

Build GUI using Qt Designer

WEEK 5

Instructor: Ph.D. Oleg Tymchuk

Overview





Qt Designer's Editing Modes

Creating Main Windows in Qt Designer

Using a Designer UI File in Application

Qt Designer's Editing Modes

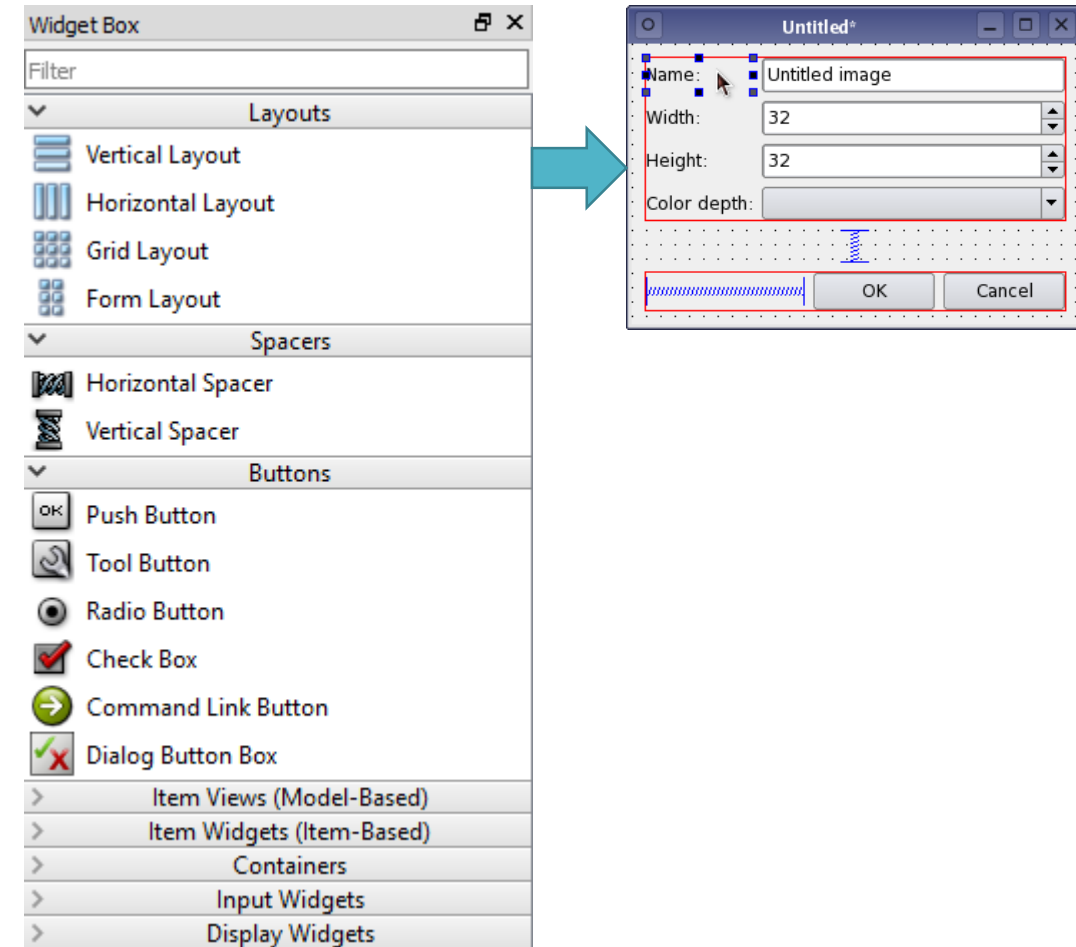
- Qt Designer is the Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets.
- Qt Designer provides four editing modes: Widget Editing Mode, Signals and Slots Editing Mode, Buddy Editing Mode and Tab Order Editing Mode.

	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.

Qt Designer's Editing Modes.

