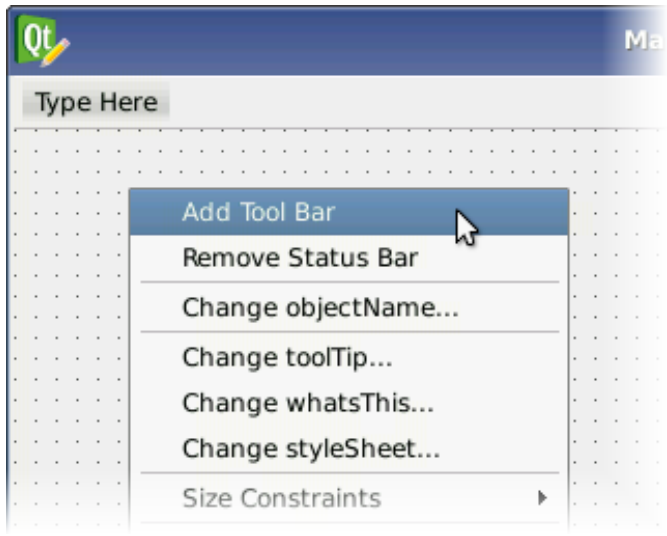


Creating Main Windows in Qt Designer

- Double-click the Type Here placeholder to begin editing, or double-click Add Separator to insert a new separator line after the last entry in the menu.
- The menu entry's text is displayed using a line edit, and can be modified.
- Insert the required text for the new entry, optionally using the ampersand character (&) to mark the letter to use as a mnemonic for the entry.
- Press Return or Enter to accept the new text, or press Escape to reject it.
- The action created for this menu entry will be accessible via the Action Editor, and any associated keyboard shortcut can be set there.
- Just like with menus, entries can be moved around simply by dragging and dropping them in the preferred location. When an entry is dragged over a closed menu, the menu will open to allow it to be inserted there. Since menu entries are based on actions, they can also be dropped onto toolbars, where they will be displayed as toolbar buttons.

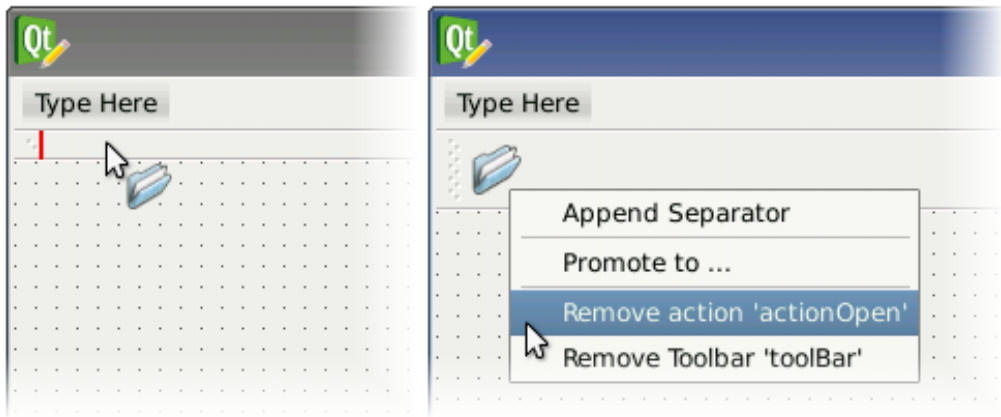
Creating Main Windows in Qt Designer

- Toolbars are added to a main window in a similar way to the menu bar: Select the Add Tool Bar option from the form's context menu.
- Alternatively, if there is an existing toolbar in the main window, you can click the arrow on its right end to create a new toolbar.
- Toolbars are removed from the form via an entry in the toolbar's context menu.



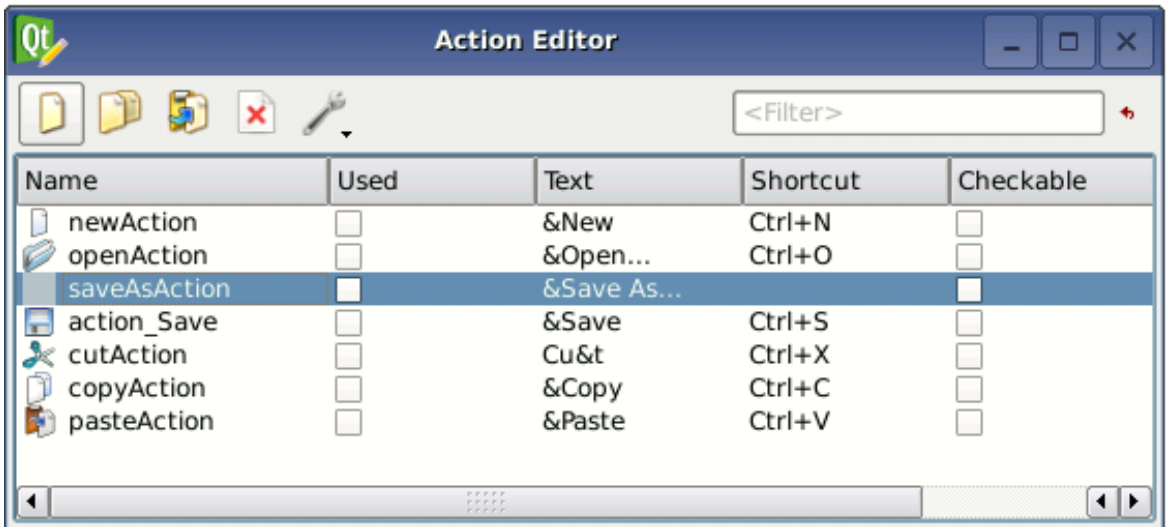
Creating Main Windows in Qt Designer

- Toolbar buttons are created as actions in the Action Editor and dragged onto the toolbar.
- Since actions can be represented by menu entries and toolbar buttons, they can be moved between menus and toolbars.
- To share an action between a menu and a toolbar, drag its icon from the action editor to the toolbar rather than from the menu where its entry is located.
- Toolbar buttons are removed via the toolbar's context menu.



