INTRODUCTION TO QT PROGRAMMING

MANUEL RODRIGUEZ
AUTOMOTIVE FIELD APPLICATIONS ENGINEER

AMF-AUT-T2795 | AUGUST 2017

SECURE CONNECTIONS
FOR A SMARTER WORLD

AGENDA

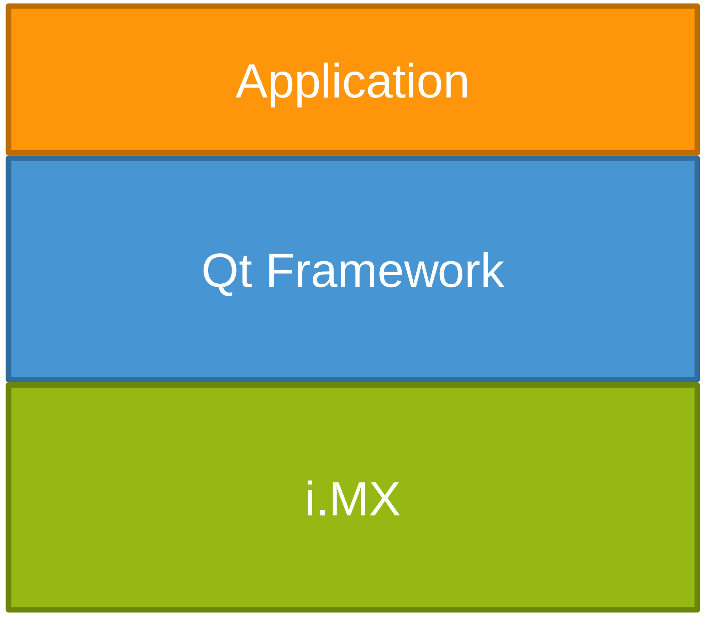
- What is Qt?
- Setting up Qt creator
- Hello world
- Building a calculator
- Building a weather station

01.

What is Qt?

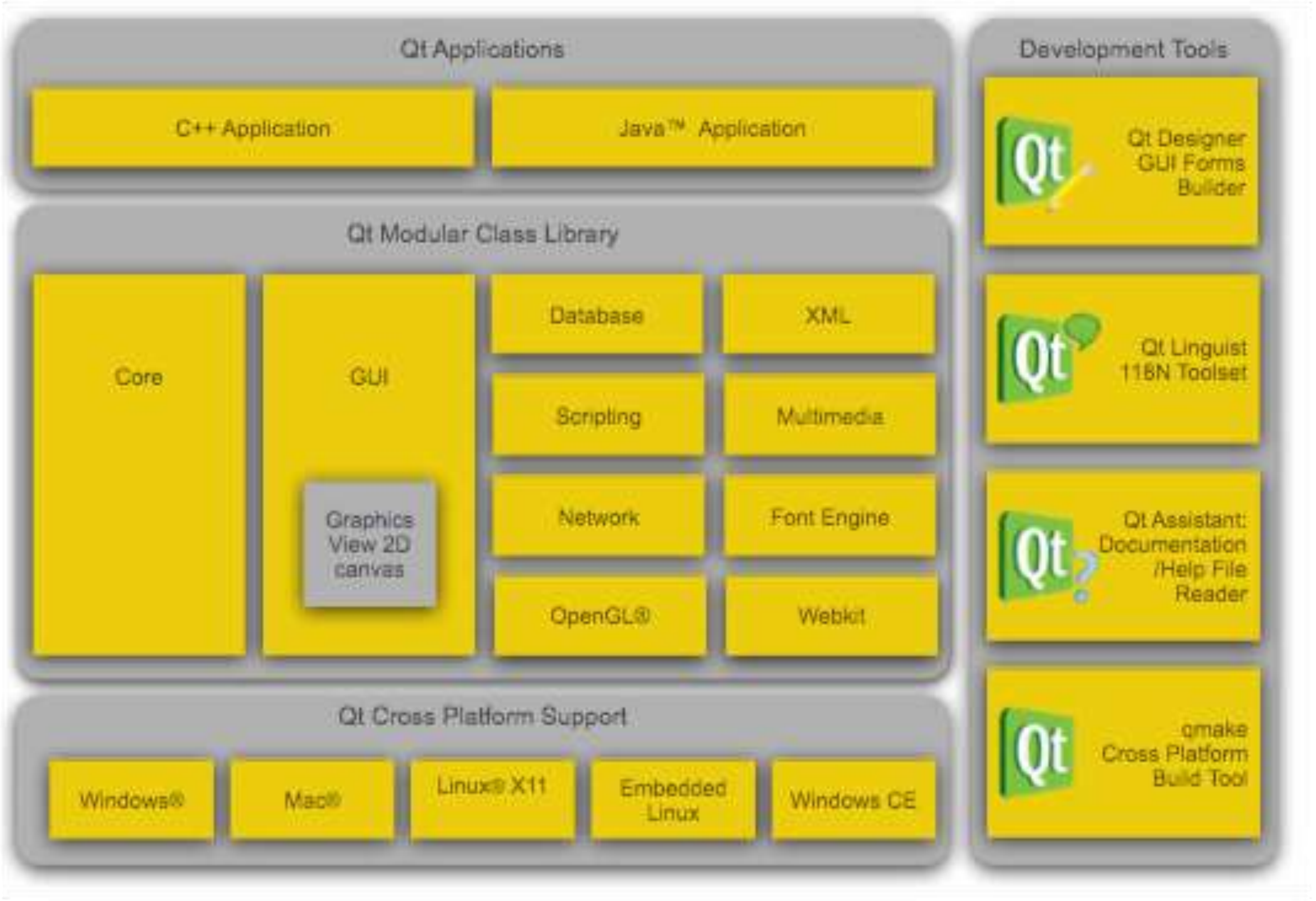
Introduction

- Qt (“cute”) is a cross-platform application framework that is used for developing application software that can be run on various software and hardware platforms with little or no change in the underlying codebase, while still being a native application with native capabilities and speed.



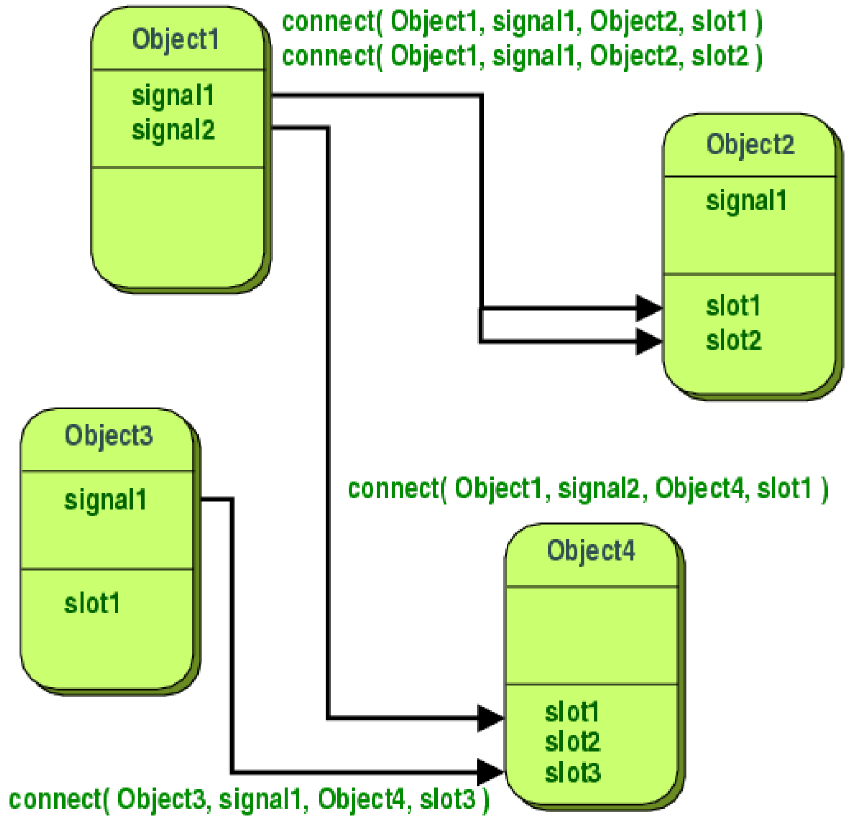
Introduction

- Much more than a GUI framework, Qt also offers support for SQL, XML, Network and some sensors.
- Write once, deploy everywhere.
- Qt is available with both commercial and open source (GPL 2.0, GPL 3.0, and LGPL 3.0) licenses.



Signals and slots

- Signals and slots are used for communication between objects.
- The concept is that GUI widgets can send signals containing event information which can be received by other controls using special functions known as slots.



Object1
signal1
signal2
connect(Object1, signal1, Object2, slot1)
connect(Object1, signal1, Object2, slot2)
Object2
signal1
slot1
slot2
Object3
signal1
slot1
connect(Object1, signal2, Object4, slot1)
Object4
slot1
slot2
slot3
connect(Object3, signal1, Object4, slot3)

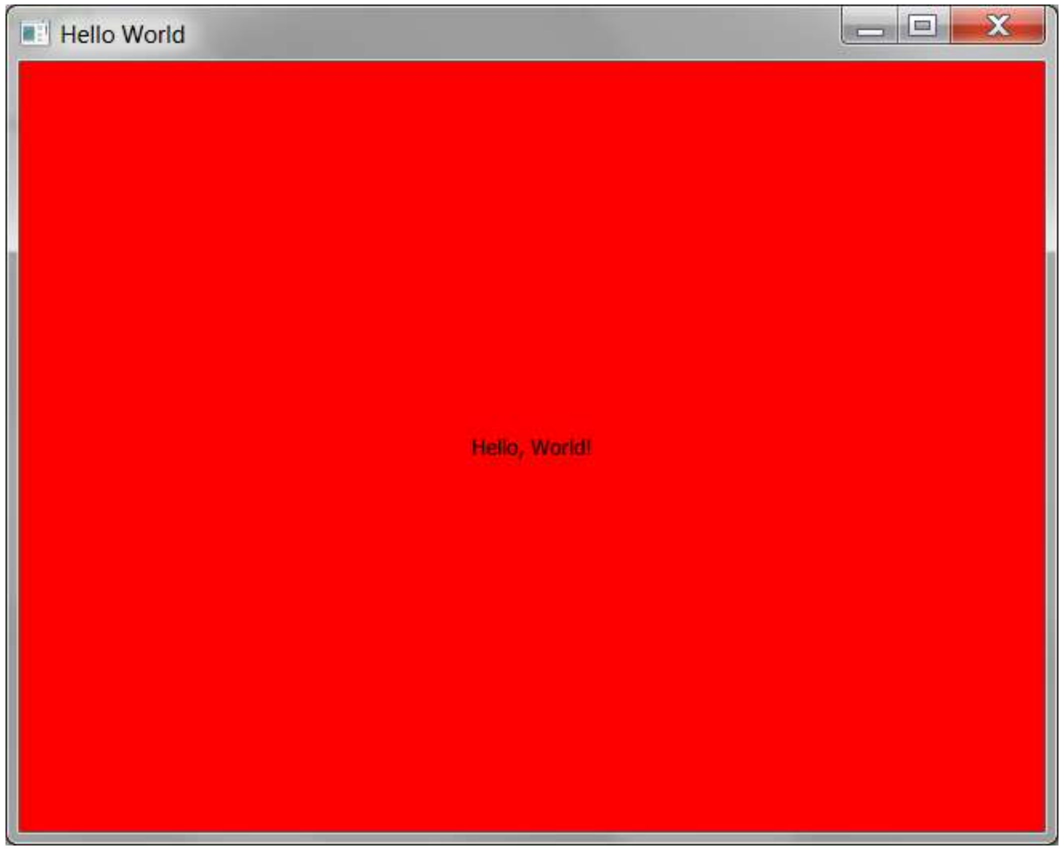
QML

- QML is a declarative language that allows user interfaces to be described in terms of their visual components and how they interact and relate with one another.
- A QML document defines a hierarchy of objects with a highly-readable, structured layout.
- Every QML document consists of two parts: an imports section and an object declaration section.

```
import QtQuick 2.6

Rectangle {
    width: 200
    height: 100
    color: "red"
    Text {
        anchors.centerIn: parent
        text: "Hello, World!“
    }
}
```

QML

- Output from the last example:



```
Hello, World!
```

02.