Widget Editing Mode

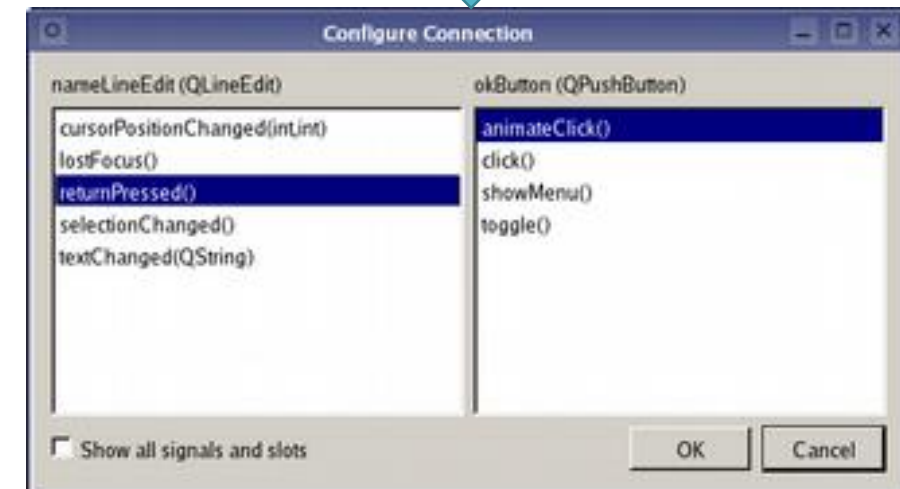
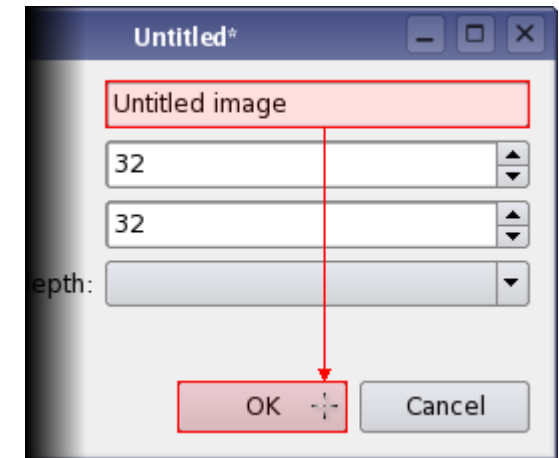
- In the Widget Editing Mode, objects can be dragged from the main window's widget box to a form, edited, resized, dragged around on the form, and even dragged between forms
- Object properties can be modified interactively, so that changes can be seen immediately
- To create and edit new forms, open the File menu and select New Form... or press Ctrl+N. Existing forms can also be edited by selecting Open Form... from the File menu or pressing Ctrl+O
- By default, new forms are opened in widget editing mode. To switch to Edit mode from another mode, select Edit Widgets from the Edit menu or press the F3 key



Qt Designer's Editing Modes.

Signals and Slots Editing Mode

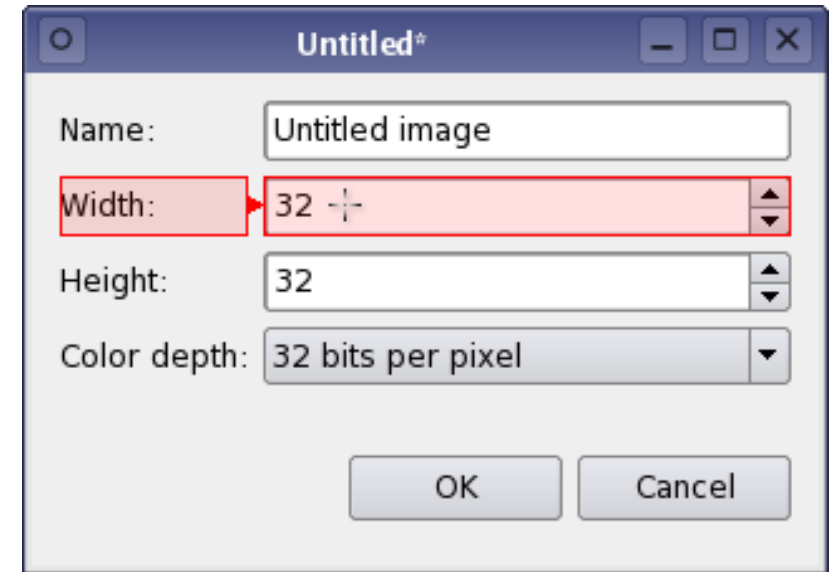
- In Qt Designer's signals and slots editing mode, you can connect objects in a form together using Qt's signals and slots mechanism
- To begin connecting objects, enter the signals and slots editing mode by opening the Edit menu and selecting Edit Signals/Slots, or by pressing the F4 key
- All widgets and layouts on the form can be connected together. However, spacers just provide spacing hints to layouts, so they cannot be connected to other objects
- To make a connectionn, press the left mouse button and drag the cursor towards the object you want to connect it to