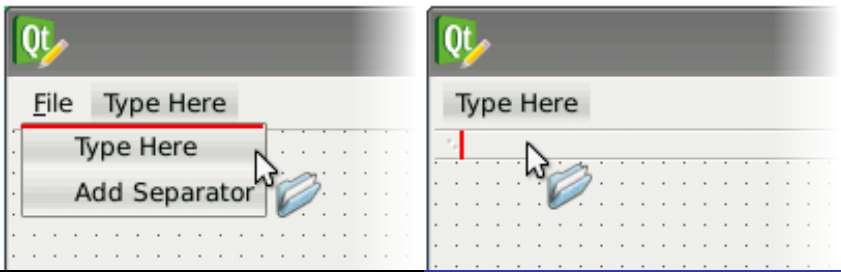
Creating Main Windows in Qt Designer

- With the menu bar and the toolbars in place, it's time to populate them with actions. New actions for both menus and toolbars are created in the action editor window, simplifying the creation and management of actions.
- Enable the action editor by opening the Tools menu, and switching on the Action Editor option.
- The action editor allows you to create New actions and Delete actions. It also provides a search function, Filter, using the action's text.



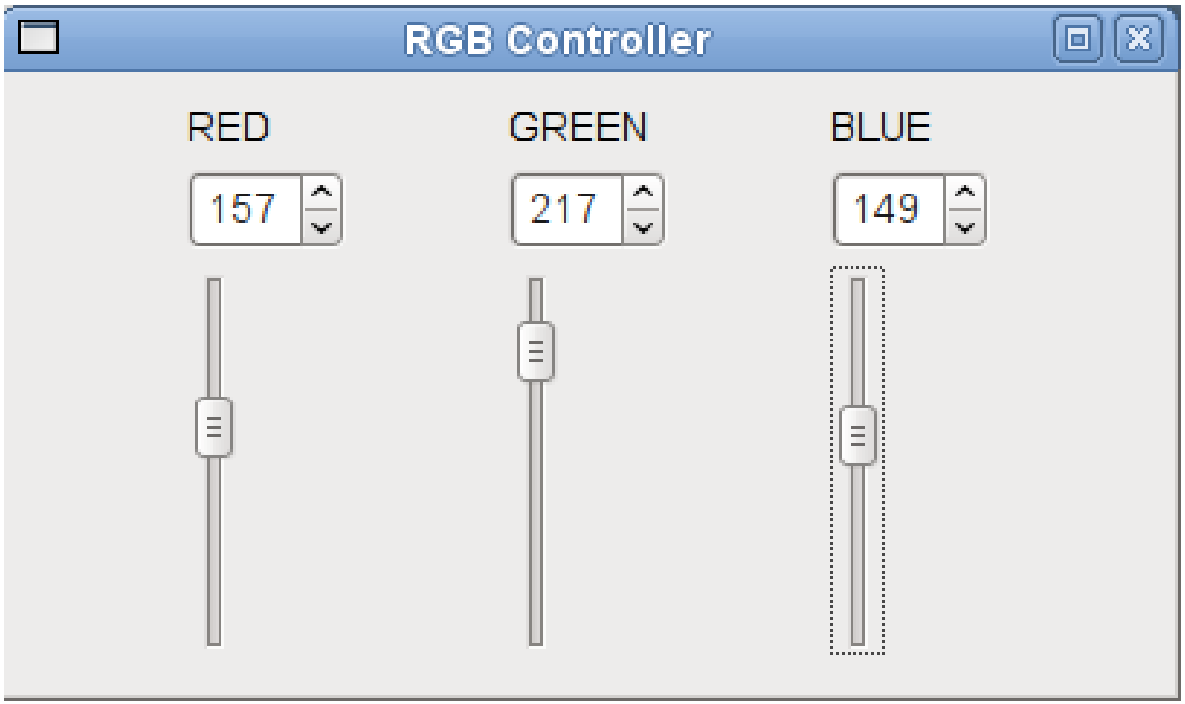
Creating Main Windows in Qt Designer

- To create an action, use the action editor's New button, which will then pop up an input dialog. Provide the new action with a Text – this is the text that will appear in a menu entry and as the action's tooltip. The text is also automatically added to an "action" prefix, creating the action's Object Name.
- In addition, the dialog provides the option of selecting an Icon for the action, as well as removing the current icon.
- Once the action is created, it can be used wherever actions are applicable.
- To add an action to a menu or a toolbar, simply press the left mouse button over the action in the action editor, and drag it to the preferred location.
- Qt Designer provides highlighted guide lines that tell you where the action will be added. Release the mouse button to add the action when you have found the right spot.



