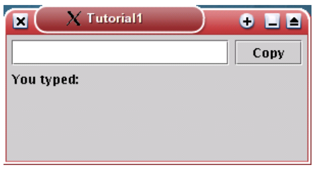
Tutorial 1

In this tutorial you will learn the very basics of using the UICompiler. You will learn to create a basic GUI in Qt Designer and how to use that to show it in your own application. Plus you will do some basic action handling.

In other words; this tutorial will learn you what the work-flow of the UICompiler solution is.

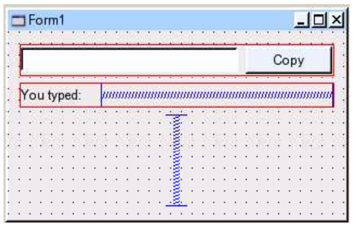


Picture 1: end result

We will create an input text box and a button with an action; see Picture1

For this project we need

1. A UI design. We create this in Qt Designer
2. An implementing java file which contains our main() and the code to act on the button press.
3. A build file to convienently manage the project.



Picture 2: The finished UI in Designer

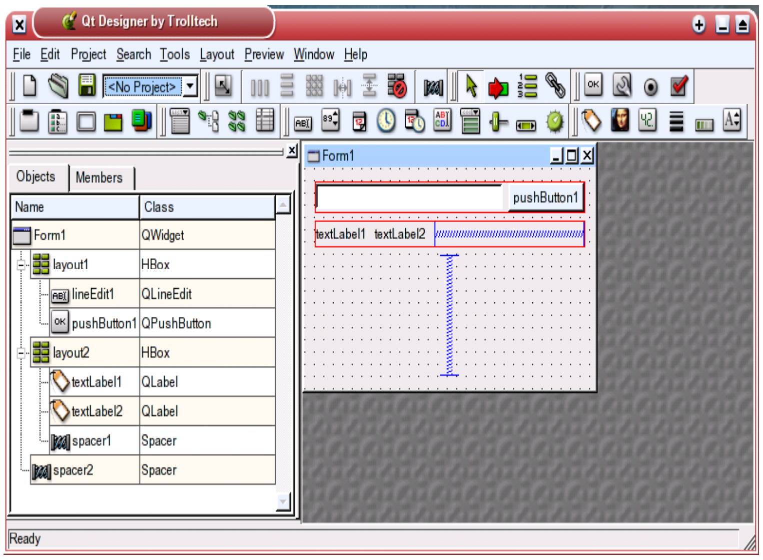
1. The UserInterface

please follow the tutorials of Qt Designer to get up to speed on how to use the application; that is material that is clearly out of scope for this document. The tutorials can be reached from the help menu of Designer after starting it.

Our goal is to get the design from Picture 2.

To do this follow these steps;

1. Create a new, empty Widget.
2. Insert a QLineEdit and a QButton by placing them randomly on the widget. (use the Tools menu)
3. Select the QLineEdit and the QButton and put them in a horizontal layouter. (use the Layout menu)
   If the ordering of the two widgets is incorrect, press undo and move the widgets, after this put them again in a layouter. The end result should have a red outline around the two widgets.
4. Insert 2 QLabels and a horizontal spacer and put those in a Horizontal layouter.
5. Insert a vertical spacer.
6. make sure you have nothing selected and use the function to layout vertically. This places all widgets in the window one beneath another.



Name	Class
Form1	QWidget
layout1	HBox
lineEdit1	QLineEdit
pushButton1	QPushButton
layout2	HBox
textLabel1	QLabel
textLabel2	QLabel
spacer1	Spacer
spacer2	Spacer

Ready

Picture 3: design after quick place and group.

! Note The widgets we joined in a layouter before behave as one object in the new layout.

At this point you should have something like Picture 3. All elements have default values, and this can vary subtely between versions, so don't worry too much if your version looks a bit different. We now have to change the texts on the buttons and labels. You can do that by using the right mouse button on a component and selecting the 'edit' option.

Notice that all widgets have automatic-generated names. These names are names you don't really want to use in your finished application. So we should rename them to something logical.

At this point I would like to point out that the screenshot right shows autogenerated variable names starting with a lowercase character. I found that preferrable to the released version of Qt Designer which uses names like 'TextLabel14'. And since the application is open source I just changed the behavior on my copy. That change has been forwarded to the maintainers who promised me it will be in the next major release.