Qt Designer's Editing Modes.

Buddy Editing Mode

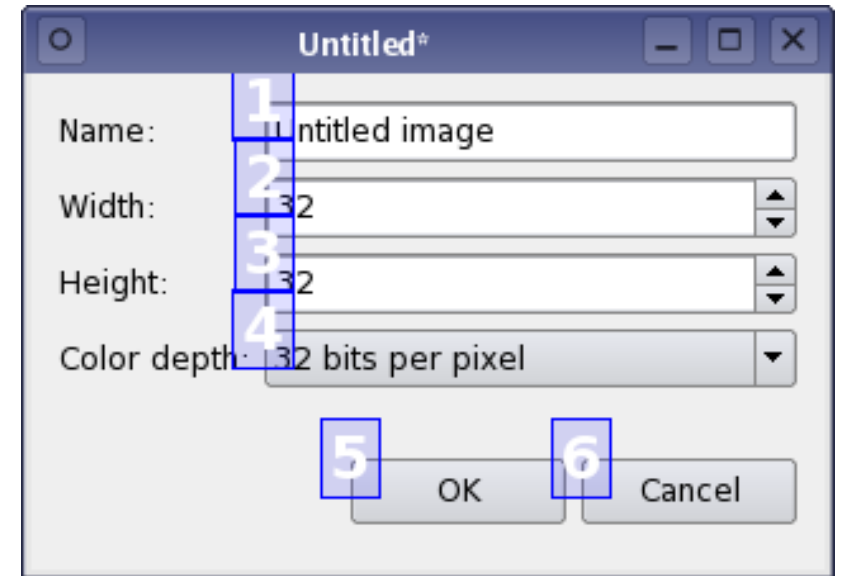
- A buddy widget accepts the input focus on behalf of a QLabel when the user types the label's shortcut key combination
- The buddy concept is also used in Qt's model/view framework
- Making Buddies: To define a buddy widget for a label, click on the label, drag the connection to another widget on the form, and release the mouse button



Qt Designer's Editing Modes.