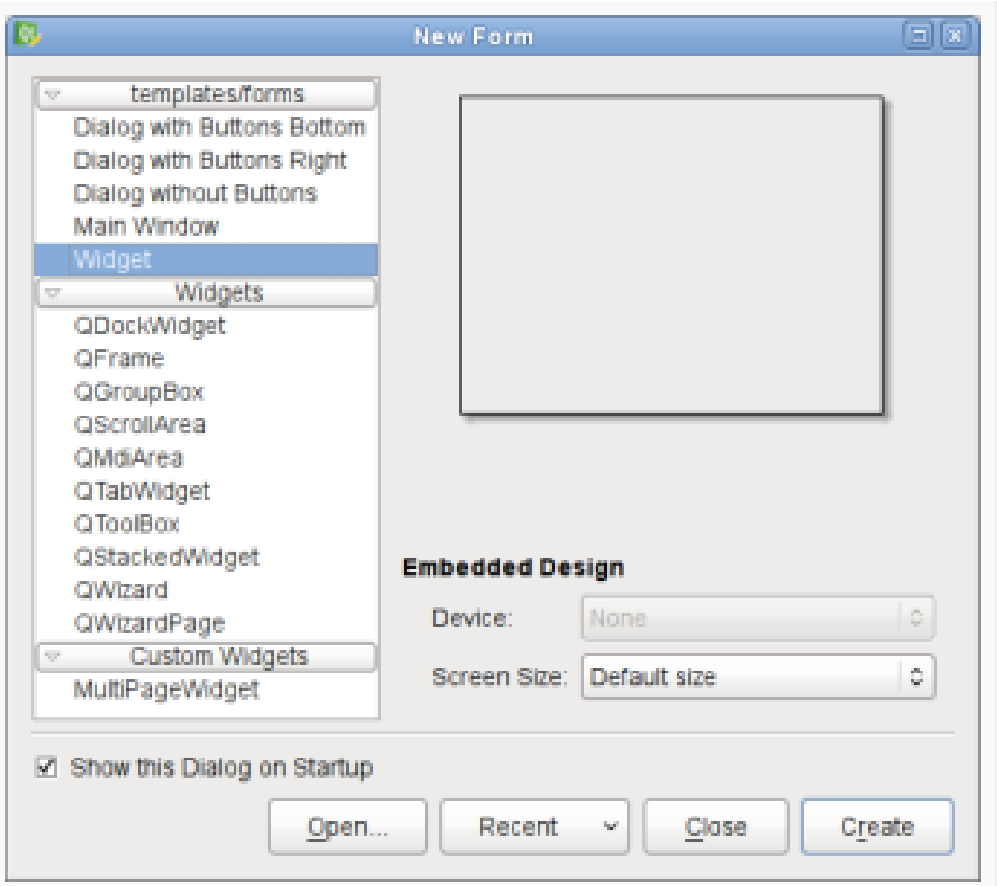
A Quick Start to Qt Designer

- Suppose you would like to design a small widget (see screenshot below) that contains the controls needed to manipulate Red, Green and Blue (RGB) values.

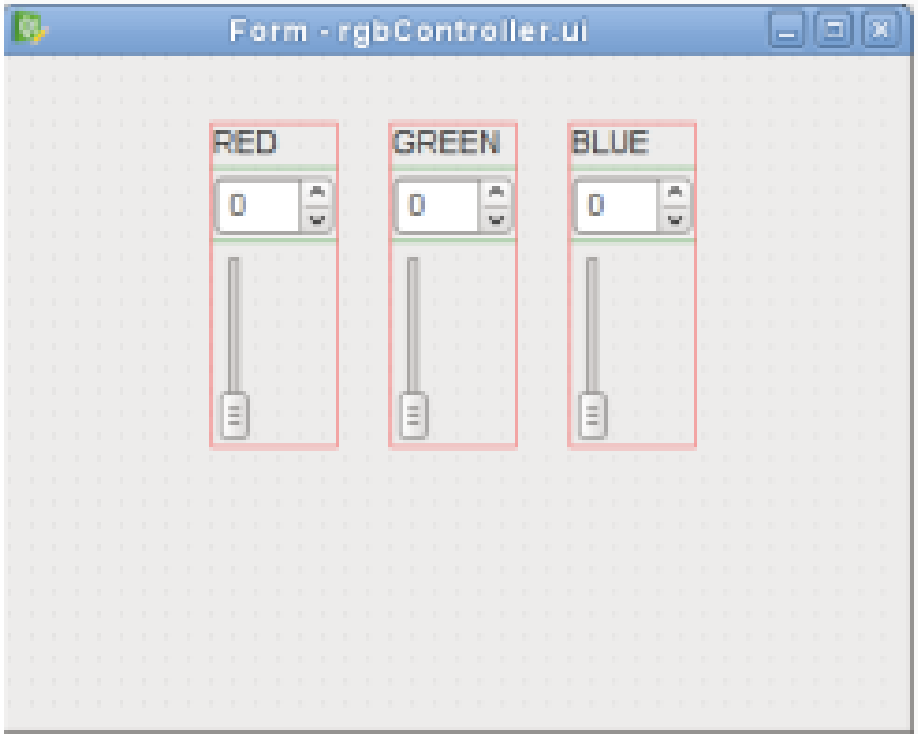


A Quick Start to Qt Designer

- **Choosing a Form:** You start by choosing Widget from the New Form dialog.



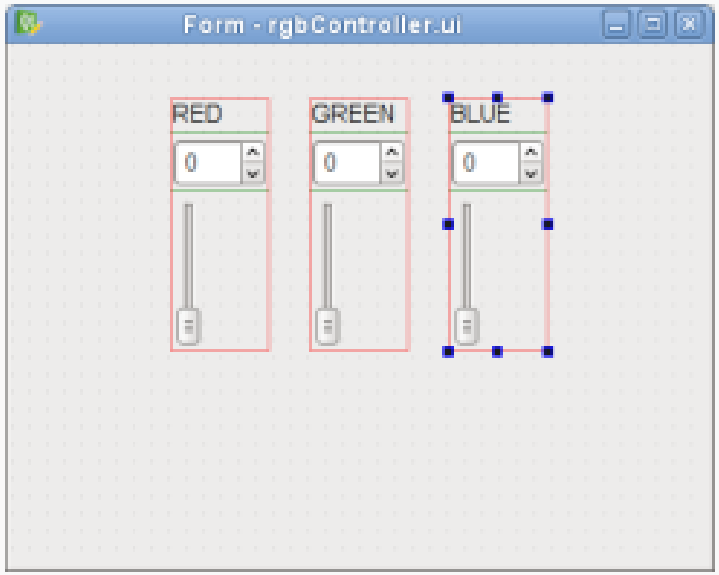
A Quick Start to Qt Designer

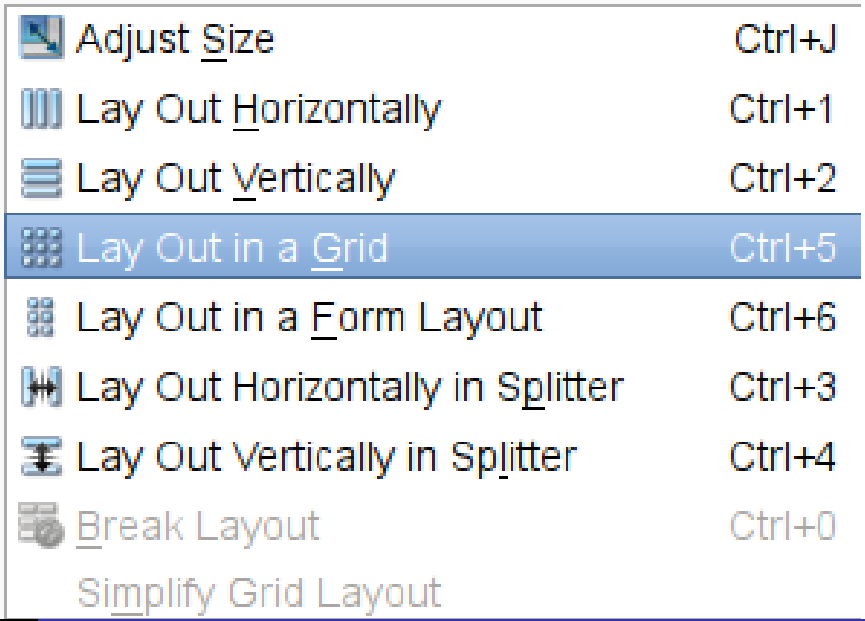


- **Placing Widgets on a Form:**Drag three labels, three spin boxes and three vertical sliders on to your form.
- To change the label's default text, simply double-click on it.
- You can arrange them according to how you would like them to be laid out.

