Qt Quick – Overview and basic GUI

Timo Strömmer, Jan 3, 2011

Contents

Creating a hello world project with QtCreator

QUICK START

Installation

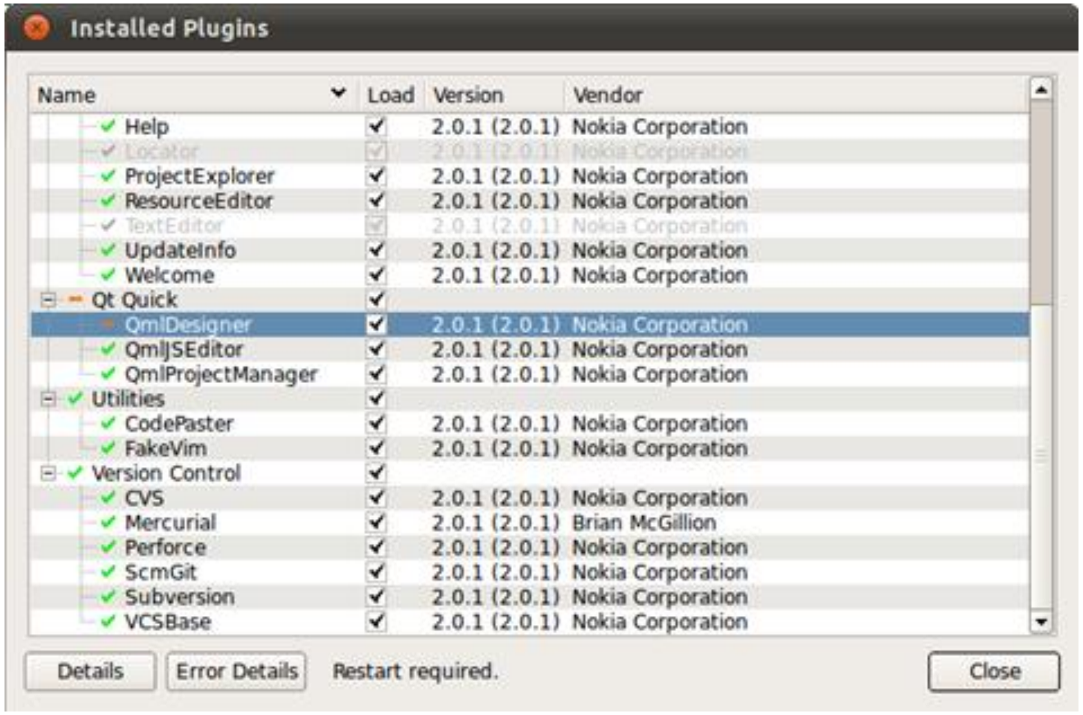
- Qt SDK mess
  - http://qt.nokia.com/downloads/downloads
    - Latest Qt meant for desktop
  - http://www.forum.nokia.com/Develop/Qt/
    - Meant for mobile devices (but not desktop)
    - Only ”preliminary support for Qt 4.7”
  - Will be merged into one product in the future

Installation

- Install Qt 4.7 from Ubuntu repositories
  - Needed, for running in desktop
  - *sudo apt-get install build-essential libqt4-dev qt4-qmlviewer*
- Download and install the forum Nokia version of Nokia Qt SDK
- Run qtcreator
  - ~/NokiaQtSDK/QtCreator/bin/qtcreator

Installation

- Select *Help / About plugins* from menu
  - Enable QmlDesigner and re-start qtcreator



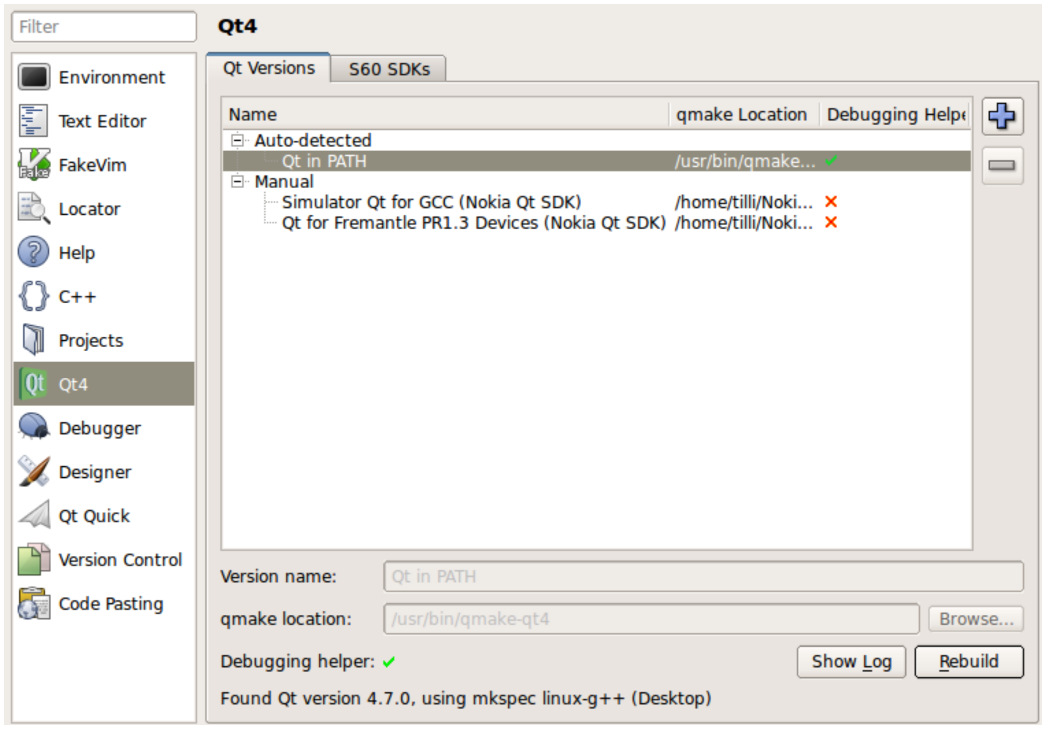
Installed Plugins

Name	Load	Version	Vendor
✓ Help	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ Locator	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ ProjectExplorer	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ ResourceEditor	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ TextEditor	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ UpdateInfo	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ Welcome	✓	2.0.1 (2.0.1)	Nokia Corporation
Qt Quick	✓		
QmlDesigner	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ QmlJSEditor	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ QmlProjectManager	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ Utilities	✓		
✓ CodePaster	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ FakeVim	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ Version Control	✓		
✓ CVS	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ Mercurial	✓	2.0.1 (2.0.1)	Brian McGillion
✓ Perforce	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ ScmGit	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ Subversion	✓	2.0.1 (2.0.1)	Nokia Corporation
✓ VCSBase	✓	2.0.1 (2.0.1)	Nokia Corporation

Details | Error Details | Restart required. | Close

Installation

- Check *Tools / Options* that Qt libraries exist
  - Rebuild debug helper for C++ development



Name	qmake Location	Debugging Helpe
Auto-detected		
Qt in PATH	/usr/bin/qmake...	✓
Manual		
Simulator Qt for GCC (Nokia Qt SDK)	/home/tilli/Noki...	✗
Qt for Fremantle PR1.3 Devices (Nokia Qt SDK)	/home/tilli/Noki...	✗