Setting up Qt creator with the i.MX

Setting up the meta tool-chain with Qt creator

- In order to be able to cross-compile applications a meta-toolchain is created with the Yocto project.
- After installing the meta-toolchain Qt creator is configured to use the tools that were created by the Yocto Project to build applications for the target device.
- This process is thoroughly explained in the following thread in our community:
- https://community.nxp.com/docs/DOC-328543

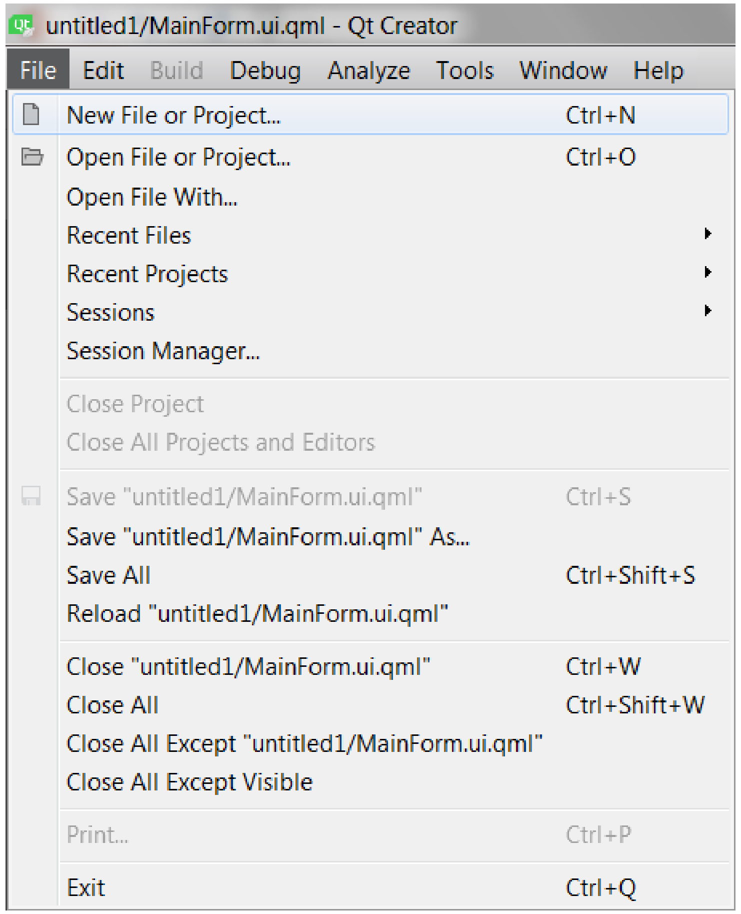
- Since the computers on this hands-on have Qt creator already installed we will skip these steps, you can follow the document above to setup your environment at home.

03.

Hello world

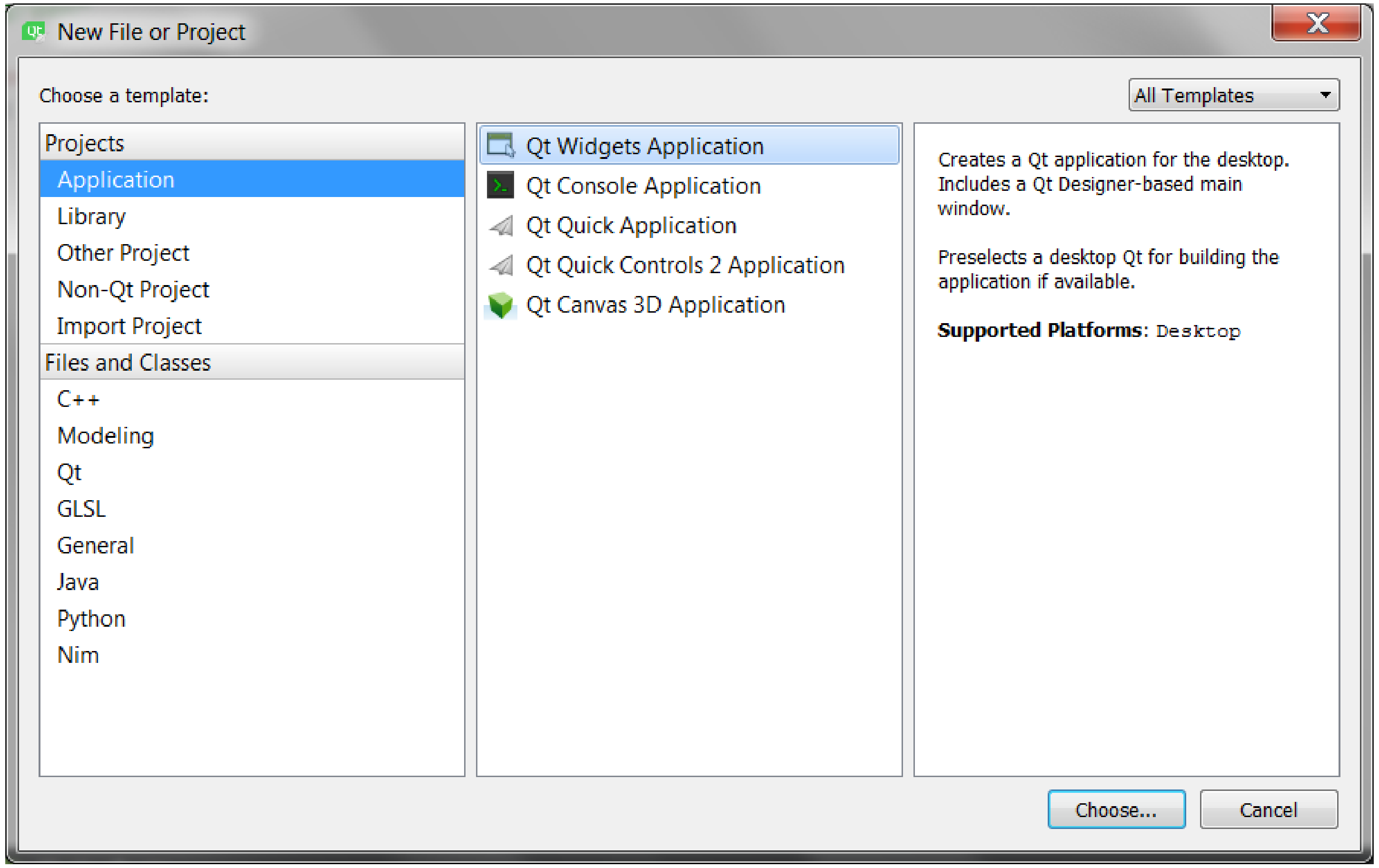
Creating the project…

- Click on File>New File or Project


untitled1/MainForm.ui.qml - Qt Creator
File
Edit
Build
Debug
Analyze
Tools
Window
Help
New File or Project... Ctrl+N
Open File or Project... Ctrl+O
Open File With...
Recent Files
Recent Projects
Sessions
Session Manager...
Close Project
Close All Projects and Editors
Save "untitled1/MainForm.ui.qml" Ctrl+S
Save "untitled1/MainForm.ui.qml" As...
Save All Ctrl+Shift+S
Reload "untitled1/MainForm.ui.qml"
Close "untitled1/MainForm.ui.qml" Ctrl+W
Close All Ctrl+Shift+W
Close All Except "untitled1/MainForm.ui.qml"
Close All Except Visible
Print... Ctrl+P
Exit Ctrl+Q

Creating the project…

- Select Application>Qt Widgets Application



Creates a Qt application for the desktop. Includes a Qt Designer-based main window.

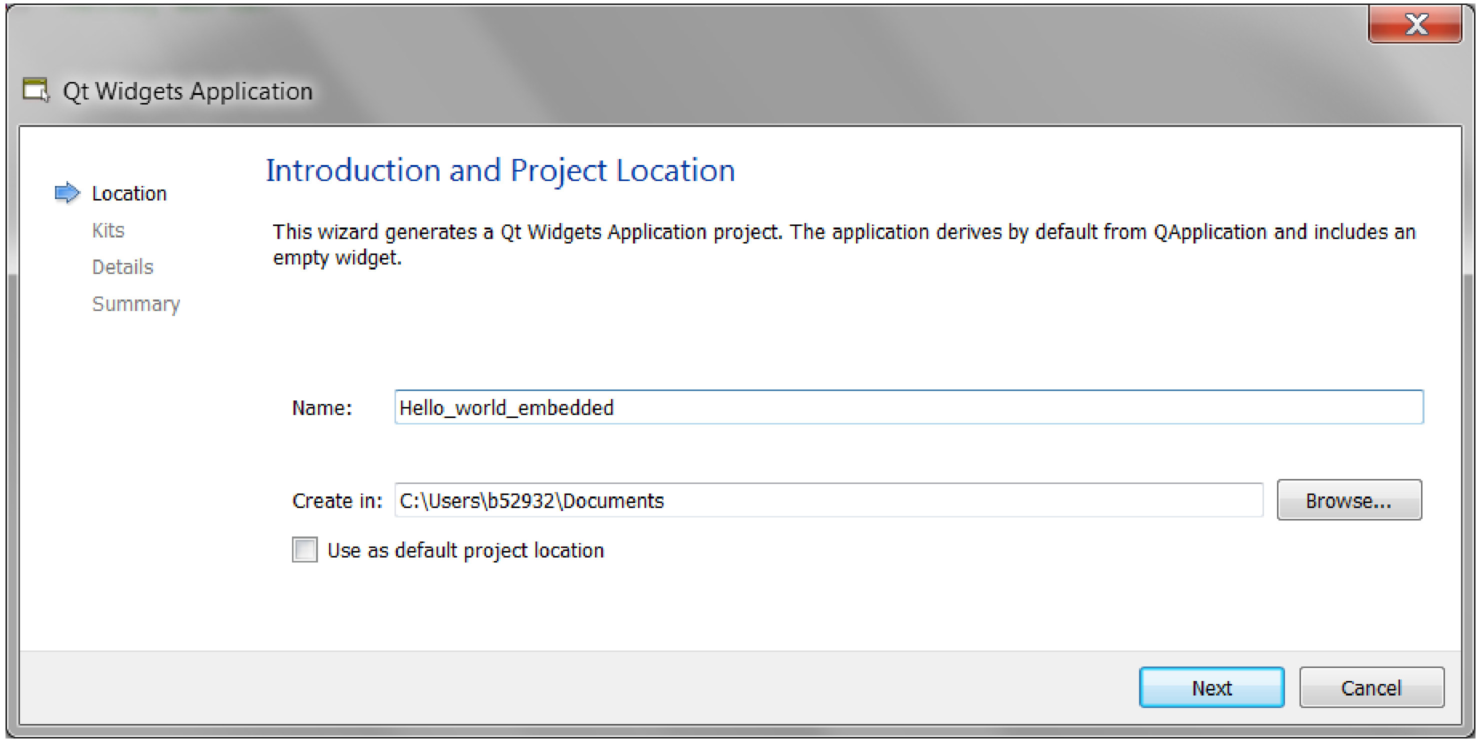
Preselects a desktop Qt for building the application if available.

Supported Platforms: `Desktop`

NOTE: Qt Quick UI projects cannot be deployed to embedded or mobile target platforms. For those platforms, create a Qt Quick application instead or a Qt Widget app.

Creating the project..

- Select a name for the project and the folder to store it.


Qt Widgets Application

Introduction and Project Location

This wizard generates a Qt Widgets Application project. The application derives by default from QApplication and includes an empty widget.