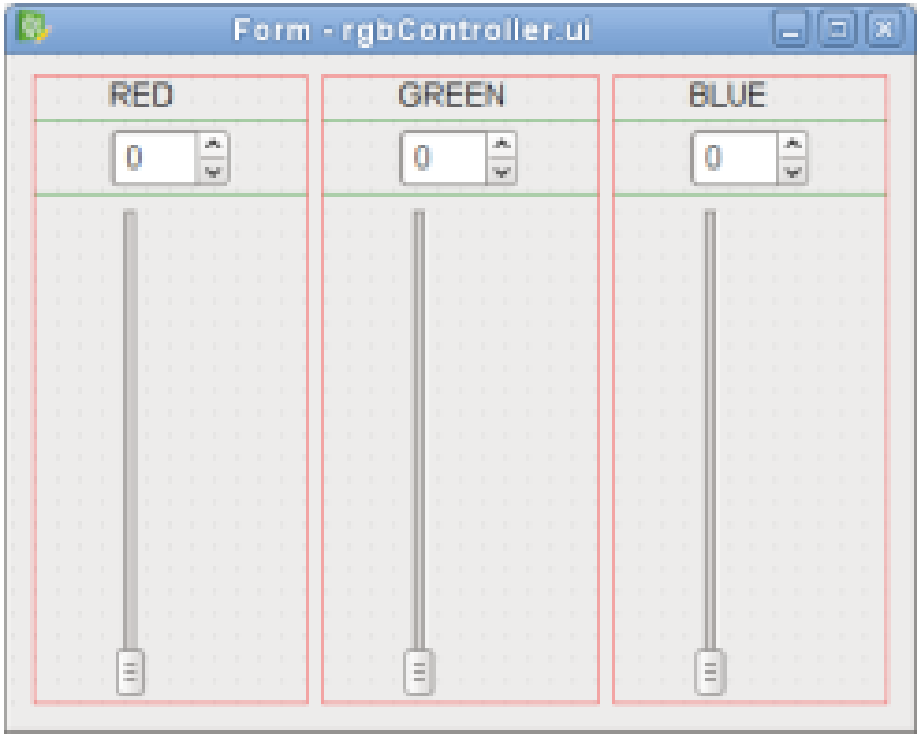
A Quick Start to Qt Designer

- To ensure that they are laid out exactly like this in your program, you need to place these widgets into a layout.
- We will do this in groups of three. Select the "RED" label. Then, hold down Ctrl while you select its corresponding spin box and slider. In the Form menu, select Lay Out in a Grid.
- Repeat the step for the other two labels along with their corresponding spin boxes and sliders as well.





A Quick Start to Qt Designer



- The next step is to combine all three layouts into one main layout.

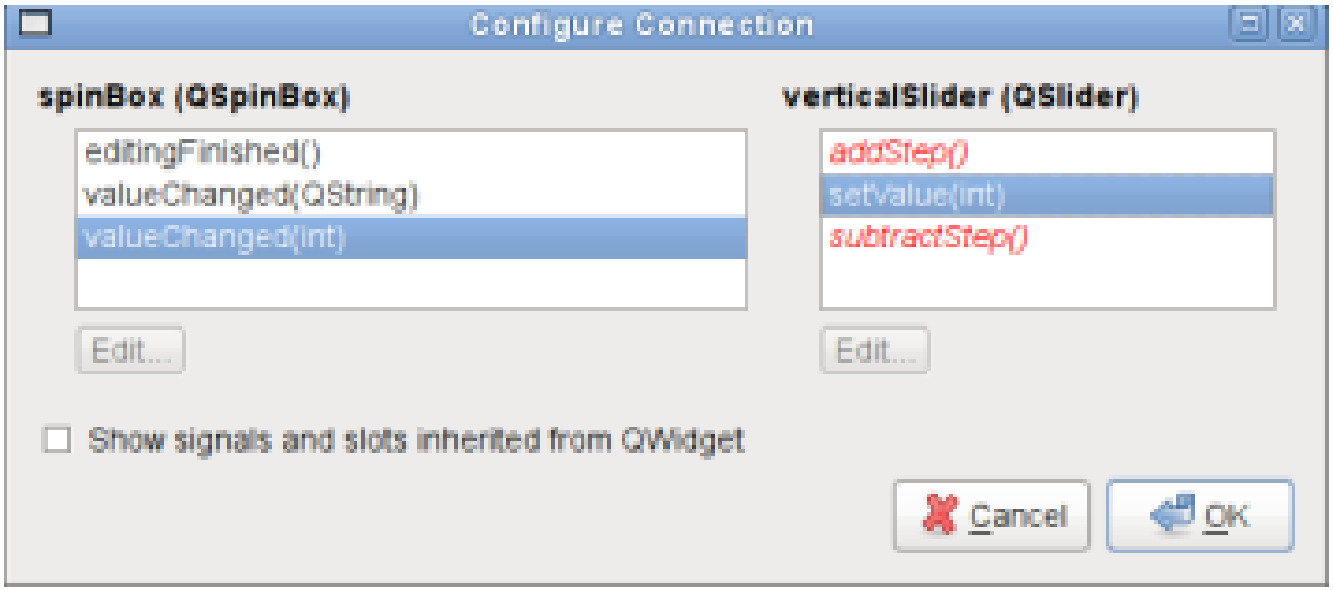
- The main layout is the top level widget's (in this case, the QWidget) layout.
- It is important that your top level widget has a layout; otherwise, the widgets on your window will not resize when your window is resized.
- To set the layout, Right click anywhere on your form, outside of the three separate layouts, and select Lay Out Horizontally.
- Alternatively, you could also select Lay Out in a Grid.

A Quick Start to Qt Designer

- When you click on the slider and drag it to a certain value, you want the spin box to display the slider's position.
- To accomplish this behavior, you need to connect the slider's valueChanged() signal to the spin box's setValue() slot.
- You also need to make the reverse connections, e.g., connect the spin box's valueChanged() signal to the slider's setValue() slot.
- To do this, you have to switch to Edit Signals/Slots mode, either by pressing F4 or selecting Edit Signals/Slots from the Edit menu.