Tab Order Editing Mode

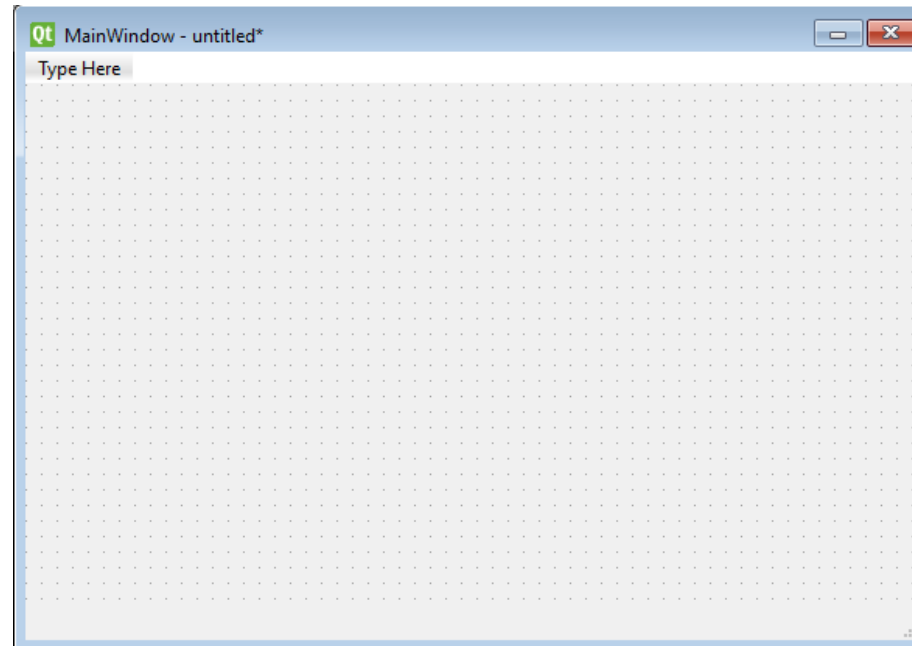
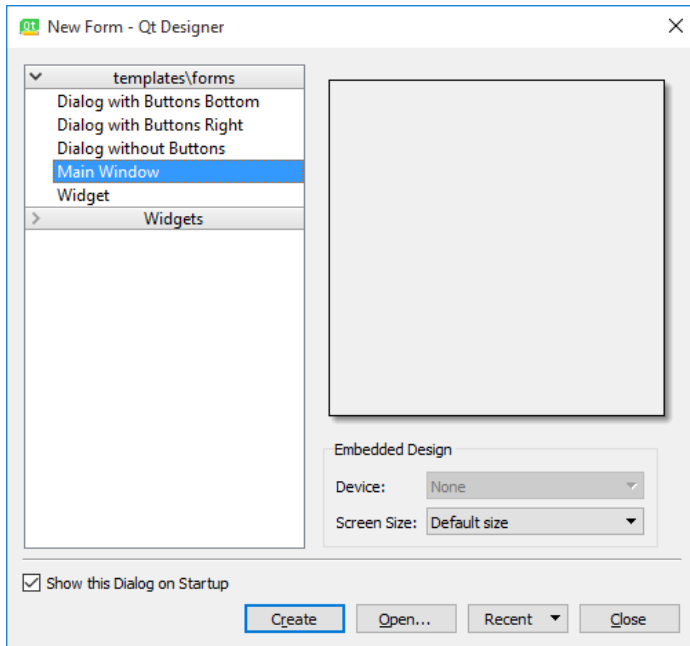
- Qt lets the user navigate between input widgets with the Tab and Shift+Tab keyboard shortcuts
- The default tab order is based on the order in which widgets are constructed
- To enter tab order editing mode, open the Edit menu and select Edit Tab Order
- The tab order is defined by clicking on each 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



Creating Main Windows in Qt Designer

Creating Main Window in 2 Easy Steps:

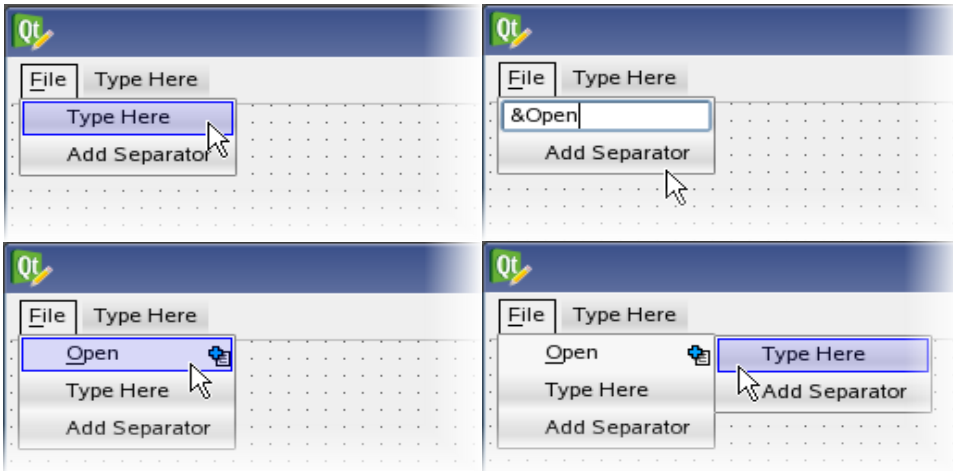
1. Open the File menu and select the New Form... Option (Ctrl+N)
2. Select the Main Window template. This template provides a main application window containing a menu bar by default