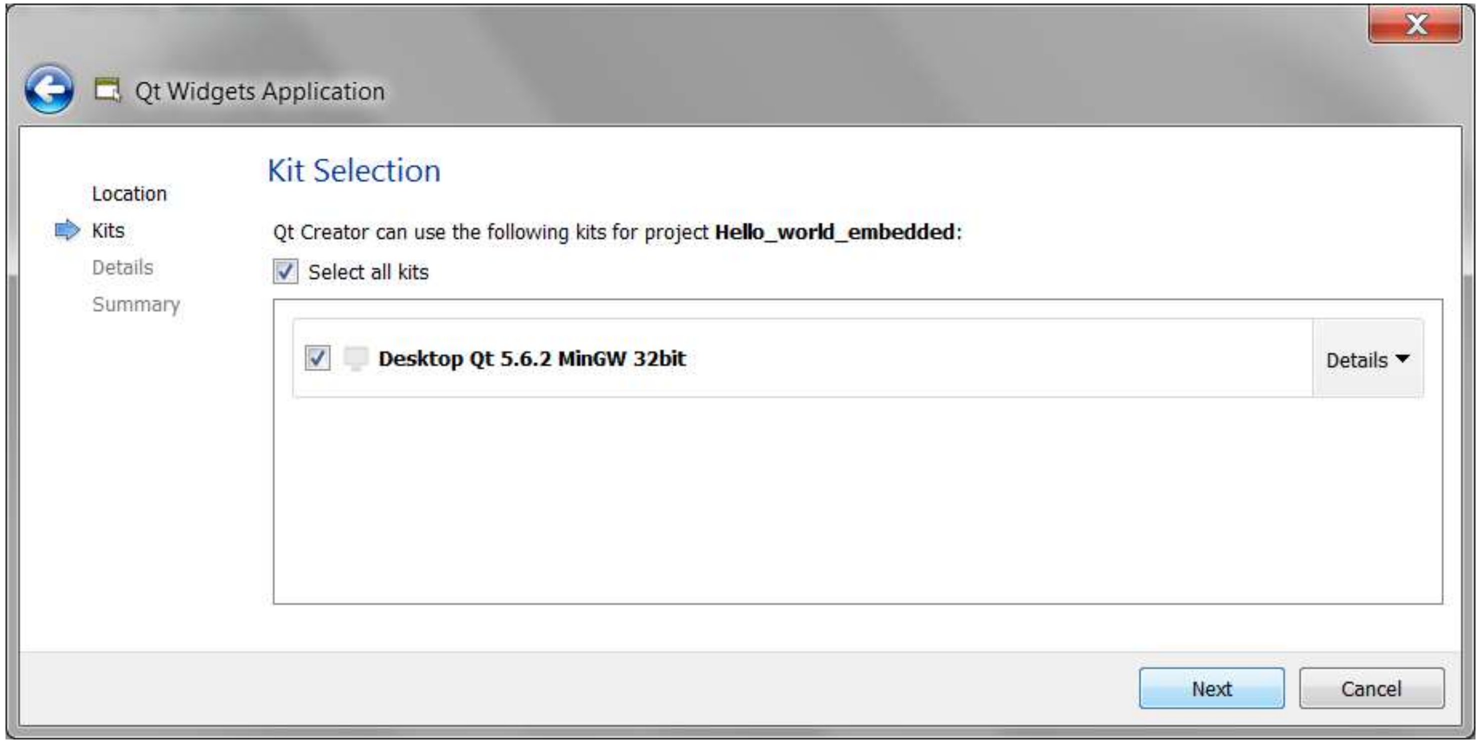
Name: Hello_world_embedded

Create in: C:\Users\b52932\Documents

Use as default project location

Creating the project…

- Select the kit (target) for the project, on this hands-on the Desktop will be the target, but once you configure your kit for the i.MX you can start building applications for it.



Kit Selection

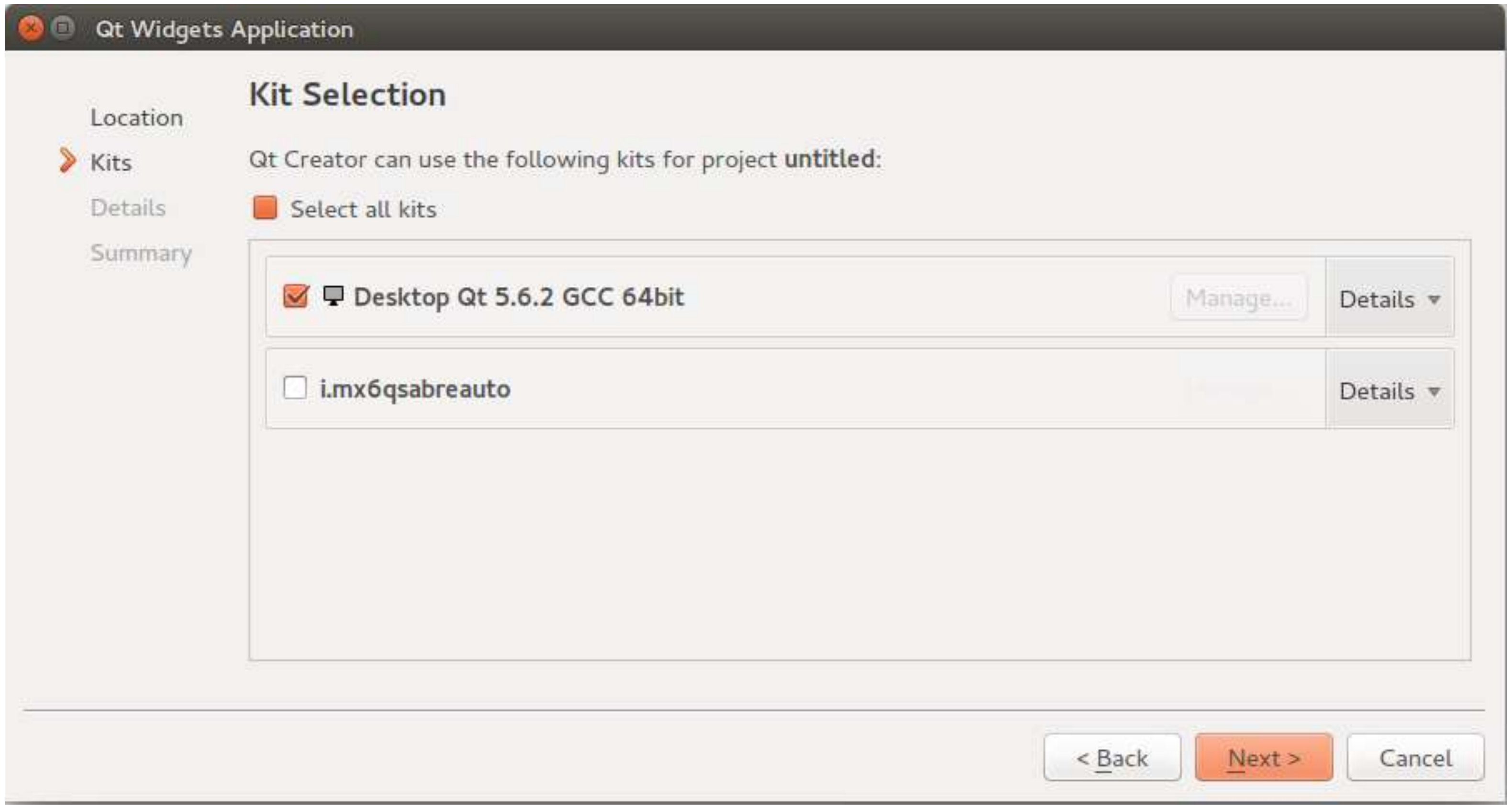
Qt Creator can use the following kits for project **Hello_world_embedded**:

☑ Select all kits

☑ **Desktop Qt 5.6.2 MinGW 32bit**

Creating the project…

- This is how the kit selection looks like after the i.mx target is added. (Ubuntu)



Qt Widgets Application

Kit Selection

Qt Creator can use the following kits for project **untitled**:

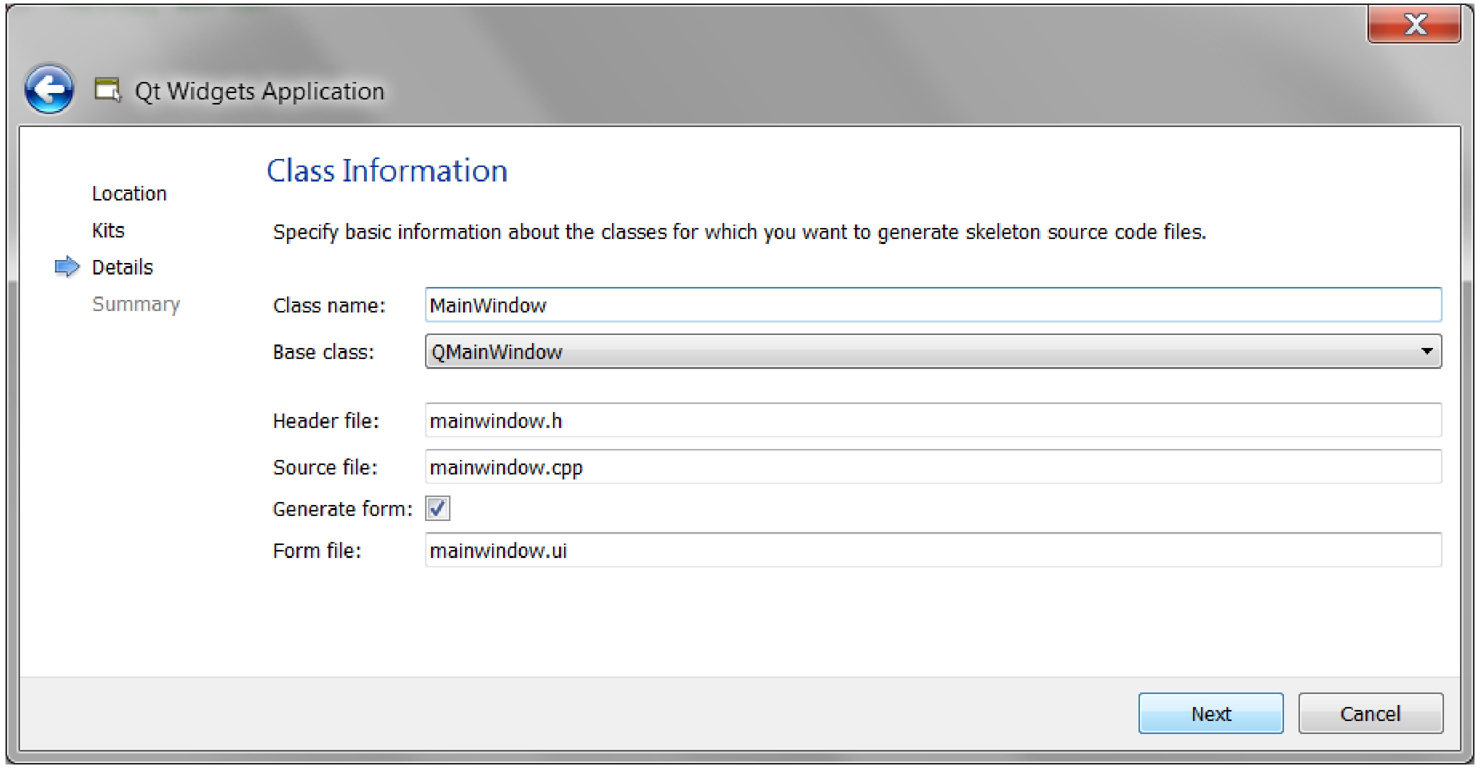
☐ Select all kits

✓ Desktop Qt 5.6.2 GCC 64bit

☐ i.mx6qsabreauto

Creating the project…

- Select the class name and click next (we will leave it in its default state).


Qt Widgets Application

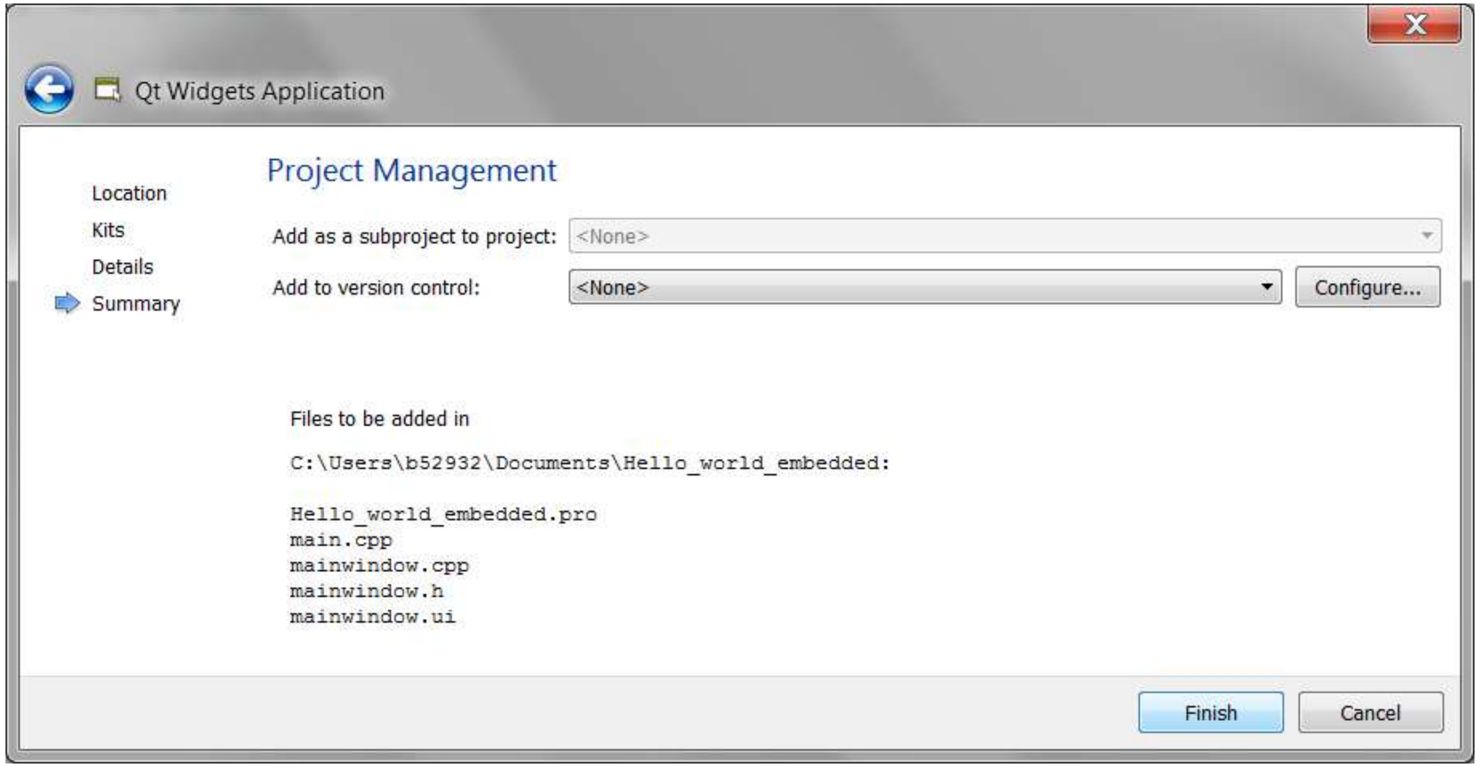
Class Information

Specify basic information about the classes for which you want to generate skeleton source code files.