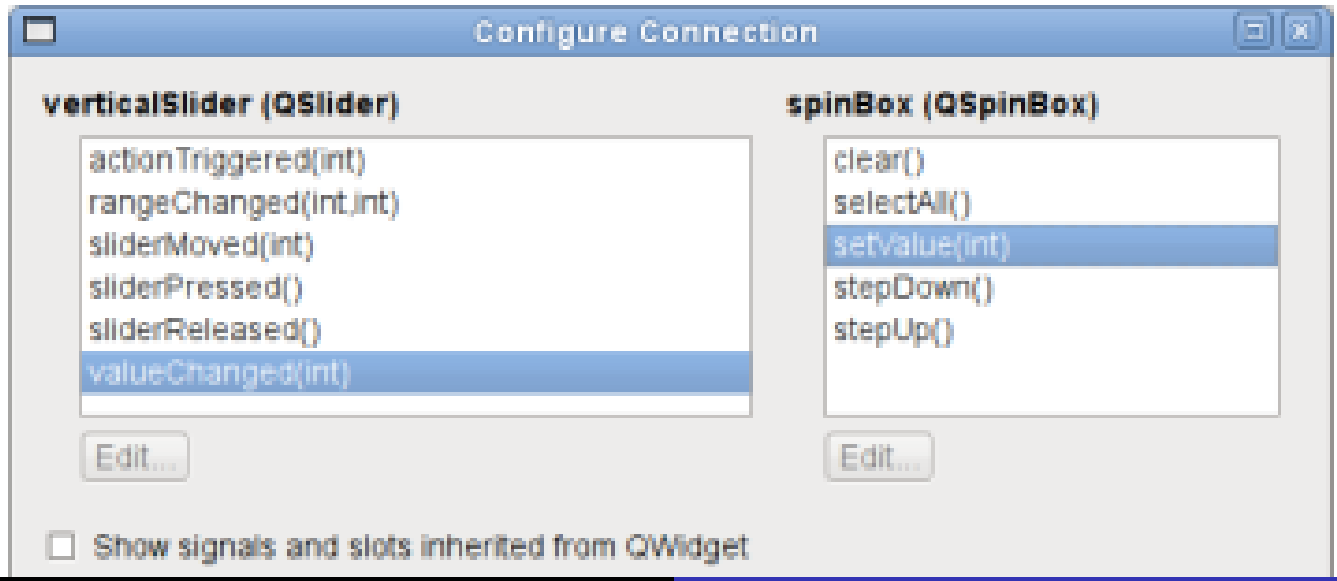
A Quick Start to Qt Designer

- Click on the slider and drag the cursor towards the spin box.
- The Configure Connection dialog, shown below, will pop up.
- Select the correct signal and slot and click OK.

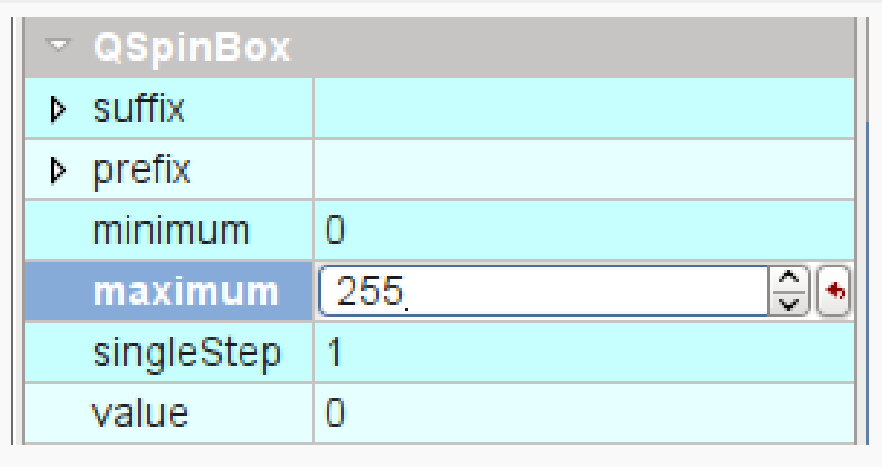


A Quick Start to Qt Designer

- Repeat the step (in reverse order), clicking on the spin box and dragging the cursor towards the slider, to connect the spin box's valueChanged() signal to the slider's setValue() slot.
- You can use the screenshot below as a guide to selecting the correct signal and slot.
- Now that you have successfully connected the objects for the "RED" component of the RGB Controller, do the same for the "GREEN" and "BLUE" components as well.



A Quick Start to Qt Designer



- Since RGB values range between 0 and 255, we need to limit the spin box and slider to that particular range.

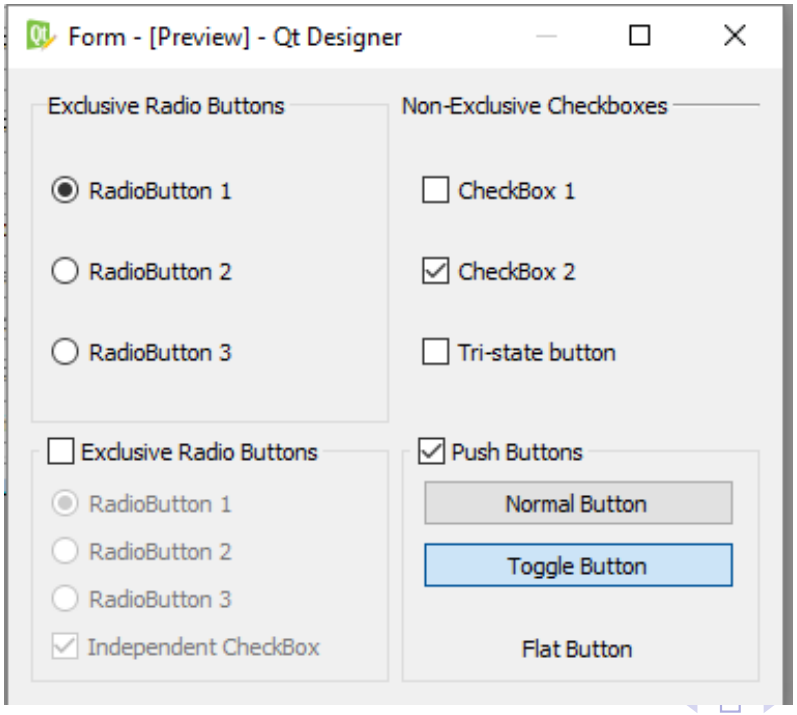
- **Setting Widget Properties:** Click on the first spin box. Within the Property Editor, you will see QSpinBox's properties. Enter "255" for the maximum property.
- Then, click on the first vertical slider, you will see QAbstractSlider's properties. Enter "255" for the maximum property as well.
- Repeat this process for the remaining spin boxes and sliders.