Version name: Qt in PATH

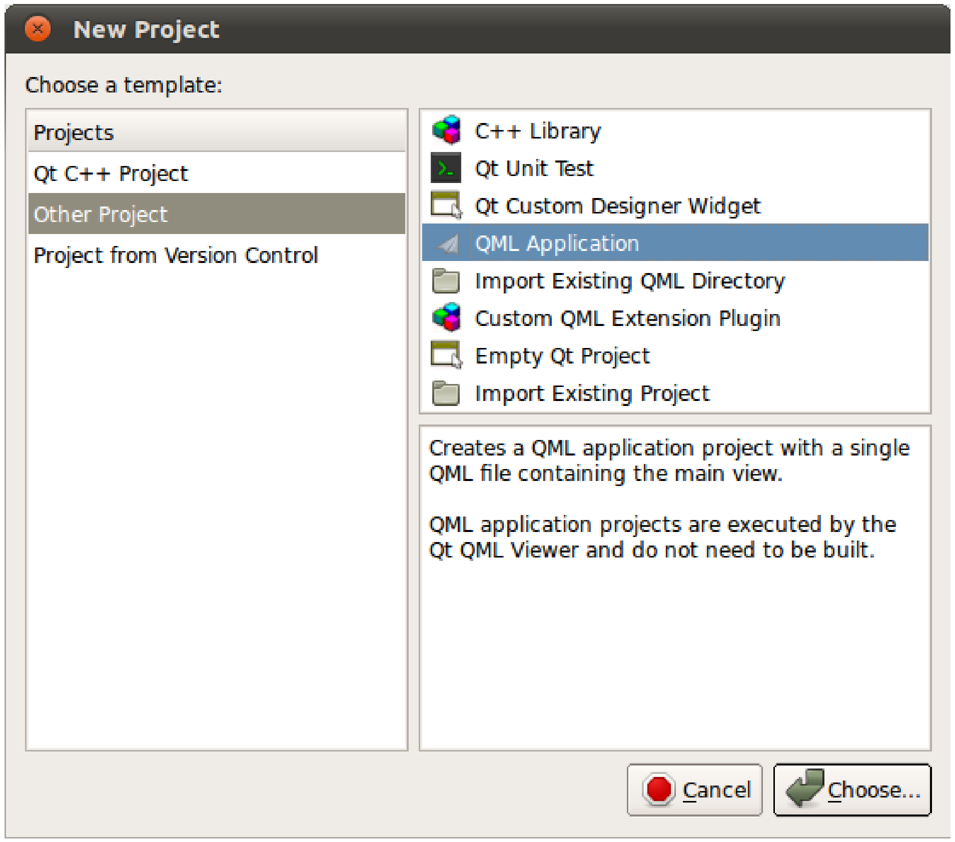
qmake location: /usr/bin/qmake-qt4

Debugging helper: ✓

Found Qt version 4.7.0, using mkspec linux-g++ (Desktop)

Quick start

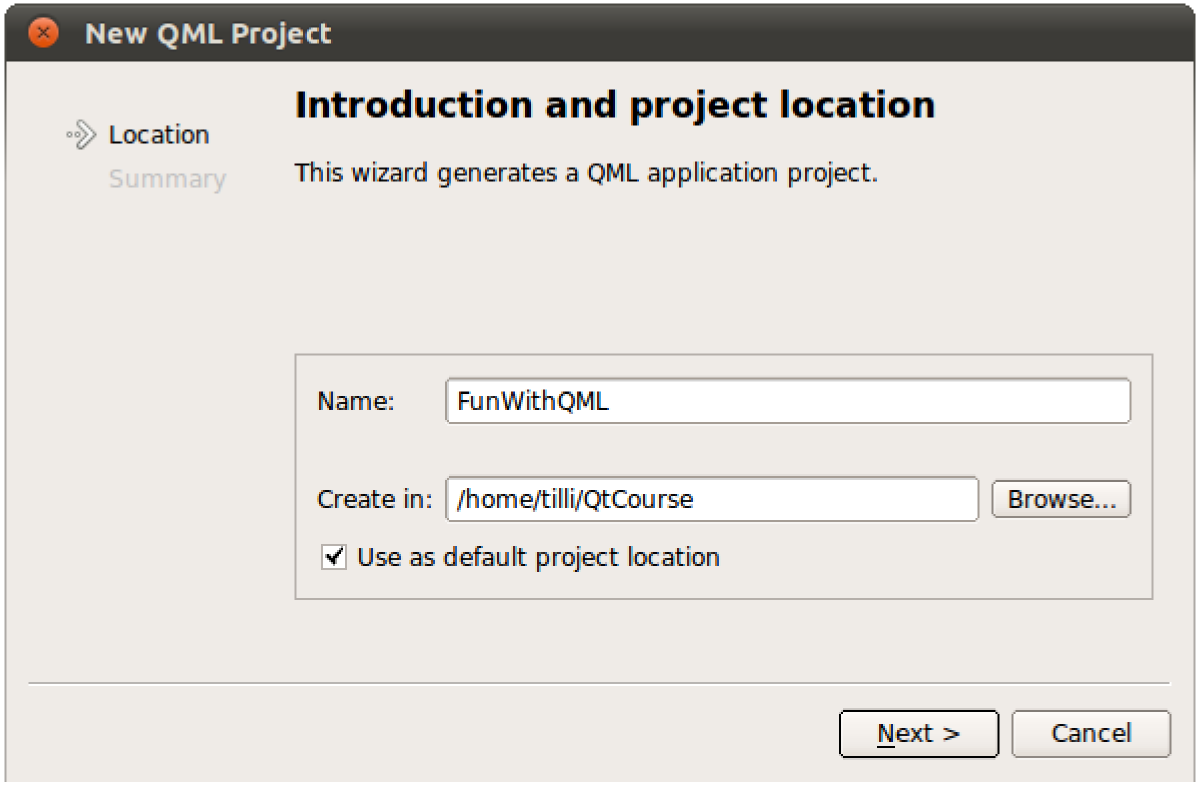
- Select *File / New File or Project*



Creates a QML application project with a single QML file containing the main view.

QML application projects are executed by the Qt QML Viewer and do not need to be built.

Quick start


New QML Project

Location
Summary

Introduction and project location

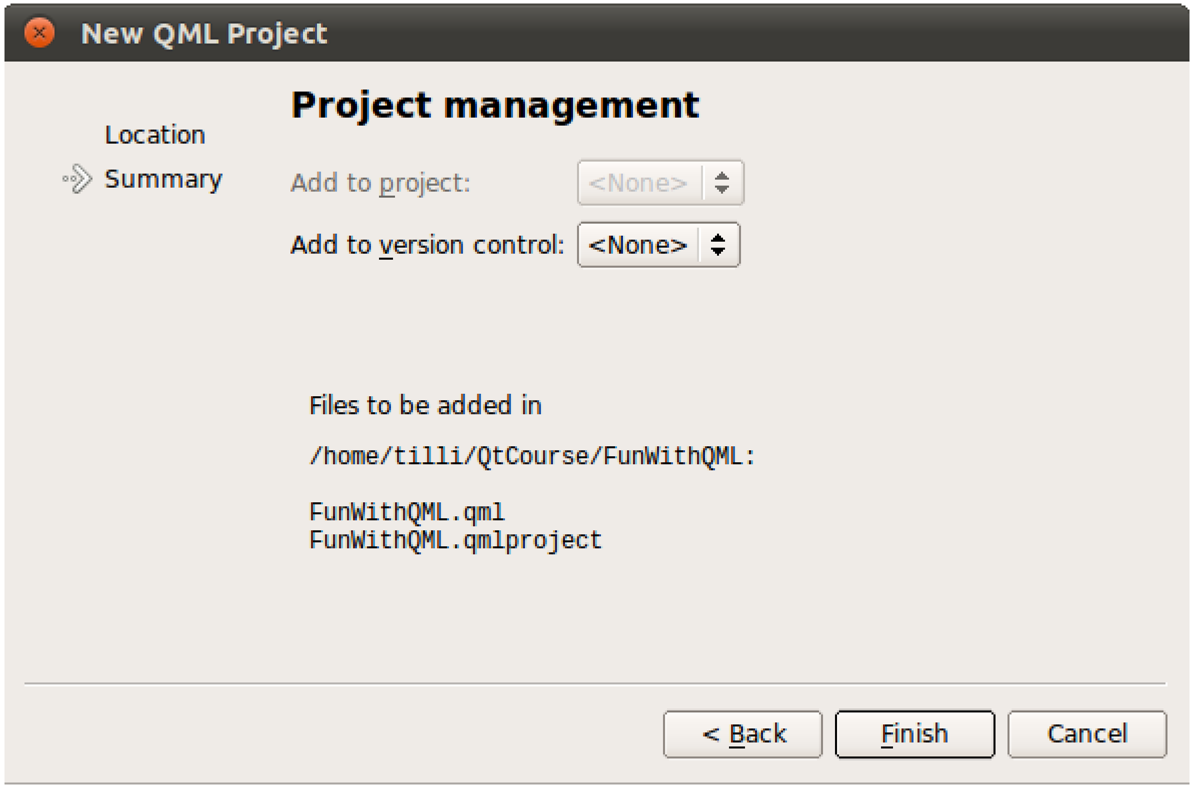
This wizard generates a QML application project.