Class name: MainWindow
Base class: QMainWindow
Header file: mainwindow.h
Source file: mainwindow.cpp
Generate form: ✓
Form file: mainwindow.ui

Creating the project…

- Click on finish and the project will be created.



Project Management

Add as a subproject to project: <None>

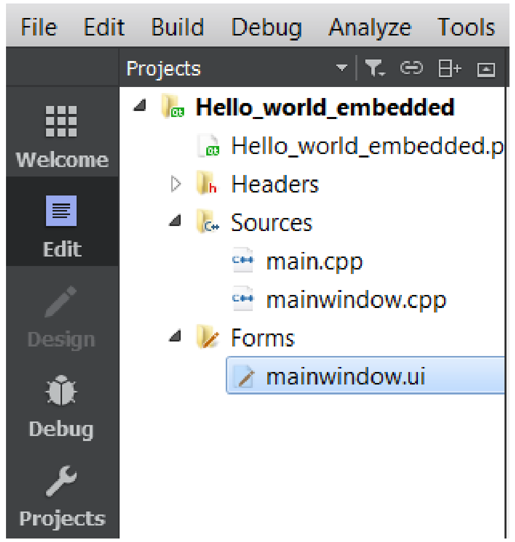
Add to version control: <None>

Files to be added in

```
C:\Users\b52932\Documents\Hello_world_embedded:

Hello_world_embedded.pro
main.cpp
mainwindow.cpp
mainwindow.h
mainwindow.ui
```

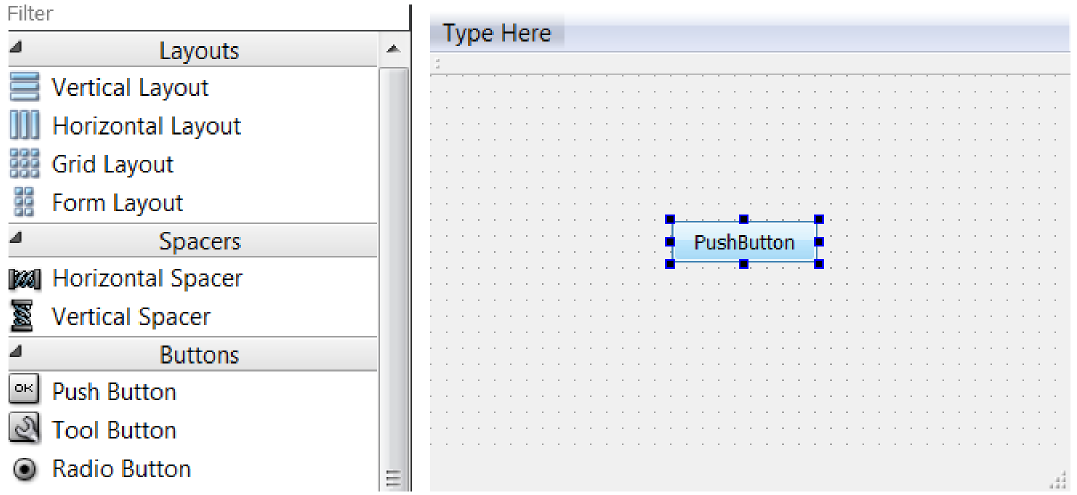
Hello world!

- Click on the forms folder and then click again on the mainwindow.ui
- This file contains the “layout” information for our GUI.



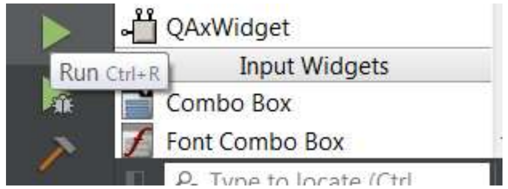
Hello world!

- Drag and drop a push button widget to your GUI.
- Double click on it and type “Hello world” on it.

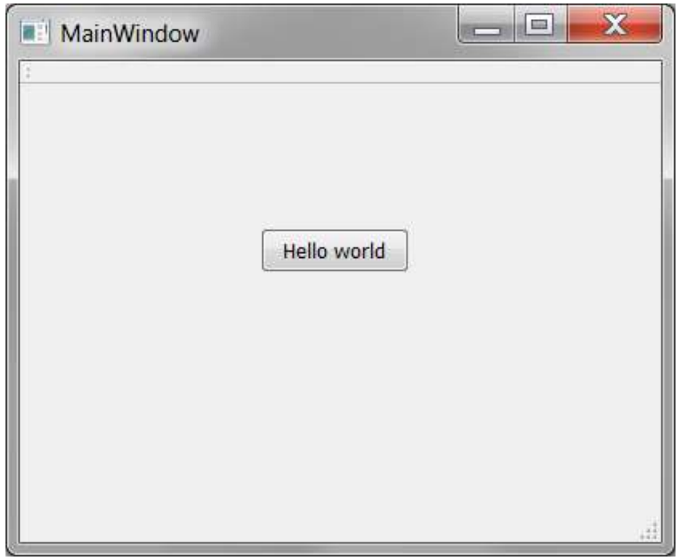


Hello world!

- Run the project by clicking at the icon below or pressing Ctrl+R

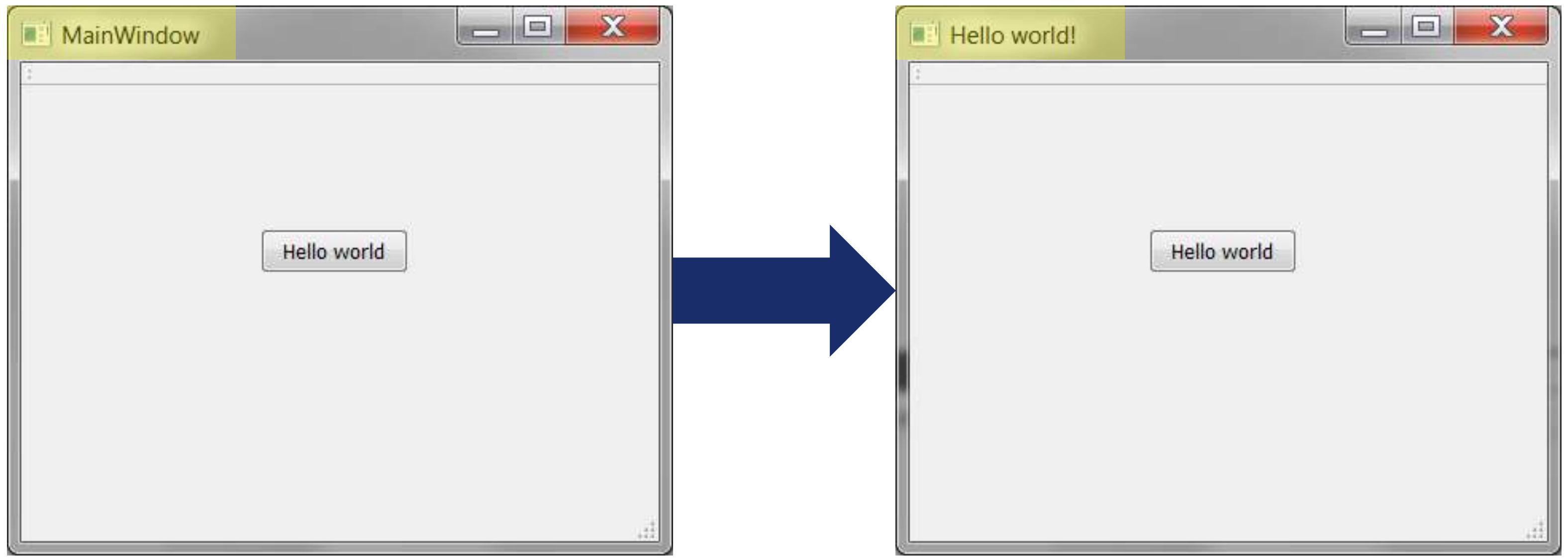


- The following window should now appear:



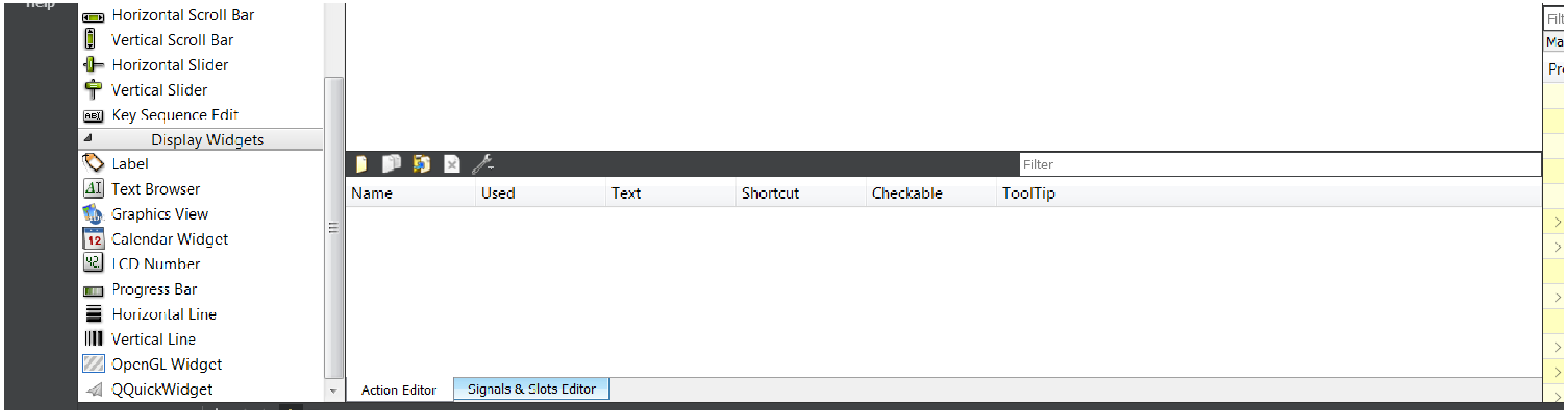
Challenge!

- Play around in the design view and find a way to change the name of the window from “MainWindow” to “Hello world!”



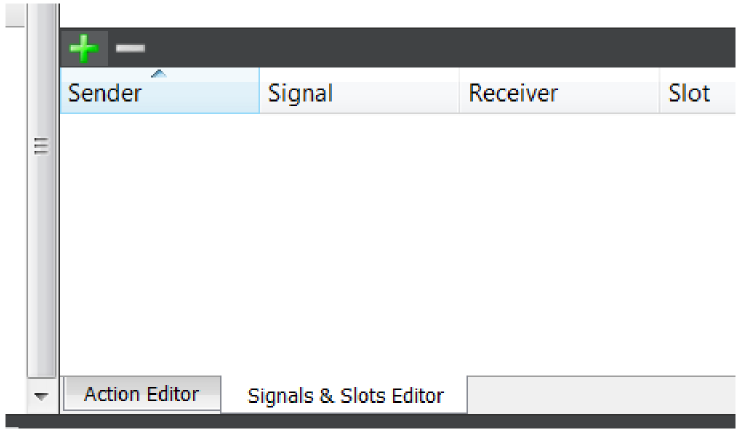
Signals and slots

- In the design view click on the “Signals & Slots Editor” tab.