A Quick Start to Qt Designer

- Now, we preview your form to see how it would look in your application - press Ctrl + R or select Preview from the Form menu.
- Try dragging the slider - the spin box will mirror its value too (and vice versa).
- Also, you can resize it to see how the layouts that are used to manage the child widgets, respond to different window sizes.

Group Box Example

- The Group Box example shows how to use the different kinds of group boxes in Qt.
- Group boxes are container widgets that organize buttons into groups, both logically and on screen. They manage the interactions between the user and the application so that you do not have to enforce simple constraints.
- Group boxes are usually used to organize check boxes and radio buttons into exclusive groups.

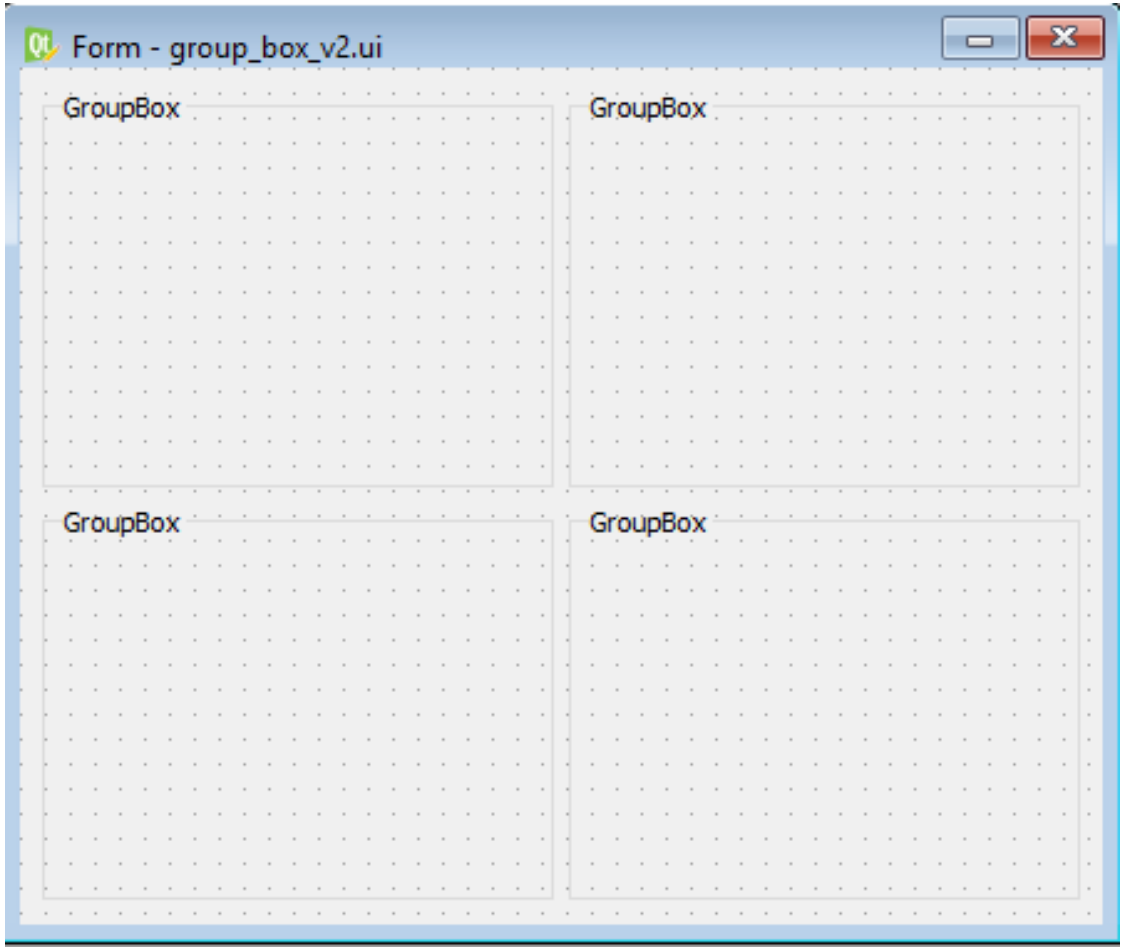


Group Box Example

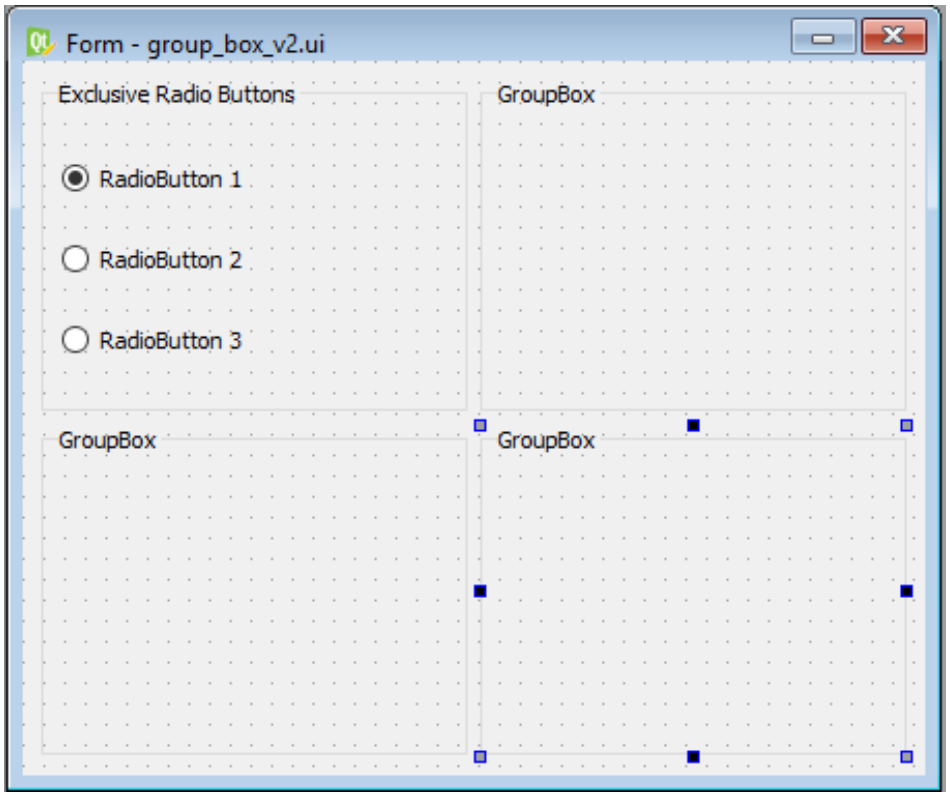
- Firstly, include four group boxes:
  - an exclusive radio button group,
  - a non-exclusive checkbox group,
  - an exclusive radio button group with an enabling checkbox,
  - and a group box with normal push buttons.
- Then, create a grid layout and fills it with each of the group boxes that are to be displayed.