Name: FunWithQML

Create in: /home/tilli/QtCourse

[✓] Use as default project location

Quick start



Project management

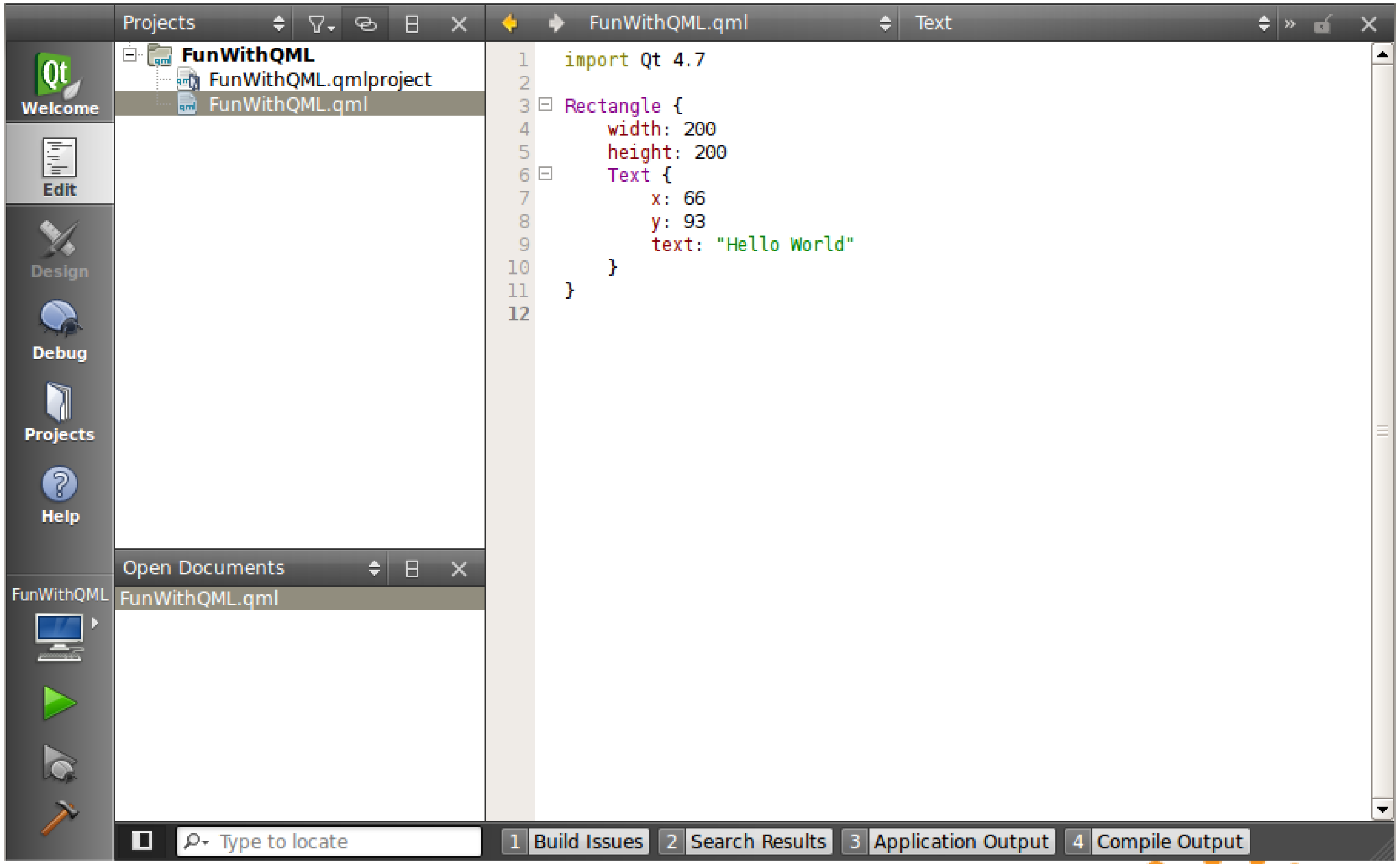
Add to project: <None>

Add to version control: <None>

Files to be added in

```
/home/tilli/QtCourse/FunWithQML:

FunWithQML.qml
FunWithQML.qmlproject
```

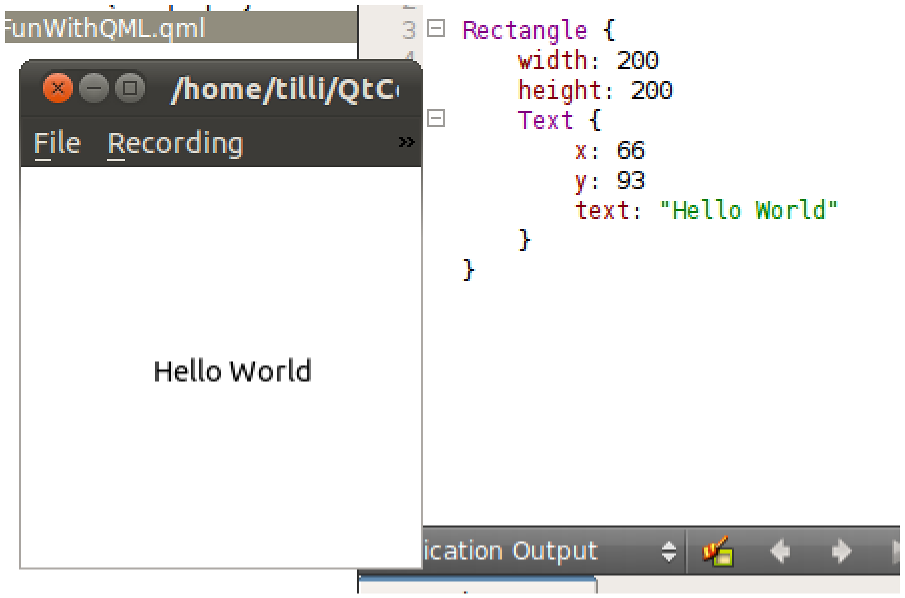
Quick start



```
import Qt 4.7

Rectangle {
      width: 200
      height: 200
      Text {
            x: 66
            y: 93
            text: "Hello World"
      }
}
```

Quick start

- Run the program with Ctrl+R



```
Rectangle {
    width: 200
    height: 200
    Text {
        x: 66
        y: 93
        text: "Hello World"
    }
}
```

Excercise

- Try it out, create and run a QML application project
  - Add some other text entries
  - Optional: Add an image