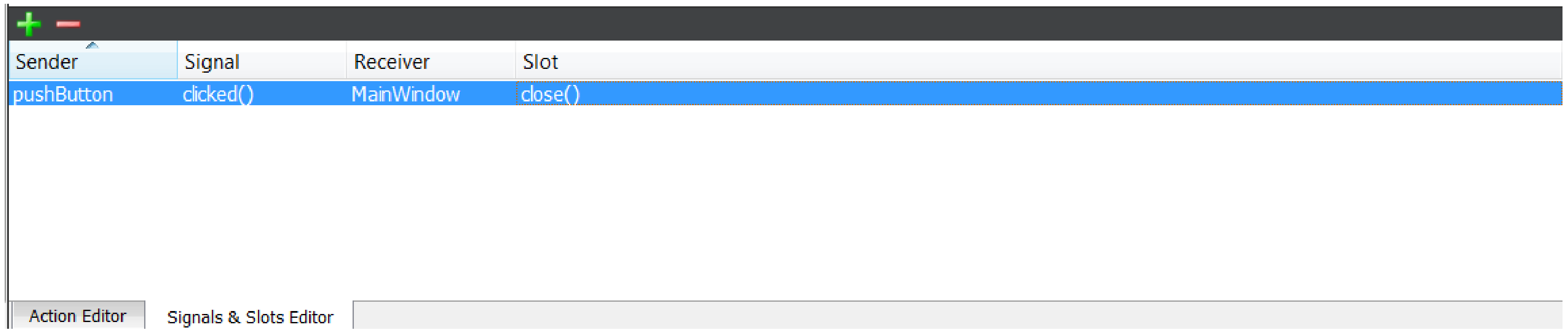
Signals and slots

- Click on the plus sign to add a signal & slot connection



Signals and slots

- Select the pushButton object as the sender and the “clicked()” signal.
- Select the MainWindow as the receiver and the “close()” slot.
- Run the project, now clicking on the button will close the window.



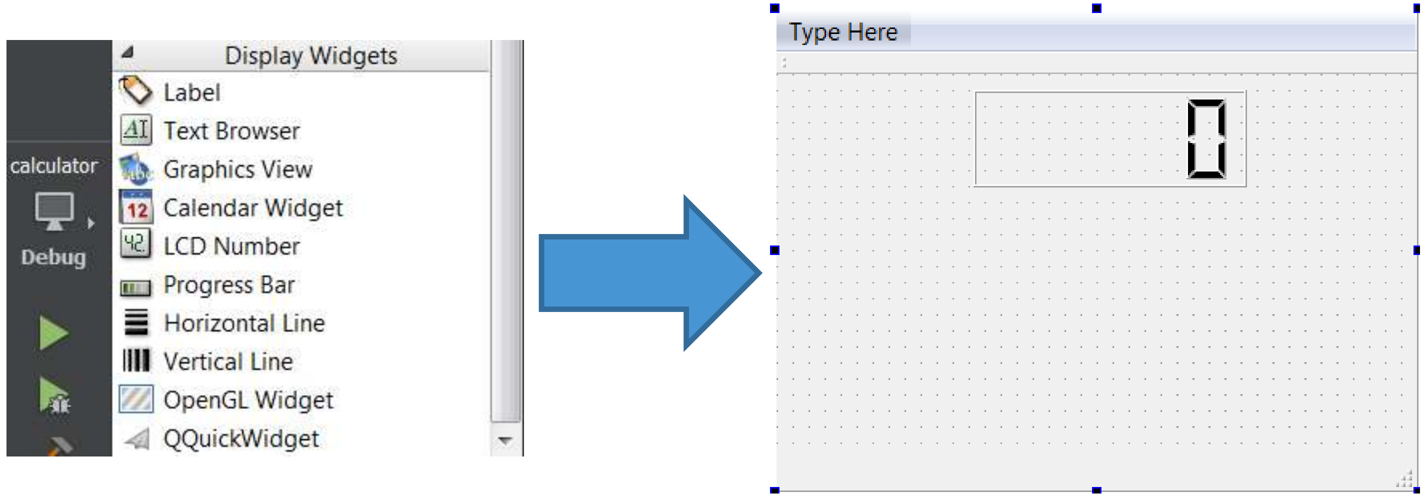
Sender	Signal	Receiver	Slot
pushButton	clicked()	MainWindow	close()

04.

Building a calculator

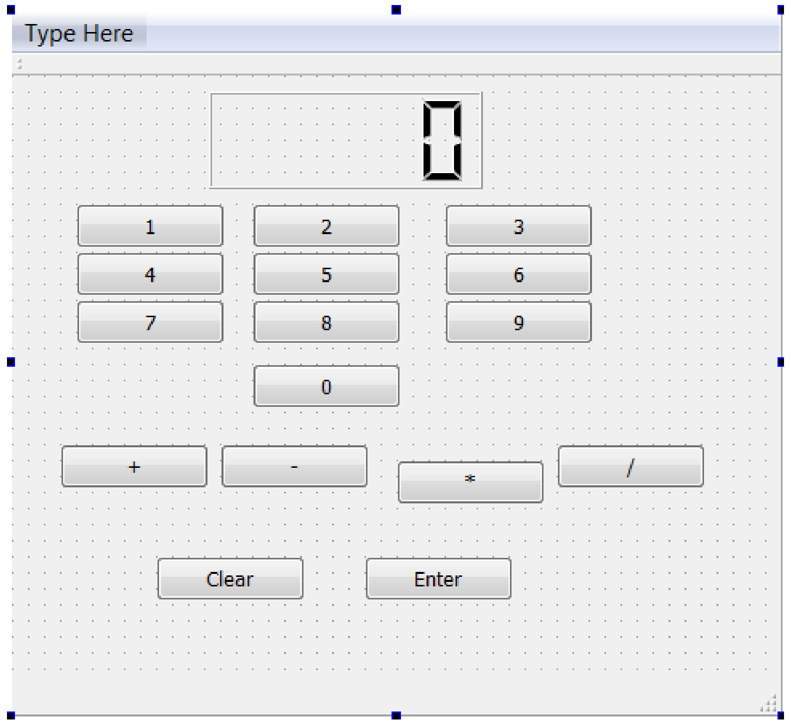
Building a calculator

- Follow the previous steps to create a new project, you can select the name of the class to be Calculator.
- Go to the design view and drag and drop an LCD number to your GUI



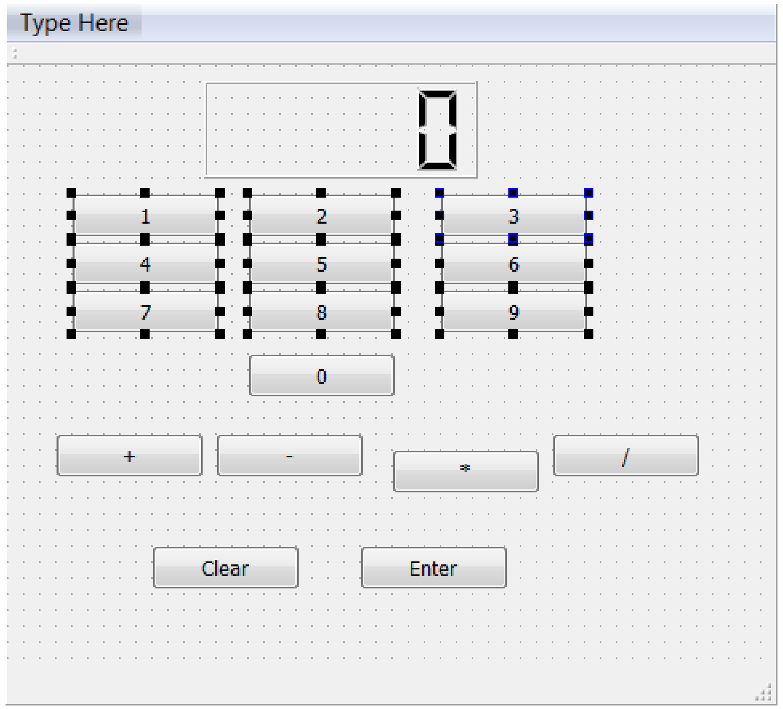
Building a calculator

- Drag and drop all the buttons as in the following image, do not worry about the placement we will arrange them in the next step.



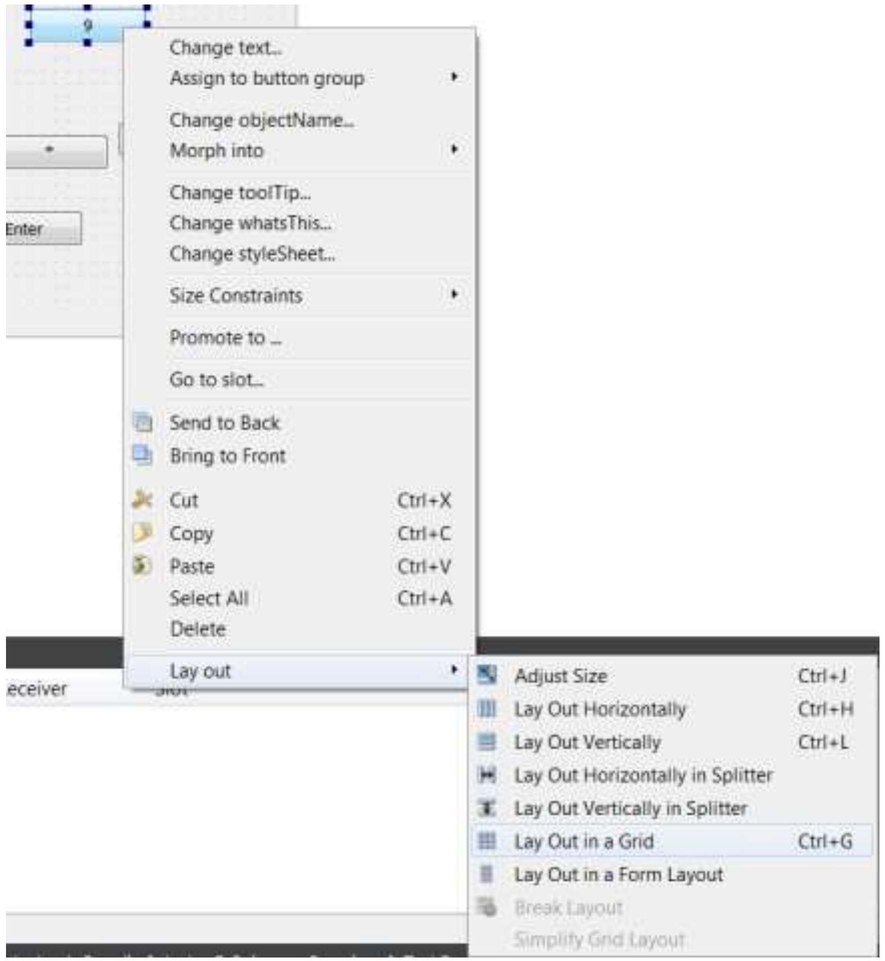
Building a calculator

- Select the buttons that need to be arranged in a "cluster"



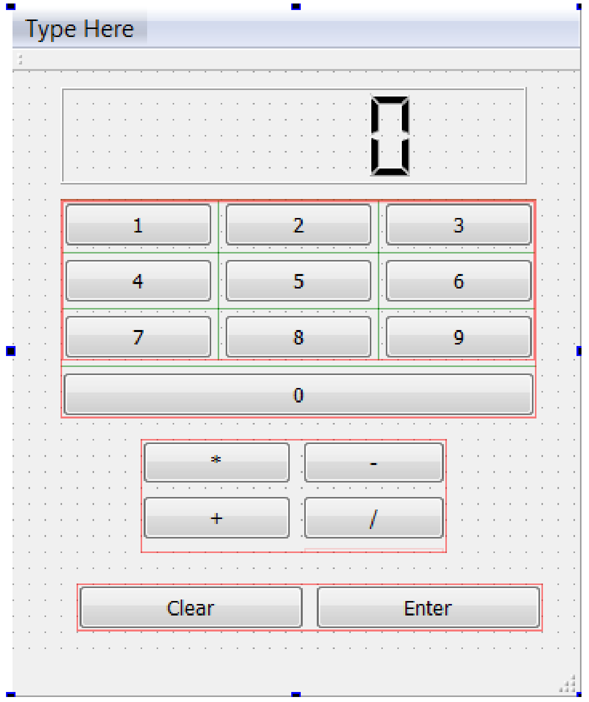
Building a calculator

- Once selected, right click on one of them and select the Lay Out in a Grid option.