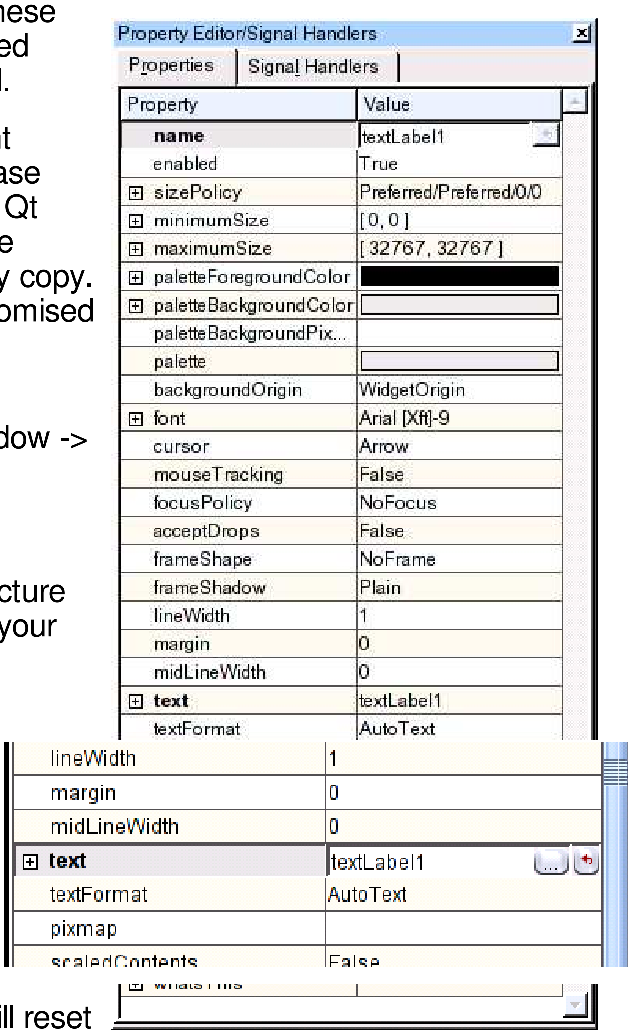
In order to change the names of the widgets we need the 'Property editor'. If you don't have it on screen, use the 'Window -> views' menu to enable it.

! Note The non-default values are displayed in bold.

Change the automatically generated name (textLabel1 in Picture 4) to something logical; this is the name that you will find in your java class.

Changing the text in the lables can be done by either changing the text in the same Property editor, or by changing it in the dialog itself. Double click on the left most label and change the text to 'You typed:'.

The right most label is empty, use the Property editor to set the text to default. Select the 'text' entry (halfway down in picture 4) by clicking on the value 'textLabel1'. You should see two little buttons appear right of the value (see inset in Picture 4), press the right one with the red icon on it. This will reset the value to default. In the case of the label this means it will not contain any text.



Property	Value
name	textLabel1
enabled	True
⊞ sizePolicy	Preferred/Preferred/0/0
⊞ minimumSize	[0, 0]
⊞ maximumSize	[32767, 32767]
⊞ paletteForegroundColor	■
⊞ paletteBackgroundColor	□
paletteBackgroundPix...	
palette	□
backgroundOrigin	WidgetOrigin
⊞ font	Arial [Xft]-9
cursor	Arrow
mouseTracking	False
focusPolicy	NoFocus
acceptDrops	False
frameShape	NoFrame
frameShadow	Plain
lineWidth	1
margin	0
midLineWidth	0
⊞ **text**	textLabel1
textFormat	AutoText
pixmap	
scaledContents	False

Picture 4: The properties of a text label.

The names and values of the button and the text box should also be changed to values that make sense, please take care to call the button 'copyButton'. This makes the rest of the tutorial easier to follow.

! Note The designer file is present in the tutorial package to compare or use for the rest of this tutorial.

2. Generating the Java Base File

The interface has to become a java class, and at the point of saving we have to choose a name for that class. This name has to be the same in two places; the filename we save it in, and also the name of the class in Designer.

Open the Form Settings dialog from the edit menu and change the 'Form1' to 'Tutorial1Base'

When you have a finished file in Qt Designer you have to save it. Create an empty directory somewhere and save the user interface file with the name of the java class it should generate. In this case save it under Tutorial1Base.ui

! Note The ui file has a capital T and a capital B since that is what your java class should also have.