Overview

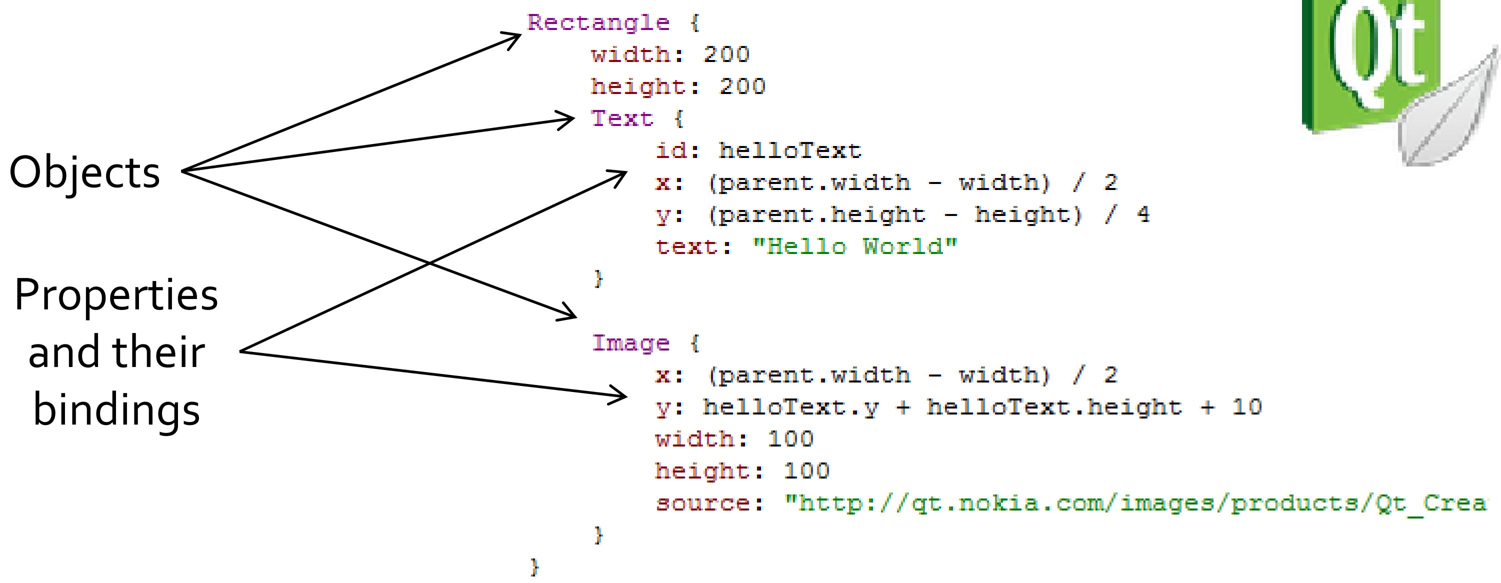
QT QUICK

What is Qt Quick

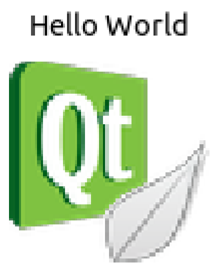
- QML – a language for UI design and development
- Qt declarative – Module for integrating QML and Qt C++ libraries
- Qt Creator tools – Complete development environment
  - QML design and code
  - C++ integration
  - Packaging and deployment

QML overview

- JavaScript-based *declaractive* language
  - Expressed as *bindings* between *properties* that are *structured* into *object tree*



```
Rectangle {
    width: 200
    height: 200
    Text {
        id: helloText
        x: (parent.width - width) / 2
        y: (parent.height - height) / 4
        text: "Hello World"
    }
    Image {
        x: (parent.width - width) / 2
        y: helloText.y + helloText.height + 10
        width: 100
        height: 100
        source: "http://qt.nokia.com/images/products/Qt_Crea
    }
}
```



QML overview

- Contrast with an *imperative language*

Property bindings are statements that get evaluated whenever property changes

```
Rectangle {
    width: 200
    height: 200
    Text {
        id: helloText
        x: (parent.width - width) / 2
        y: (parent.height - height) / 4
        text: "Hello World"
    }
}
```

Statements are evaluated once

```
Rectangle r = new Rectangle();
r.setWidth(200);
r.setHeight(200);
Text helloText = new Text();
helloText.setParent(r);
helloText.setText("Hello World");
helloText.setX((r.width() - helloText.width()) / 2);
helloText.setY((r.height() - helloText.height()) / 4);
```

QML overview

- JavaScript / JSON, not XML
    - unlike MXML (Flash), XUL (Gecko), XAML (.Net)
    - But, has support for XPath queries, so can easily integrate with XML-based web services

Qt Declarative

- Declarative module is a C++ framework for gluing QML and C++ code together
  - Integrating QML ”scripts” into C++ application
  - Integrating C++ plug-in’s into QML application
- Still lacking some basics
  - First official version with Qt4.7 (2010/09/21)
  - GUI component project in development
    - Buttons, dialogs etc.

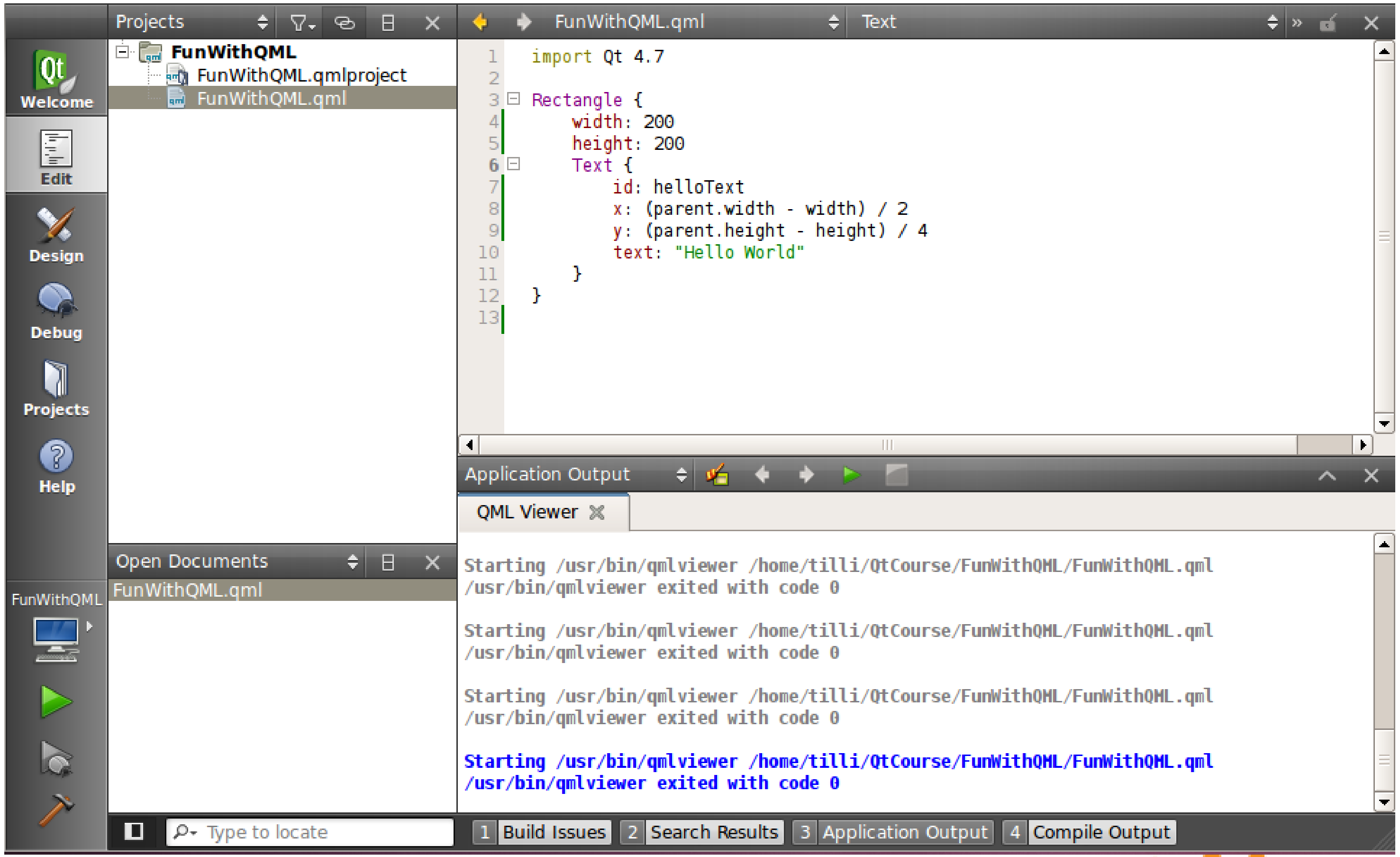
Qt Creator

- Qt Creator integrates C++ and QML development into single IDE
  - QML designer for visual editing
  - QML and C++ code editors
  - Same code can be run at desktop or device

Qt Creator intro

- This is interactive part…
  - QML editor
  - QML designer
  - Project management
  - Session management