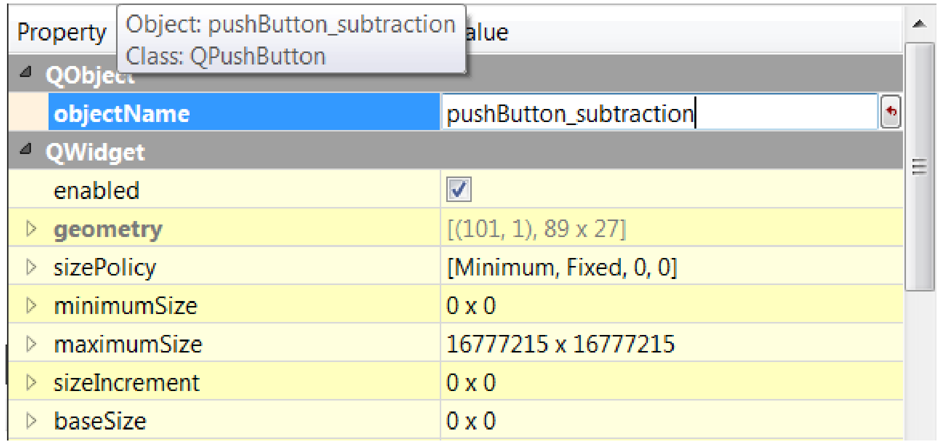
Building a calculator

- This is how your calculator might look like after applying the lay out to some buttons.



Building a calculator

- To ease the signals & slots handling we will change the name of the button objects to reflect their functionality, e.g. pushButton_0 up to pushButton_9 and a pushButton object for each of the available math operations.

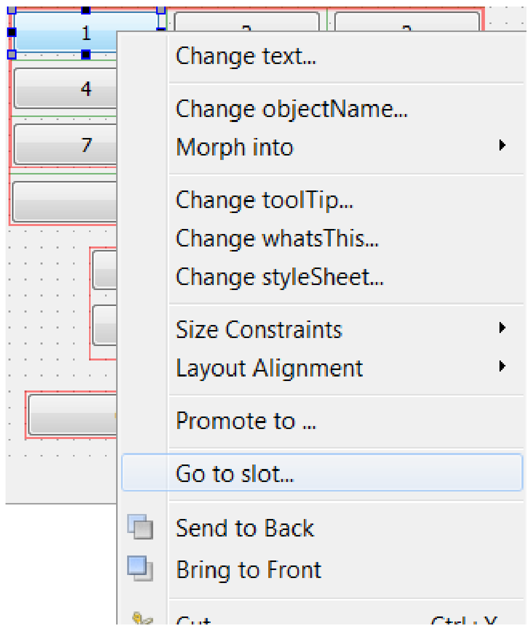


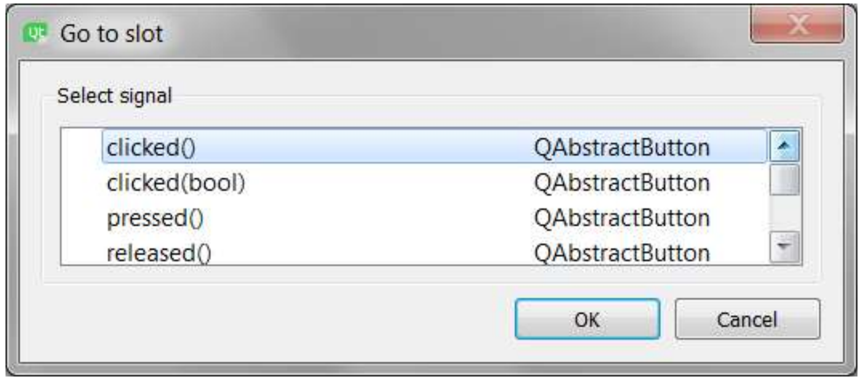
Object: pushButton_subtraction
Class: QPushButton

Property	Value
QObject	
objectName	pushButton_subtraction
QWidget	
enabled	✓
geometry	[(101, 1), 89 x 27]
sizePolicy	[Minimum, Fixed, 0, 0]
minimumSize	0 x 0
maximumSize	16777215 x 16777215
sizeIncrement	0 x 0
baseSize	0 x 0

Building a calculator

- Right click one of the buttons and select the Go to slot… option. Then select the signal to be used, clicked() on this case.

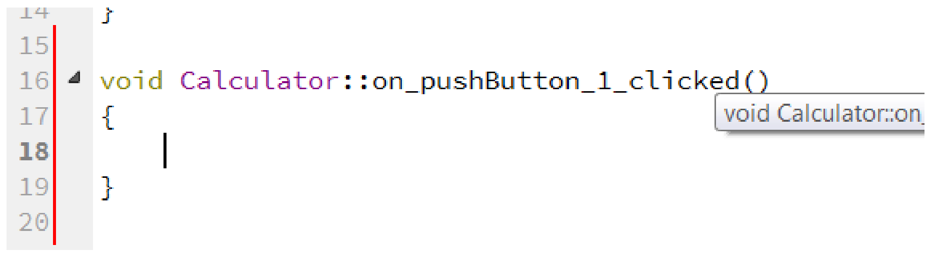




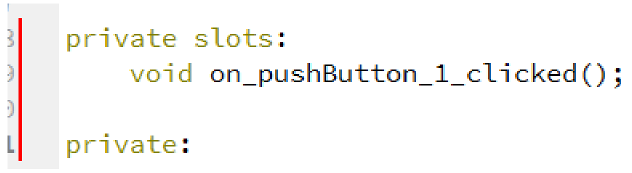
clicked()	QAbstractButton
clicked(bool)	QAbstractButton
pressed()	QAbstractButton
released()	QAbstractButton

Building a calculator

- The previous step will generate a stub function in calculator.cpp and a prototype in calculator.h



```
}

void Calculator::on_pushButton_1_clicked()
{

}

```

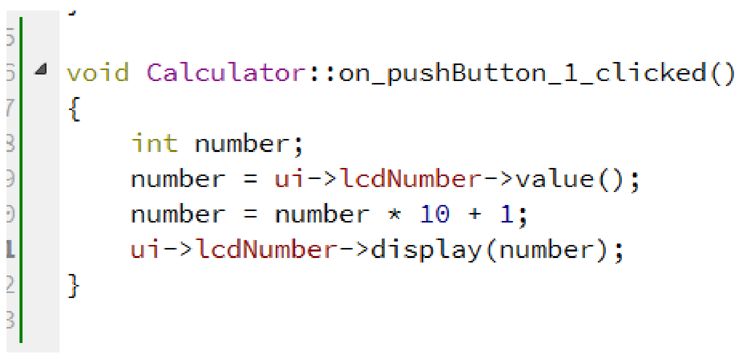


```
private slots:
    void on_pushButton_1_clicked();

private:
```

- The format remains the same for all the buttons you can simply copy/paste the stubs and change their name or perform the previous step on all of them.

Building a calculator

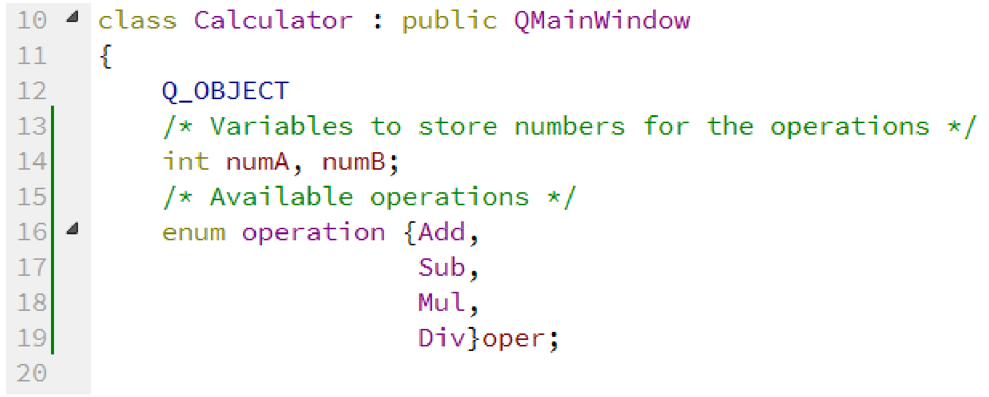
- This is the “template” that we will use for all the number buttons, this updates the value on the display with respect the clicked button.



```
void Calculator::on_pushButton_1_clicked()
{
    int number;
    number = ui->lcdNumber->value();
    number = number * 10 + 1;
    ui->lcdNumber->display(number);
}
```

Building a calculator

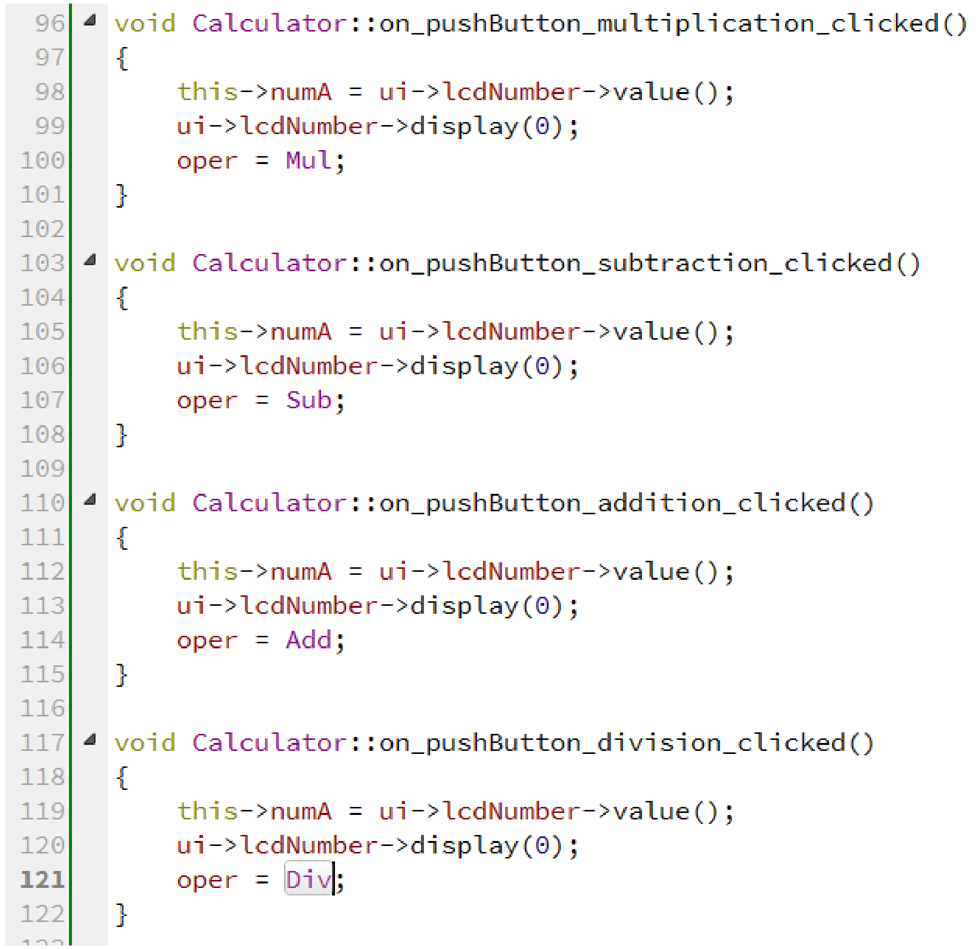
- We will add two variables on calculator.h to hold the value of the numbers for the operations.
- We will also declare an enum to hold all the available operations.



```
class Calculator : public QMainWindow
{
    Q_OBJECT
    /* Variables to store numbers for the operations */
    int numA, numB;
    /* Available operations */
    enum operation {Add,
                    Sub,
                    Mul,
                    Div}oper;