In the same dir as the newly created Tutorial1Base.ui you should put an ant build file that uses the uic.jar to create the java file out of the designer file.

Use the following file and place it under build.xml

```
   <project name="tutorial1" default="jar" basedir=".">
      <property name="src" value="." />
      <property name="jars" value="jars/" />
      <property name="build.sysclasspath" value="ignore"/>

1)|      <taskdef name="uicompiler" classname="uic.anttask.UICompiler"
  |         classpath="${jars}/uic.jar"/>

      <!-- CLASSPATH -->
      <path id="myclass.path">
         <pathelement location="${build}" />
         <pathelement path="${jars}/graphics.jar" />
      </path>

      <target name="clean">
         <delete>
            <fileset dir="${src}" includes="**/*.java">

2)|            <present targetdir="${src}">
  |               <mapper type="glob" from="*.java" to="*.ui"/>
  |            </present>

            </fileset>
            <fileset dir="${src}" includes="**/*.class" />
         </delete>
      </target>


3)|   <target name="compile.ui.files">
  |      <uicompiler srcdir="${src}"
  |         includes="**/*.ui"
  |         listfiles="true"
  |      />
  |   </target>


      <target name="compile" depends="compile.ui.files" description="compile stuff">
         <javac srcdir="${src}"
            debug="true"
            classpathref="myclass.path"/>
      </target>

      <target name="jar" depends="compile" description="Create a jar file">
         <jar jarfile="t1.jar">
            <fileset dir="." includes="**/*.class"/>
            <manifest>
               <attribute name="Main-Class" value="Tutorial1"/>
               <attribute name="Class-Path" value="${jars}/graphics.jar"/>
               <attribute name="Built-By" value="${user.name}"/>
            </manifest>
         </jar>
      </target>
   </project>
```

! Note The above file can be found in the tutorial package

There are 3 parts relevant for the UICompiler, all other parts are normal ant targets, look at the ant manual for more info

```
http://jakarta.apache.org/ant/manual/index.html
```

1) To be able to use the UICompiler from ant it has to be registered; this line will to that.

2) A special part has been placed in the clean target to remove the generated java files.

3) This is the real call that will create the java files from the .ui files

Make sure the uic.jar and graphics.jar are available in the global classpath when starting ant, and also available in the place that the buildfile expects it to find; in this case the jars subdir.

Start ant and you will find a newly created java file called 'Tutorial1Base.java' created out of the 'Tutorial1Base.ui' file.

In order to get a window with the tutorial we should create a main and open a new JFrame. Since the java file is generated it is not wise to change that file, so we extend the class by creating a new one called Tutorial1.

File: Tutorial1.java

```
1)| import javax.swing.*;
  |
  | public class Tutorial1 extends Tutorial1Base {
2)|    public void copyButtonPressedSlot() {
  |       contentLabel.setText(myLineEdit.getText());
  |    }
3)|    public static void main (String args[]) {
  |       JFrame frame = new JFrame("Tutorial1");
  |       frame.addWindowListener(
  |          new java.awt.event.WindowAdapter() {
  |             public void windowClosing(java.awt.event.WindowEvent e) {
  |                System.exit(0);
  |             }
  |          }
  |       );
  |       frame.getContentPane().add(new Tutorial1());
  |       frame.setBounds (20, 20, 300, 160);
  |       frame.show();
  |    }
  | }
```

Part 1 shows that the new class extends the automatically generated Tuturial1Base.

Part 2 provides an implementation for the button handling. As we named the button 'copyButton' UICompiler will call the copyButtonPressedSlot method as soon as the button is pressed by the user, simply providing an implementation of that method is enough to handle button presses.

Another widget we placed in designer was the contentLabel, which is a label. If you get the Duh feeling that simply means the naming was logical! The last widget used in this method is the myLineEdit which is a lineEdit.

The code does nothing but copy the text of the lineEdit to the label.

Part 3 is code to open a new window using a JFrame and at the same time handle windowClosing events. For most of your usage the code can just be copied without knowing what it does. Take a look at the javadocs for Swing to find out more about that code.

3. Running the code

If you start and again it will create a jar of all the sources in this directory and register in the jar that the Tutorial1 class has a main that is to be executed when the jar is started.

The generated t1.jar can then be started by either clicking on it, or typing 'java -jar t1.jar'.

This should give you the window as showed in Picture 1, and you have successfully finished this tuturial.