Creating Main Windows in Qt Designer

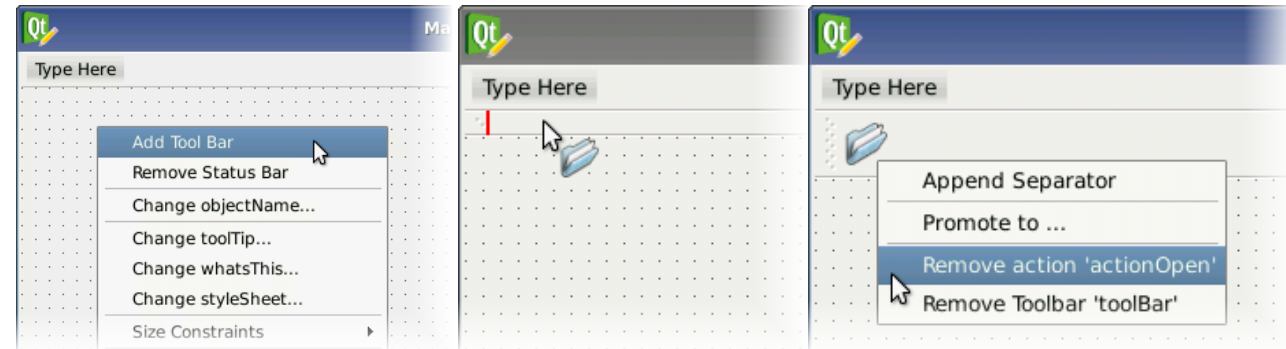
Menus

- Menus are added to the menu bar by modifying the Type Here placeholders
- Once created, the properties of a menu can be accessed using the Property Editor
- Existing menus can be removed by opening a context menu over the label in the menu bar, and selecting Remove Menu 'menu_name'



Toolbars

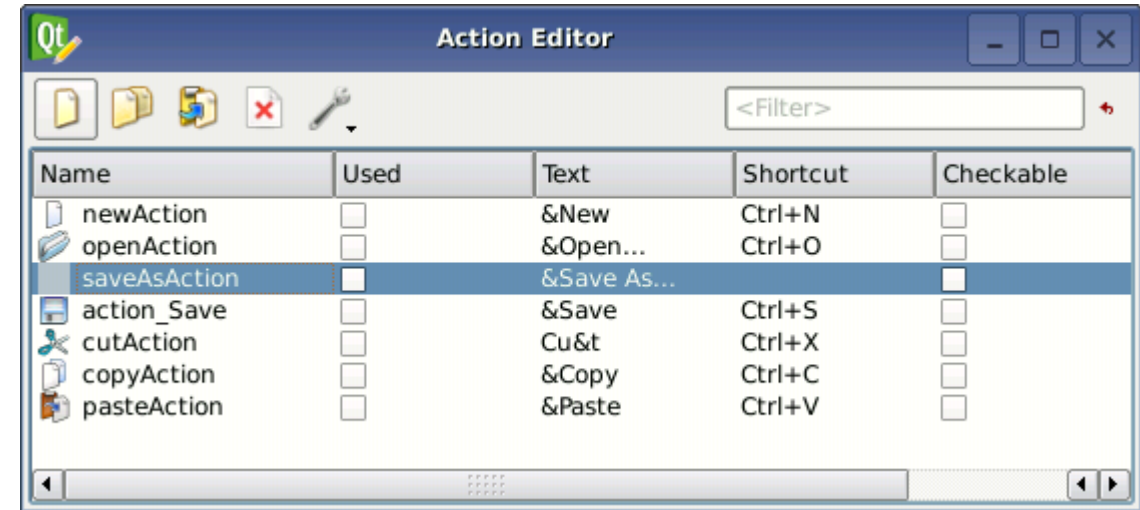
- Toolbars are added to a main window in a similar way to the menu bar: Select the Add Tool Bar option from the form's context menu
- Toolbar buttons are created as actions in the Action Editor and dragged onto the toolbar
- Since actions can be represented by menu entries and toolbar buttons, they can be moved between menus and toolbars.