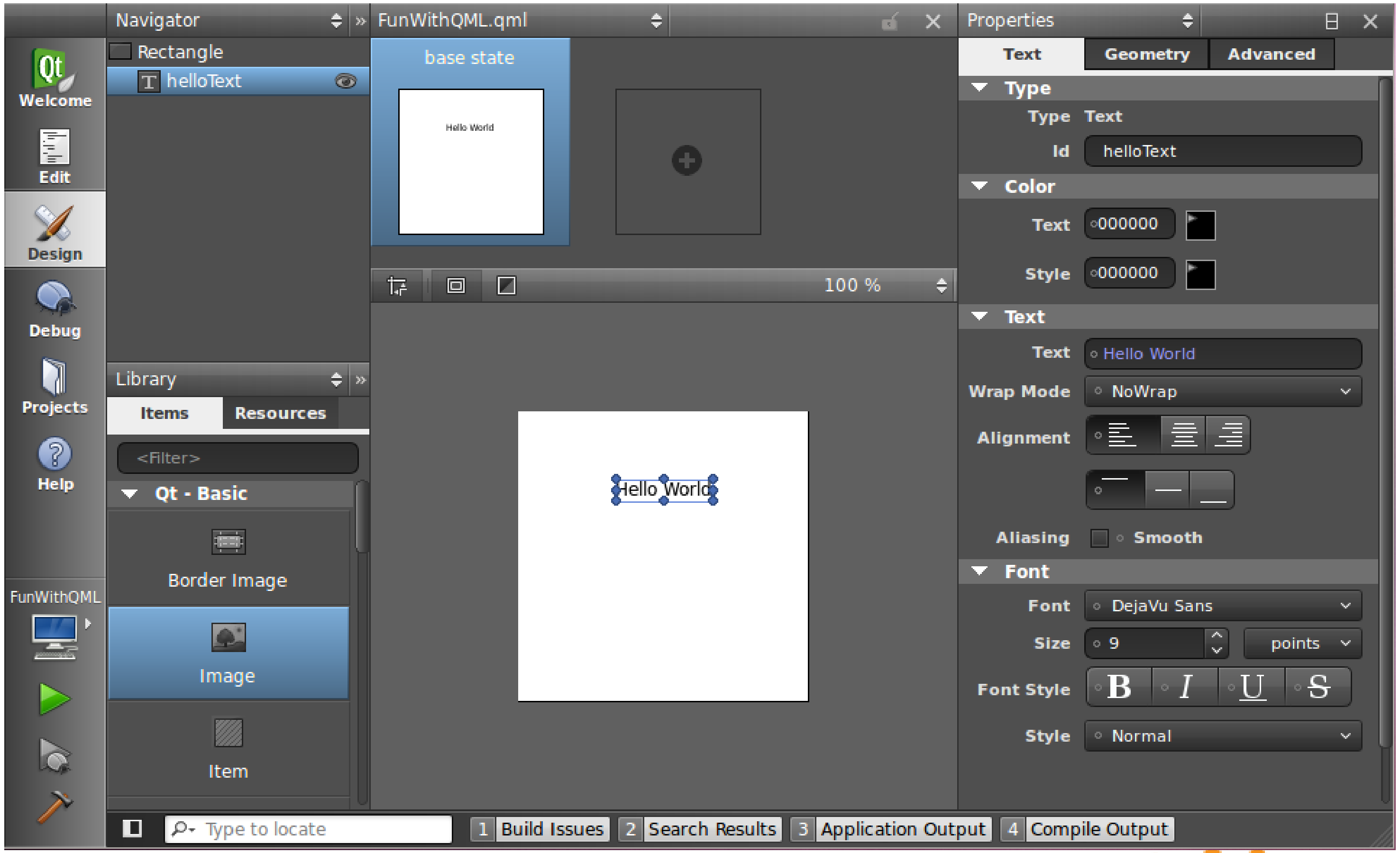
QML editor



```
import Qt 4.7

Rectangle {
    width: 200
    height: 200
    Text {
        id: helloText
        x: (parent.width - width) / 2
        y: (parent.height - height) / 4
        text: "Hello World"
    }
}
```

```
Starting /usr/bin/qmlviewer /home/tilli/QtCourse/FunWithQML/FunWithQML.qml
/usr/bin/qmlviewer exited with code 0

Starting /usr/bin/qmlviewer /home/tilli/QtCourse/FunWithQML/FunWithQML.qml
/usr/bin/qmlviewer exited with code 0

Starting /usr/bin/qmlviewer /home/tilli/QtCourse/FunWithQML/FunWithQML.qml
/usr/bin/qmlviewer exited with code 0

Starting /usr/bin/qmlviewer /home/tilli/QtCourse/FunWithQML/FunWithQML.qml
/usr/bin/qmlviewer exited with code 0
```

QML designer



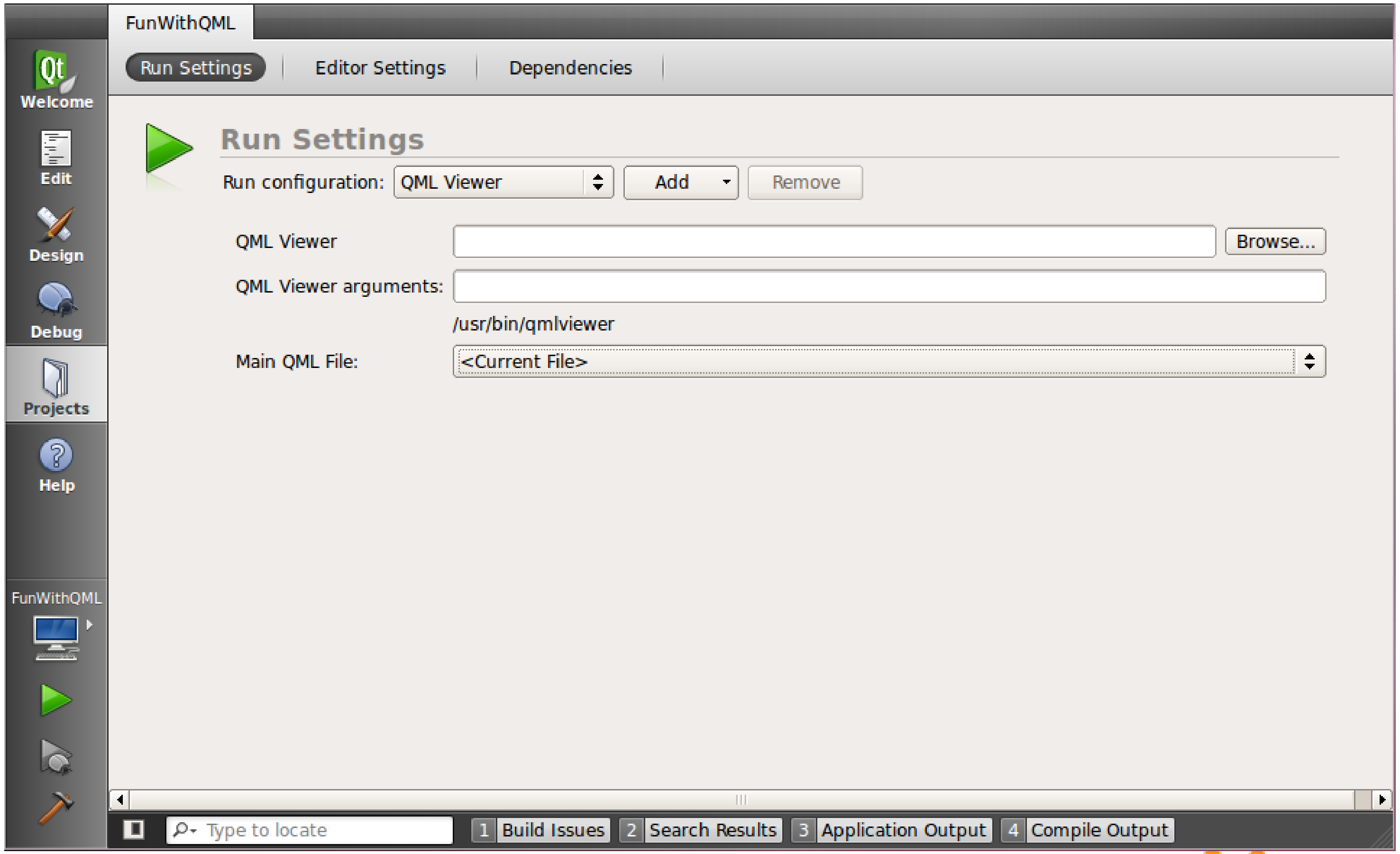
Type
Type: Text
Id: helloText

Color
Text: 000000
Style: 000000

Text
Text: Hello World
Wrap Mode: NoWrap
Alignment
Aliasing: Smooth

Font
Font: DejaVu Sans
Size: 9 points
Font Style: B I U S
Style: Normal

QML project properties



Run Settings

Run configuration: QML Viewer

QML Viewer:

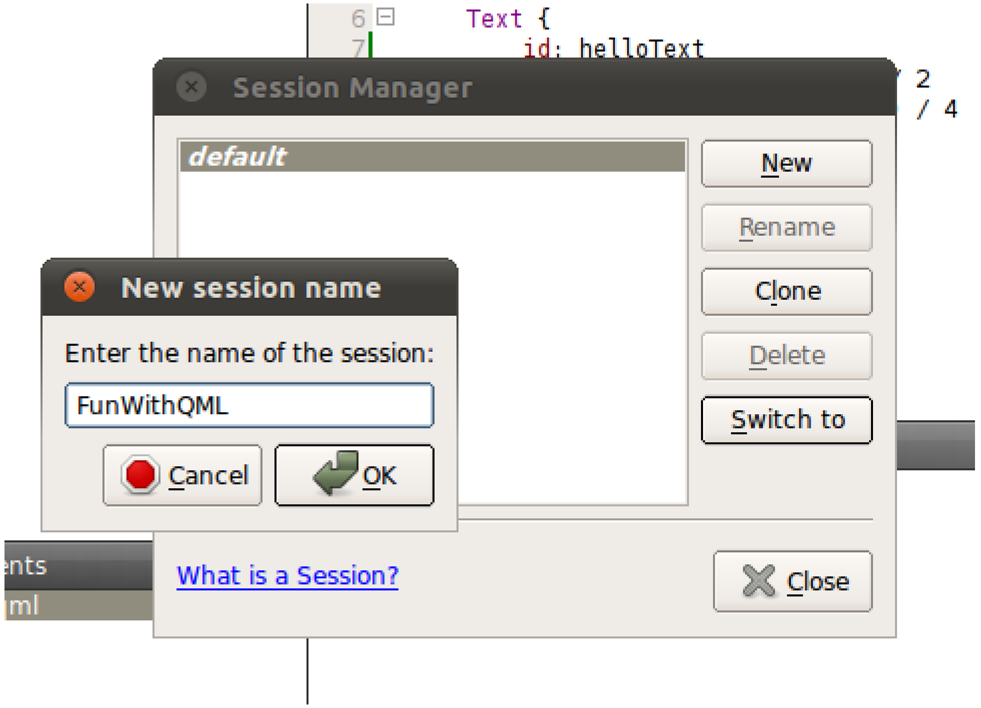
QML Viewer arguments:

/usr/bin/qmlviewer

Main QML File: <Current File>

Session management

- File -> Sessions -> Session Manager

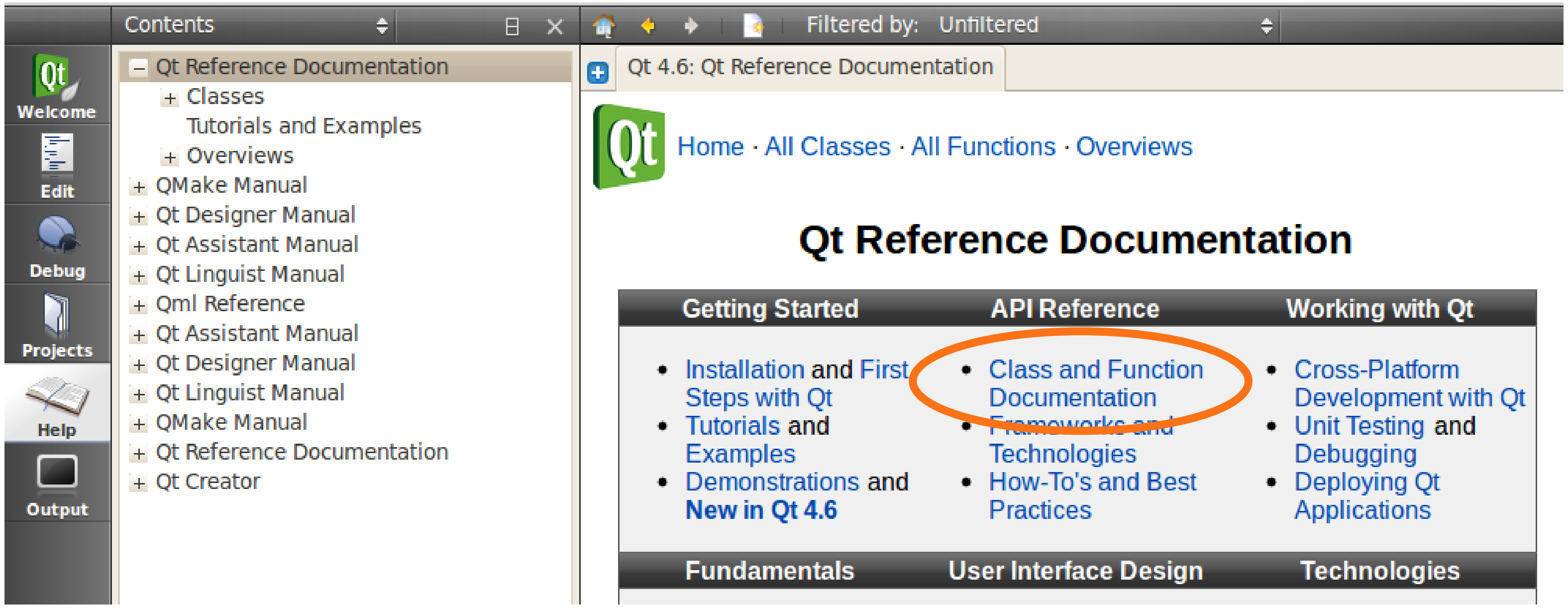


Overview of what's in there

QT MODULES

Qt modules walkthrough

- Qt documentation integrated to QtCreator
  - API reference -> Class and Function Documentation -> All Qt Modules



Qt Reference Documentation

Getting Started	API Reference	Working with Qt
• Installation and First Steps with Qt • Tutorials and Examples • Demonstrations and New in Qt 4.6	• Class and Function Documentation • Frameworks and Technologies • How-To's and Best Practices	• Cross-Platform Development with Qt • Unit Testing and Debugging • Deploying Qt Applications
Fundamentals	User Interface Design	Technologies

Core module

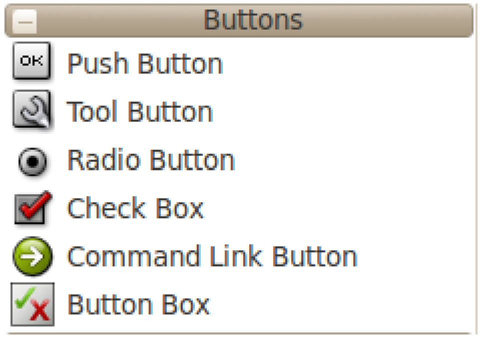
- Frameworks discussed during this course
    - Qt object model (QObject, QMetaObject)
    - Strings (QString, QByteArray)
    - Containers (QList, QMap, QHash, QLinkedList)
    - Data models (QAbstractItemModel & related)

Core module

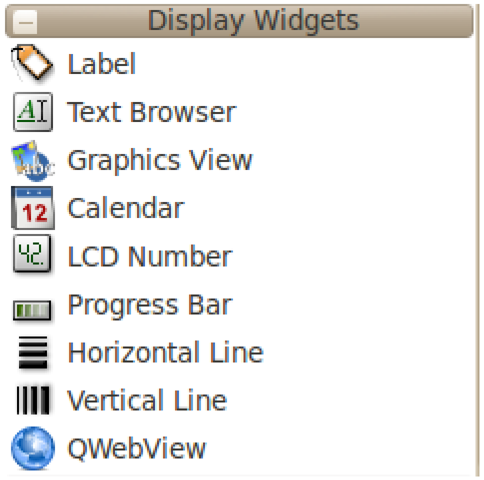
- Frameworks not discussed in this course
  - Multithreading (QFuture & related)
  - I/O devices (QIODevice, Qfile & related)
  - State machines (QStateMachine & related)

GUI module

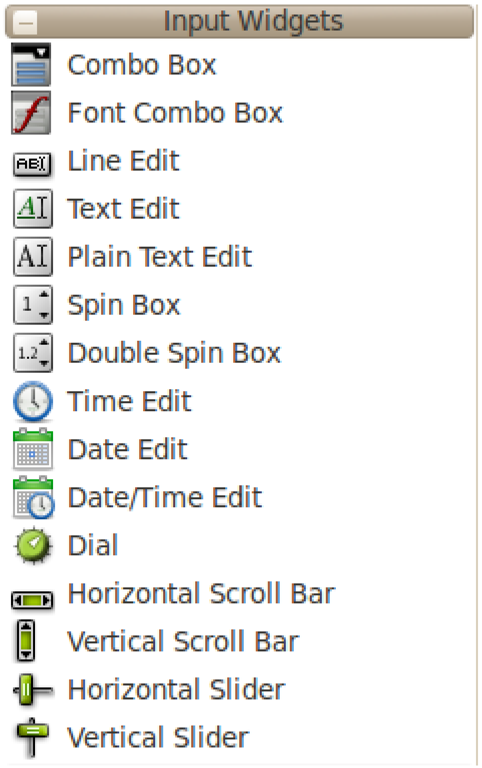
- ”Traditional” widgets



Buttons
- Push Button
- Tool Button
- Radio Button
- Check Box
- Command Link Button
- Button Box