```

Building a calculator

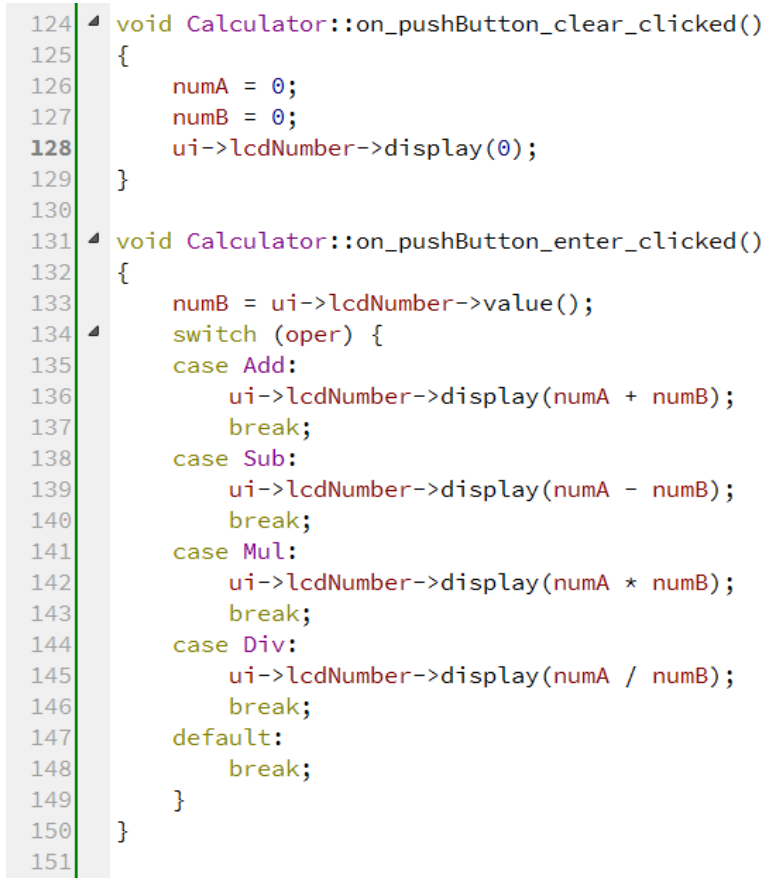
- This is how we will implement the operations, the value is obtained from the LCD object and stored as the first number for the operation, then we select the type of operation to perform.



```
void Calculator::on_pushButton_multiplication_clicked()
{
    this->numA = ui->lcdNumber->value();
    ui->lcdNumber->display(0);
    oper = Mul;
}

void Calculator::on_pushButton_subtraction_clicked()
{
    this->numA = ui->lcdNumber->value();
    ui->lcdNumber->display(0);
    oper = Sub;
}

void Calculator::on_pushButton_addition_clicked()
{
    this->numA = ui->lcdNumber->value();
    ui->lcdNumber->display(0);
    oper = Add;
}

void Calculator::on_pushButton_division_clicked()
{
    this->numA = ui->lcdNumber->value();
    ui->lcdNumber->display(0);
    oper = Div;
}
```

Building a calculator

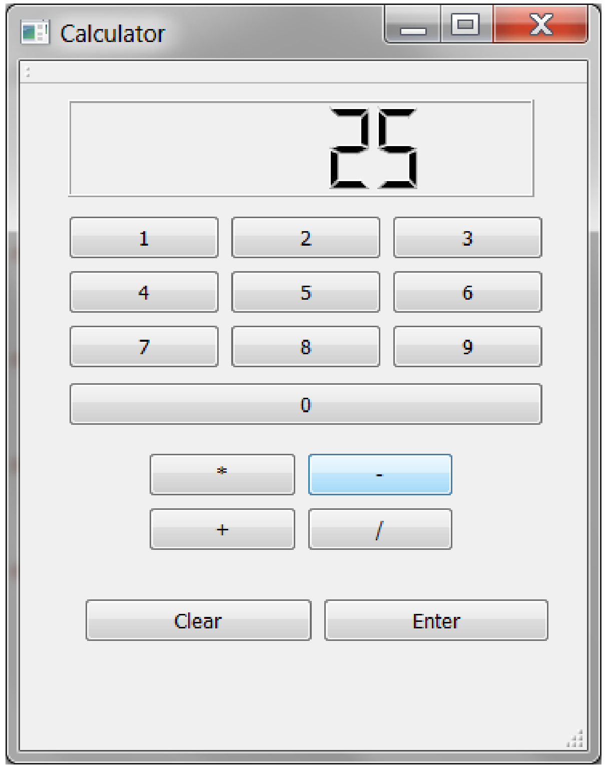
- At last we will add the functionality to the “clear” and “enter” buttons, the clear button will clear the display when clicked and the enter button will perform the selected operation and update the display value.



```
void Calculator::on_pushButton_clear_clicked()
{
    numA = 0;
    numB = 0;
    ui->lcdNumber->display(0);
}

void Calculator::on_pushButton_enter_clicked()
{
    numB = ui->lcdNumber->value();
    switch (oper) {
    case Add:
        ui->lcdNumber->display(numA + numB);
        break;
    case Sub:
        ui->lcdNumber->display(numA - numB);
        break;
    case Mul:
        ui->lcdNumber->display(numA * numB);
        break;
    case Div:
        ui->lcdNumber->display(numA / numB);
        break;
    default:
        break;
    }
}
```

Building a calculator

- Click on the run button and you should be able to see the calculator on your display.

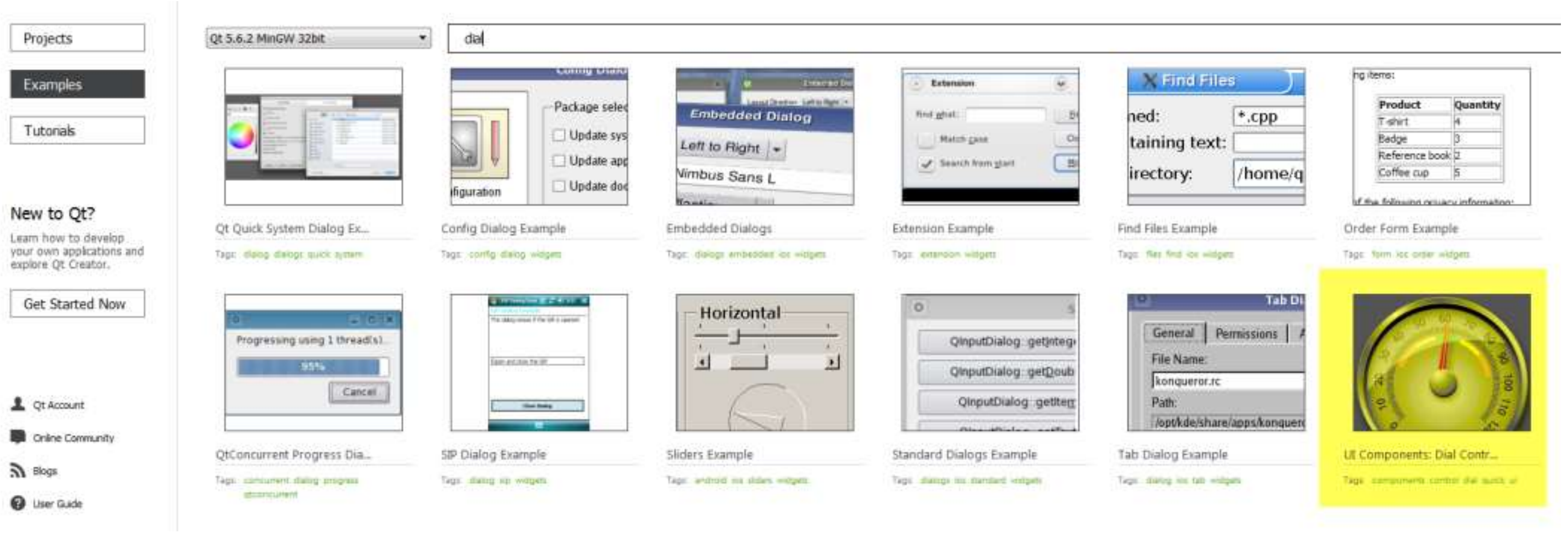


05.

Building a thermometer

Modifying an example

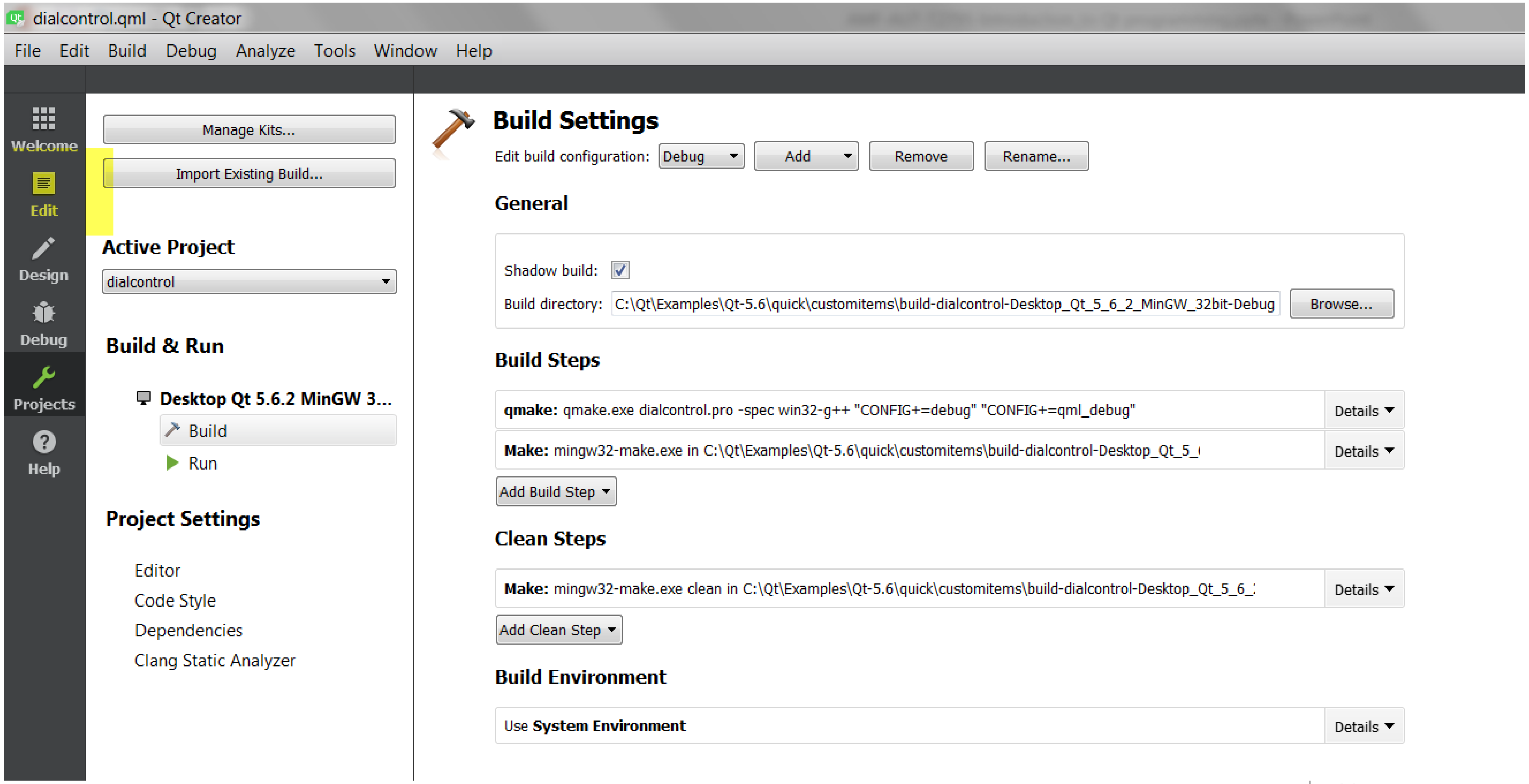
- In this section we are going to modify one of the many available examples.
- Search for the dial example and click on the example that appears at the bottom right of your screen.



Projects
Examples
Tutorials
New to Qt?
Learn how to develop your own applications and explore Qt Creator.
Get Started Now
Qt Account
Online Community
Blogs
User Guide
Qt 5.6.2 MinGW 32bit
dial
Qt Quick System Dialog Ex...
Tags: dialog dialogs quick system
Config Dialog Example
Tags: config dialog widgets
Embedded Dialogs
Tags: dialogs embedded ios widgets
Extension Example
Tags: extension widgets
Find Files Example
Tags: files find ios widgets
Order Form Example
Tags: form ios order widgets
QtConcurrent Progress Dia...
Tags: concurrent dialog progress qtconcurrent
SIP Dialog Example
Tags: dialog sip widgets
Sliders Example
Tags: android ios sliders widgets
Standard Dialogs Example
Tags: dialogs ios standard widgets
Tab Dialog Example
Tags: dialog ios tab widgets
UI Components: Dial Contr...
Tags: components control dial quick ui

Modifying an example

- Select the Edit view…



dialcontrol.qml - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Welcome
Edit
Design
Debug
Projects
Help
Manage Kits...
Import Existing Build...