Group Box Example

- Then, rename group boxes' names according to given figure.

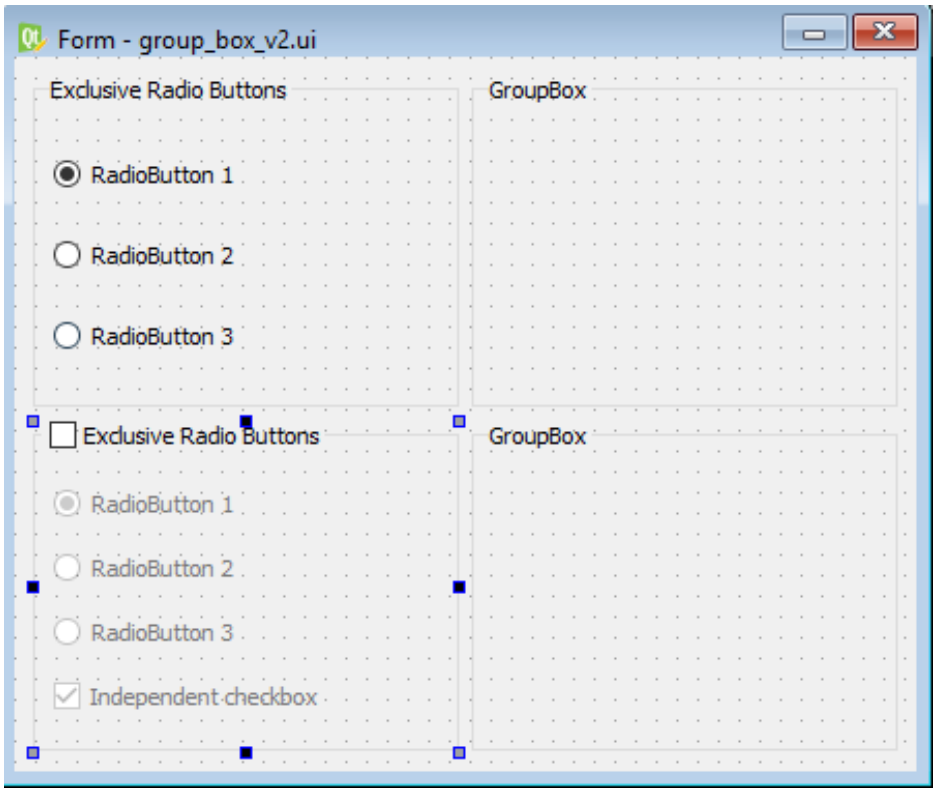


Group Box Example



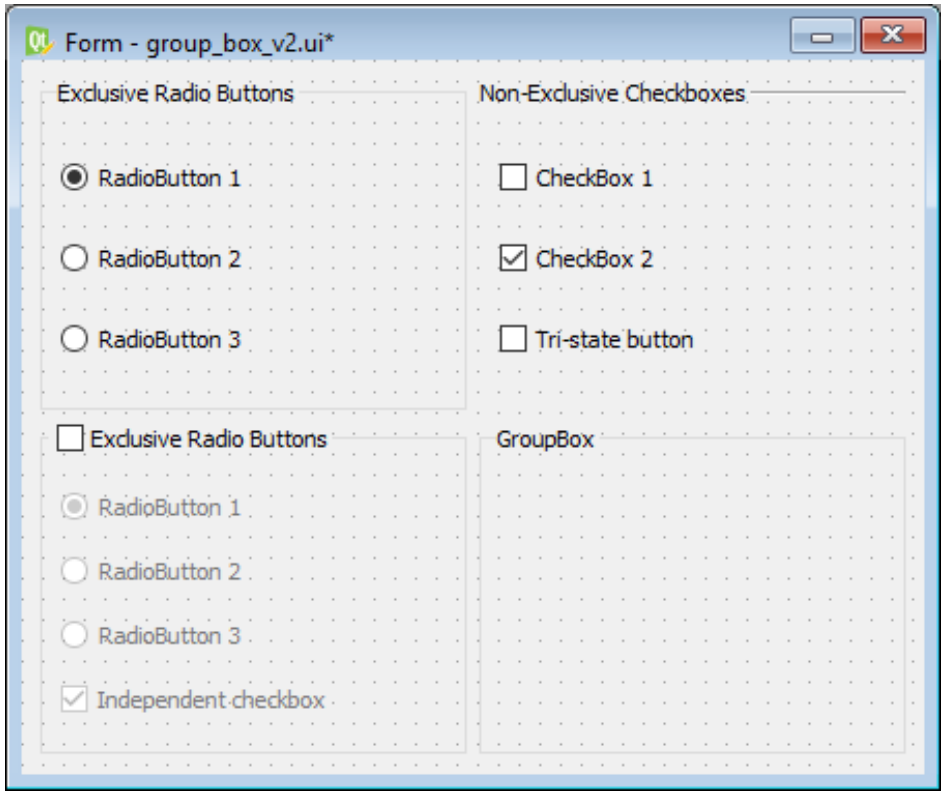
- The first group box contains and manages three radio buttons. Since the group box contains only radio buttons, it is exclusive by default, so only one radio button can be checked at any given time. We check the first radio button to ensure that the button group contains one checked button.
- We use a vertical layout within the group box to present the buttons in the form of a vertical list,