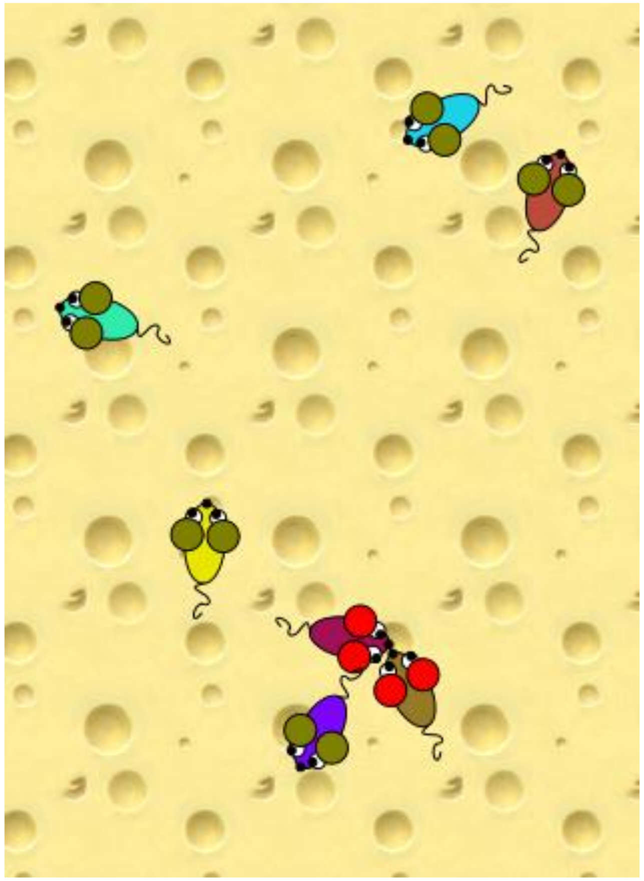
Display Widgets
- Label
- Text Browser
- Graphics View
- Calendar
- LCD Number
- Progress Bar
- Horizontal Line
- Vertical Line
- QWebView



Input Widgets
- Combo Box
- Font Combo Box
- Line Edit
- Text Edit
- Plain Text Edit
- Spin Box
- Double Spin Box
- Time Edit
- Date Edit
- Date/Time Edit
- Dial
- Horizontal Scroll Bar
- Vertical Scroll Bar
- Horizontal Slider
- Vertical Slider

GUI module

- Graphics view
  - Graphics items
  - Graphics widgets
  - Proxy widgets
- This course focuses on the QML-side, not C++ graphics framework



Network module

- Sockets, including secure ones
  - QTcpSocket, QSslSocket
- Simple HTTP and FTP API’s
  - QNetworkAccessManager

Multimedia modules

- OpenGL for 3D rendering
- OpenVG for 2D rendering
- Svg for processing vector graphics files
- Phonon multimedia framework
  - Not in mobile devices

Scripting modules

- QtDeclarative and QtScript
  - QtScript -> Basically QML without the declarative parts
  - Different C++ engines
- QtDeclarative gets the hype nowadays

Other modules

- XML
    - SAX and DOM parsers
- XmlPatterns
    - XPath, XQuery, XSLT, schemas
- WebKit browser engine
- SQL for accessing databases

Mobile development

- Mobility API’s are not part of standard QT
  - GPS, contacts, calendar, messaging, etc.
  - Latest release 1.1:
    - http://qt.nokia.com/products/qt-addons/mobility/
    - Symbian .sis packages available for download
    - N900 package can be installed from repository
      - apt-get install libqt4-mobility
    - Works in Qt Simulator on PC
  - QML integration in progress

Basic concepts

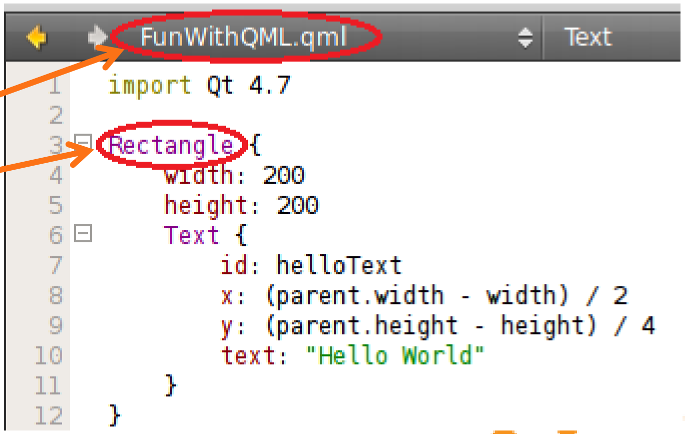
QML PROGRAMMING

QML syntax

- Based on ECMA-262 specification
  - Operating environment differs from the usual web browser
    - DOM vs. QtDeclarative
  - Supports v5 features (notably JSON)
- Declarative concepts added on top
  - Quite a lot can be done without any ”scriptiness”

Components

- A QML document (*.qml* file) describes the structure of one *Component*
  - Component name is file name
    - Name follows camel-case conventions
  - Components have inheritance hierarchy



```
import Qt 4.7

Rectangle {
    width: 200
    height: 200
    Text {
        id: helloText
        x: (parent.width - width) / 2
        y: (parent.height - height) / 4
        text: "Hello World"
    }
}
```

FunWithQML extends Rectancle

Components

- An *instance* of a component is created when the program is run

Creates *FlipText* and *MouseArea* objects as children of *Rectangle*

id property is used when referencing instances

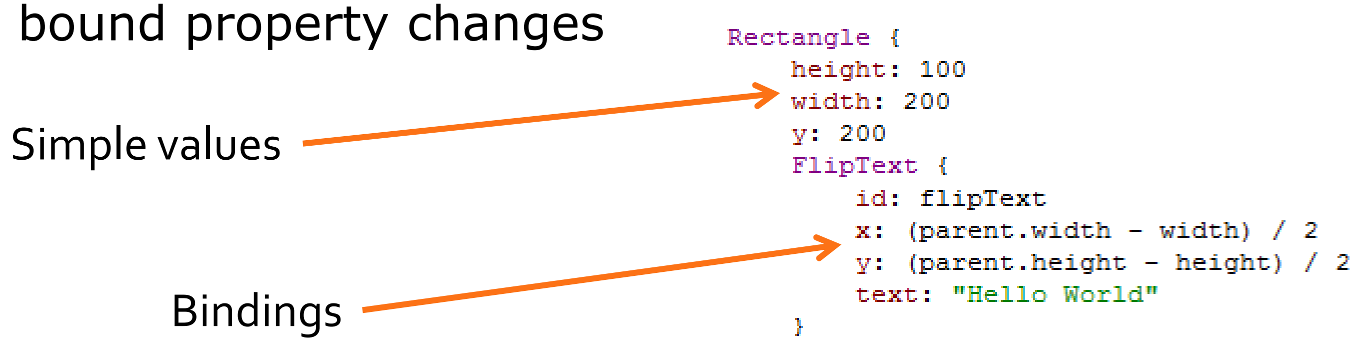
```
Rectangle {
    height: 100
    width: 200
    y: 200
    FlipText {
        id: flipText
        x: (parent.width - width) / 2
        y: (parent.height - height) / 2
        text: "Hello World"
    }
    MouseArea {
        anchors.fill: parent
        onClicked: flipText.flip()
    }
}
```

Components

- Internals of component are not automatically visible to other components
- Component’s API is defined via *properties, functions* and *signals*:
  - *Property* - expression that evaluates to a value
  - *Function* - called to perform something
  - *Signal* - callback from the component

Properties

- Properties can be referenced by name
  - Always starts with lower-case letter
- A property expression that references another property establishes a *binding*
  - Whenever the referenced property changes, the bound property changes



```
Rectangle {
    height: 100
    width: 200
    y: 200
    FlipText {
        id: flipText
        x: (parent.width - width) / 2
        y: (parent.height - height) / 2
        text: "Hello World"
    }
```