Active Project

dialcontrol

Build & Run

Desktop Qt 5.6.2 MinGW 3...

Build

Run

Project Settings

Editor

Code Style

Dependencies

Clang Static Analyzer

Build Settings

Edit build configuration: Debug | Add | Remove | Rename...

General

Shadow build: ✓

Build directory: C:\Qt\Examples\Qt-5.6\quick\customitems\build-dialcontrol-Desktop_Qt_5_6_2_MinGW_32bit-Debug | Browse...

Build Steps

qmake: qmake.exe dialcontrol.pro -spec win32-g++ "CONFIG+=debug" "CONFIG+=qml_debug" | Details

Make: mingw32-make.exe in C:\Qt\Examples\Qt-5.6\quick\customitems\build-dialcontrol-Desktop_Qt_5_ | Details

Add Build Step

Clean Steps

Make: mingw32-make.exe clean in C:\Qt\Examples\Qt-5.6\quick\customitems\build-dialcontrol-Desktop_Qt_5_6_ | Details

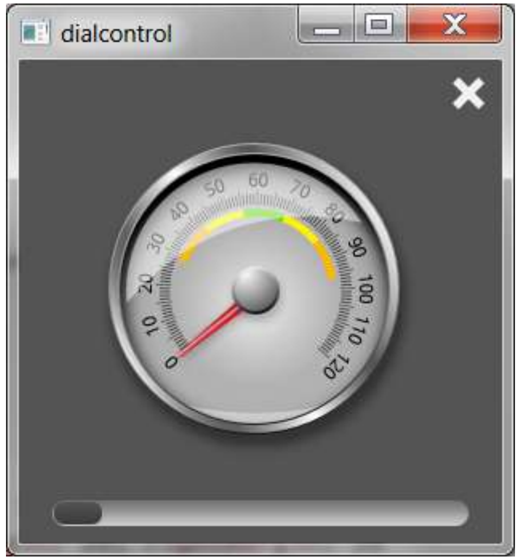
Add Clean Step

Build Environment

Use **System Environment** | Details

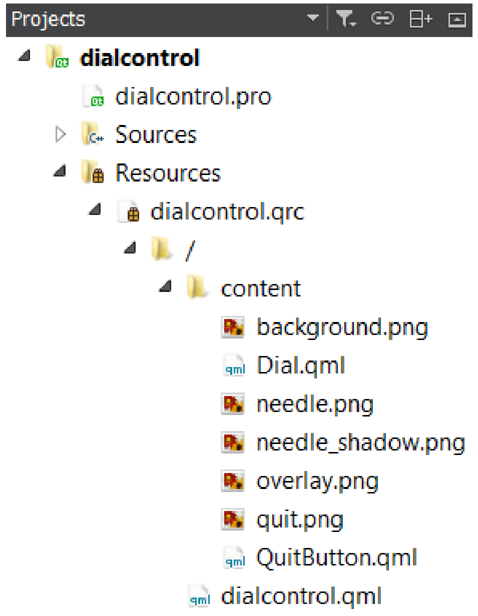
Running the example

- Click on the Run icon to see the original example



Example organization

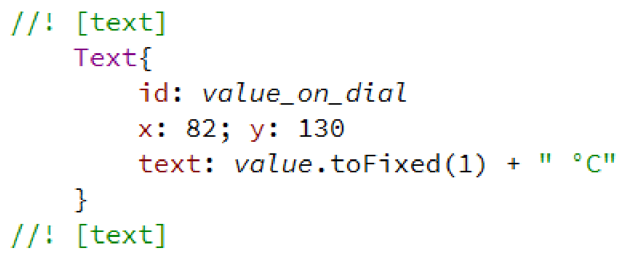
- All the images used to create the dial are stored under Resources/content.
- There is an object called Dial, that features all the functionality of the Dial.



```
Projects
◢ dialcontrol
      dialcontrol.pro
  ▷ Sources
  ◢ Resources
    ◢ dialcontrol.qrc
      ◢ /
        ◢ content
            background.png
            Dial.qml
            needle.png
            needle_shadow.png
            overlay.png
            quit.png
            QuitButton.qml
          dialcontrol.qml
```

Displaying the dial value

- We are going to modify the Dial.qml object to display the value in text.
- Go to the bottom of the Dial.qml file and add the following text object:

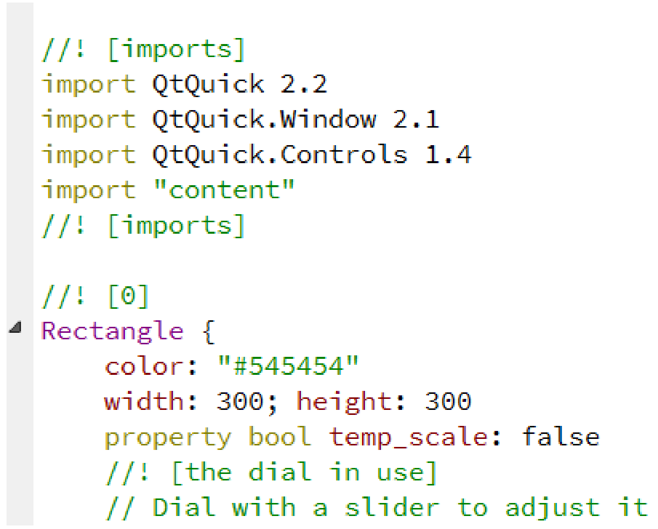


```
//! [text]
    Text{
        id: value_on_dial
        x: 82; y: 130
        text: value.toFixed(1) + " °C"
    }
//! [text]
```

- Run the example, the Dial will now display its value on text.

Switching between Celsius and Fahrenheit

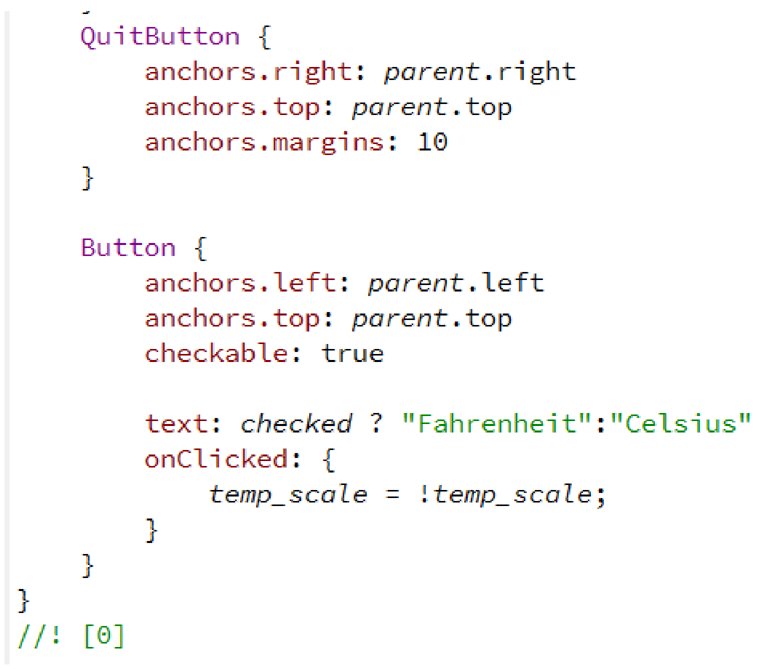
- We will add a button to control whether to display the value on Celsius or Fahrenheit.
- To do so we will modify the dialcontrol.qml file.
- Add the QtQuick.Controls 1.4 library and a variable to select the scale:



```
//! [imports]
import QtQuick 2.2
import QtQuick.Window 2.1
import QtQuick.Controls 1.4
import "content"
//! [imports]

//! [0]
Rectangle {
    color: "#545454"
    width: 300; height: 300
    property bool temp_scale: false
    //! [the dial in use]
    // Dial with a slider to adjust it
```

Switching between Celsius and Fahrenheit

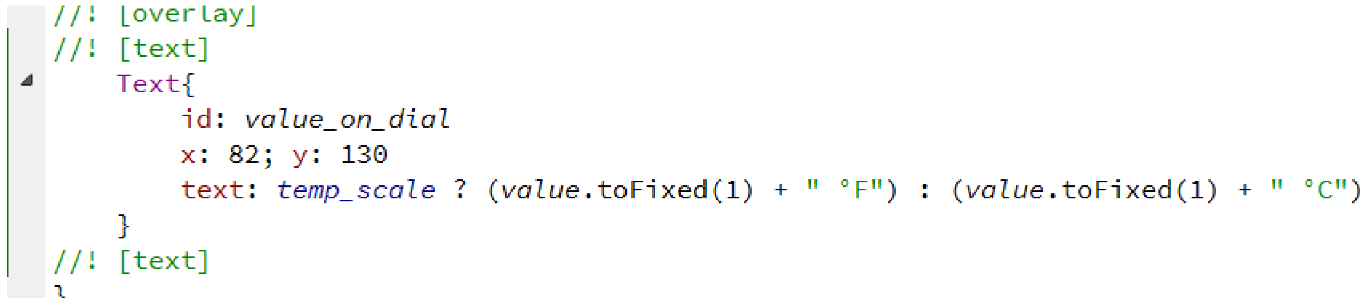
- We will add the button and place at the top left of the dial, the button will display Celsius by default and it will change the label after being clicked.



```
    QuitButton {
        anchors.right: parent.right
        anchors.top: parent.top
        anchors.margins: 10
    }

    Button {
        anchors.left: parent.left
        anchors.top: parent.top
        checkable: true

        text: checked ? "Fahrenheit":"Celsius"
        onClicked: {
            temp_scale = !temp_scale;
        }
    }
}
//! [0]
```

Switching between Celsius and Fahrenheit

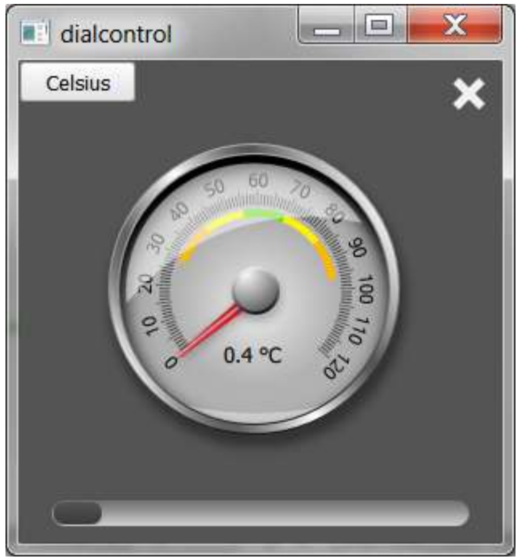
- Now we will modify the Dial.qml file to modify the value on the display accordingly, to do so we only add the following line to our previous modification.

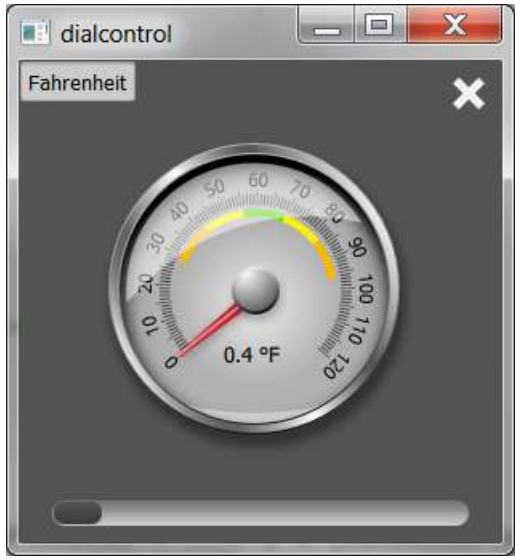


```
//! [overlay]
//! [text]
    Text{
        id: value_on_dial
        x: 82; y: 130
        text: temp_scale ? (value.toFixed(1) + " °F") : (value.toFixed(1) + " °C")
    }
//! [text]
```

Results!

- You can leverage on existing examples to create your application and learn more about Qt!





SECURE CONNECTIONS
FOR A SMARTER WORLD