Group Box Example



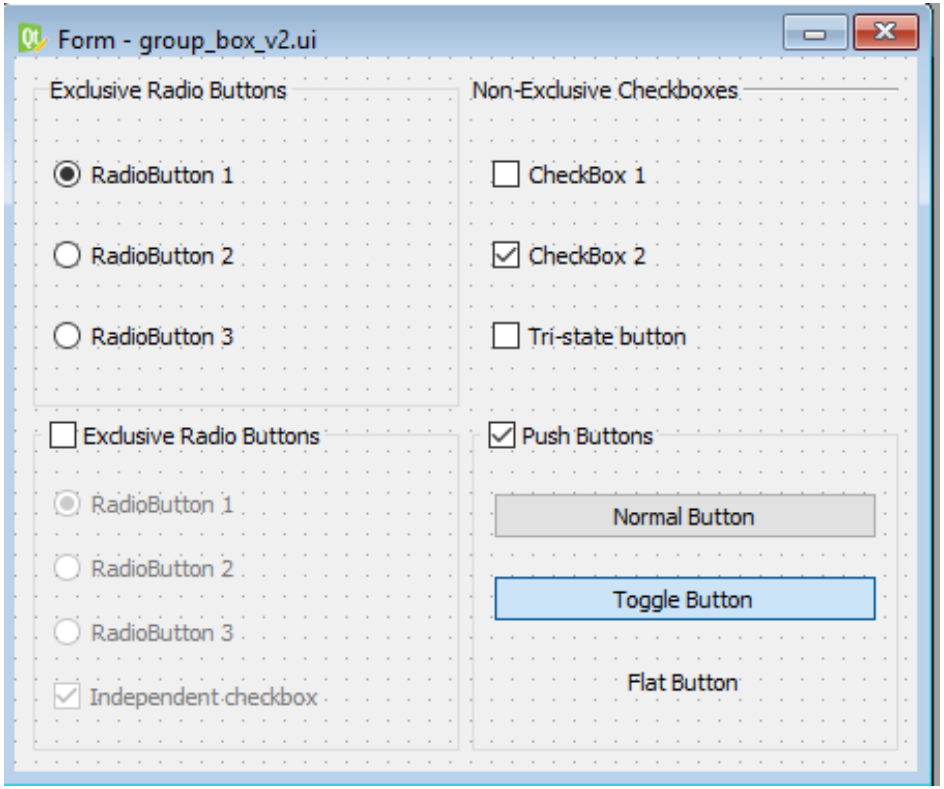
- The second group box is itself checkable, providing a convenient way to disable all the buttons inside it.
- Initially, it is unchecked, so the group box itself must be checked before any of the radio buttons inside can be checked.
- The group box contains three exclusive radio buttons, and an independent checkbox. For consistency, one radio button must be checked at all times, so we ensure that the first one is initially checked.

Group Box Example



- The third group box is constructed with a "flat" style that is better suited to certain types of dialog.
- This group box contains only checkboxes, so it is non-exclusive by default. This means that each checkbox can be checked independently of the others.
- Again, we use a vertical layout within the group box to present the buttons in the form of a vertical list.

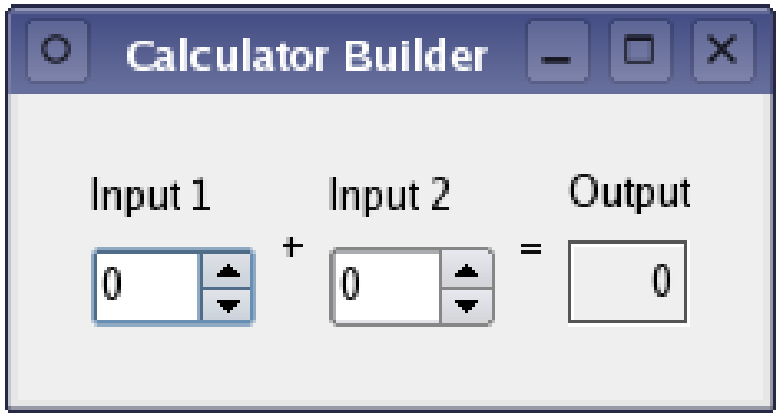
Group Box Example



- The final group box contains only push buttons and, like the second group box, it is checkable.
- We create a normal button, a toggle button, and a flat push button.
- Finally, we lay out the widgets vertically, and return the group box that we created

Sum Example

- Create a user interface from a Qt Designer form at run-time.



- Convert .ui file to .py file

Komut İstemi

```
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\burak>cd C:\Users\burak\Desktop\WPy-3670\python-3.6.7.amd64\Scripts

C:\Users\burak\Desktop\WPy-3670\python-3.6.7.amd64\Scripts>pyuic5.exe sum.ui -o sum.py

C:\Users\burak\Desktop\WPy-3670\python-3.6.7.amd64\Scripts>
```

Sum Example

```
1  from PyQt5 import QtWidgets
2
3  from sum import Ui_Form  # importing our generated file
4
5  import sys
6
7  class mywindow(QtWidgets.QWidget):
8
9      def __init__(self):
10
11         super(mywindow, self).__init__()
12
13         self.ui = Ui_Form()
14
15         self.ui.setupUi(self)
16
17 app = QtWidgets.QApplication([])
18
19 application = mywindow()
20
21 application.show()
22
23 sys.exit(app.exec())
```

Sum Example

```
        self.spinBox = QtWidgets.QSpinBox(self.widget)
        self.spinBox.setObjectName("spinBox")
        self.spinBox.valueChanged.connect(self.valuechange)
        self.horizontalLayout.addWidget(self.spinBox)

        self.label_5 = QtWidgets.QLabel(self.widget)
        self.label_5.setObjectName("label_5")
        self.horizontalLayout.addWidget(self.label_5)
        self.spinBox_2 = QtWidgets.QSpinBox(self.widget)
        self.spinBox_2.setObjectName("spinBox_2")
        self.spinBox_2.valueChanged.connect(self.valuechange)
        self.horizontalLayout.addWidget(self.spinBox_2)
def valuechange(self):
    self.label_4.setText(str(self.spinBox.value() + self.spinBox_2.value()) )
```