Creating Main Windows in Qt Designer

Actions

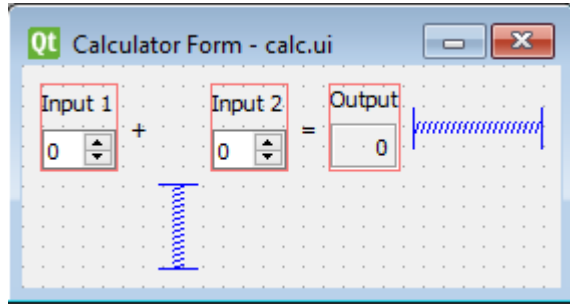
- The action editor allows you to create New actions and Delete actions. It also provides a search function, Filter, using the action's text
- Enable the action editor by opening the Tools menu, and switching on the Action Editor option
- To create an action, use the action editor's **New** button, which 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**
- 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



Using a Designer UI File in Application

- Qt Designer uses XML .ui files to store designs and does not generate any code itself
- Qt includes the uic utility that generates the C++ code that creates the user interface
- Qt also includes the QUiLoader class that allows an application to load a .ui file and to create the corresponding user interface dynamically
- PyQt5 does not wrap the QUiLoader class but instead includes the uic Python module
- Like QUiLoader this module can load .ui files to create a user interface dynamically
- Like the uic utility it can also generate the Python code that will create the user interface
- PyQt5's pyuic5 utility is a command line interface to the uic module

Using a Designer UI File in Application



Object Inspector	
Object	Class
CalculatorForm	QWidget
<name>	QVBoxLayout
label_2_2_2	QLabel
outputWidget	QLabel
<name>	QVBoxLayout
inputSpinBox2	QSpinBox
label_2	QLabel
<name>	QVBoxLayout
inputSpinBox1	QSpinBox
label	QLabel
horizontalSpacer	Spacer
label_3	QLabel
label_3_2	QLabel
verticalSpacer	Spacer

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<ui version="4.0">
```

```
  <class>CalculatorForm</class>
```

```
  <widget class="QWidget" name="CalculatorForm">
```

```
  ...
```

```
  </widget>
```

```
</ui>
```

Using a Designer UI File in Application

```
PyQt5.uic.loadUi(uifile[, baseinstance=None[, package="[, resource_suffix='_rc']]])
```

Load a Qt Designer .ui file and returns an instance of the user interface.

Parameters:

- **uifile** – the file name or file-like object containing the .ui file.
- **baseinstance** – the optional instance of the Qt base class. If specified then the user interface is created in it. Otherwise a new instance of the base class is automatically created.
- **package** – the optional package that is the base package for any relative imports of custom widgets.

Return type:

the QWidget sub-class that implements the user interface.

Using a Designer UI File in Application

```
import sys
from PyQt5 import QtWidgets, uic

class Calc(QtWidgets.QWidget):
    def __init__(self, parent=None):
        super(Calc, self).__init__(parent)
        # Set up the user interface from Designer.
        self.ui = uic.loadUi("calc.ui")

        self.ui.inputSpinBox1.valueChanged.connect(self.update)
        self.ui.inputSpinBox2.valueChanged.connect(self.update)

        self.ui.show()

    def update(self):
        a = self.ui.inputSpinBox1.value()
        b = self.ui.inputSpinBox2.value()
        self.ui.outputWidget.setText(str(a + b))

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    window = Calc()
    sys.exit(app.exec())
```