Properties

- The basics of properties:
  - *id* is used to reference an object
  - *list* properties are a collection of elements
  - *default* property can be used without a name
    - The *data* list in following example

```
Rectangle {
    height: 100
    width: 200
    y: 200
    data: [
        FlipText { /*...*/ },
        MouseArea { /*...*/ },
        Timer { /*...*/ },
        HelloSignal { /*...*/ }
    ]
}
```

```
Rectangle {
    height: 100
    width: 200
    y: 200
    FlipText { /*...*/ }
    MouseArea { /*...*/ }
    Timer { /*...*/ }
    HelloSignal { /*...*/ }
}
```

Properties

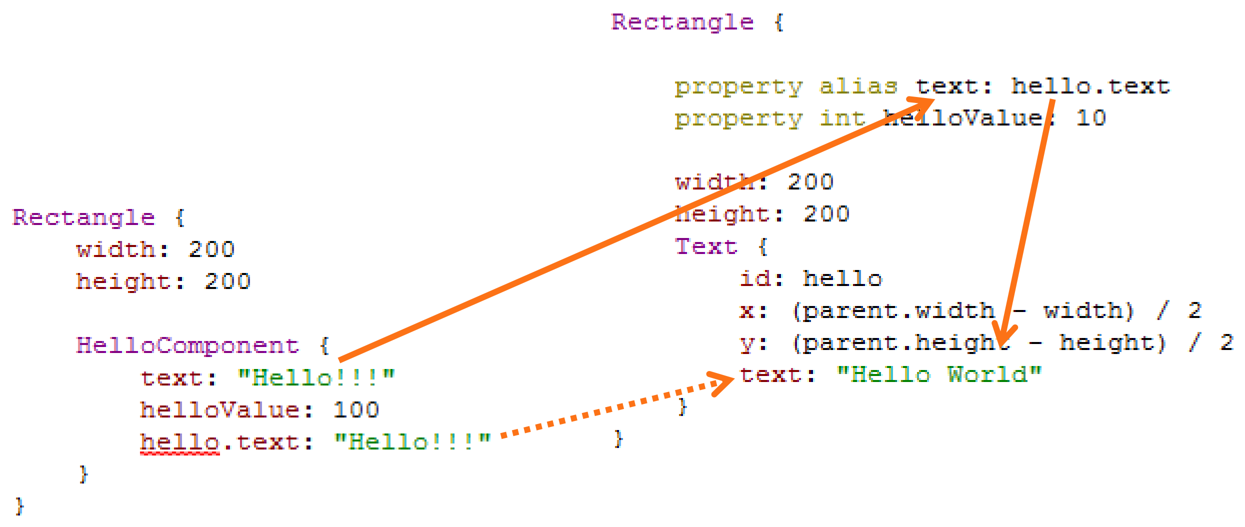
- Public properties are specified with *property* syntax
  - *Value* properties, for example:
    - *int, bool, real, string*
    - *point, rect, size*
    - *time, date*

```
Rectangle {

    property alias text: hello.text
    property int helloValue: 10

    width: 200
    height: 200
    Text {
        id: hello
        x: (parent.width - width) / 2
        y: (parent.height - height) / 2
        text: "Hello World"
    }
}
```

http://doc.qt.nokia.com/4.7/qdeclarativebasictypes.html

Alias properties

- Property *alias* exposes an internal property to public API of component



```
Rectangle {
    width: 200
    height: 200

    HelloComponent {
        text: "Hello!!!"
        helloValue: 100
        hello.text: "Hello!!!"
    }
}
```

```
Rectangle {

    property alias text: hello.text
    property int helloValue: 10

    width: 200
    height: 200
    Text {
        id: hello
        x: (parent.width - width) / 2
        y: (parent.height - height) / 2
        text: "Hello World"
    }
}
```

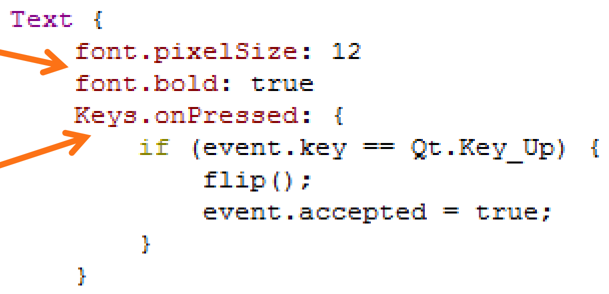
Not working directly

Properties

- Properties can be *grouped* or *attached*
    - Both are referenced with '.' notation
    - Grouping and attaching is done on C++ side, not within QML

font contains a group of Properties related to the font of the text field

All properties of *Keys* component have been attached to Text and can be used by '.' notation



```
Text {
    font.pixelSize: 12
    font.bold: true
    Keys.onPressed: {
        if (event.key == Qt.Key_Up) {
            flip();
            event.accepted = true;
        }
    }
```

Signals

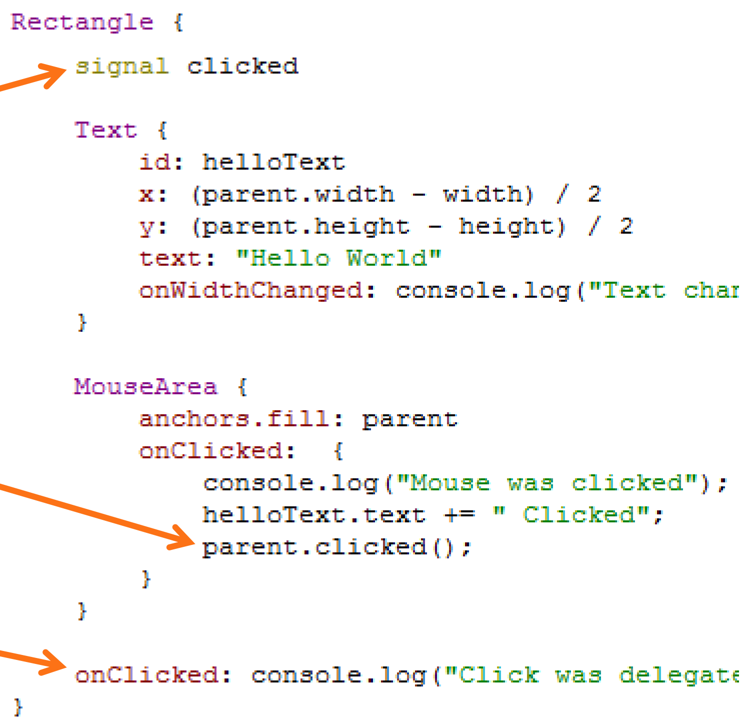
- A component may emit signals, which are processed in *signal handlers*
  - Signal handlers follow *onSignalName* syntax

```
MouseArea {
    anchors.fill: parent
    onClicked:  {
        console.log("Mouse was clicked");
        helloText.text += " Clicked";
        parent.clicked();
    }
}
```

Mouse click
signal handler

Signals

- Signals can be defined with *signal* keyword



```
Rectangle {
    signal clicked

    Text {
        id: helloText
        x: (parent.width - width) / 2
        y: (parent.height - height) / 2
        text: "Hello World"
        onWidthChanged: console.log("Text char
    }

    MouseArea {
        anchors.fill: parent
        onClicked:  {
            console.log("Mouse was clicked");
            helloText.text += " Clicked";
            parent.clicked();
        }
    }

    onClicked: console.log("Click was delegate
}
```

Custom signal

Calling the signal

Custom signal handler

Functions

- A component may export functions that can be called from other components
  - Note: Not *declarative* way of doing things
    - Destroys property bindings

```
Rectangle {
    height: 100
    width: 200
    y: 200
    FlipText {
        id: flipText
        x: (parent.width - width) / 2
        y: (parent.height - height) / 2
        text: "Hello World"
    }
    MouseArea {
        anchors.fill: parent
        onClicked: flipText.flip()
    }
}
```

```
Text {
    rotation: parent.rotation

    function flip() {
        if (rotation == 0) {
            rotation = 180
            text = "Hello World Upside Down"
        } else {
            rotation = 0
            text = "Hello World"
        }
    }
}
```

Building a GUI

QML ELEMENTS

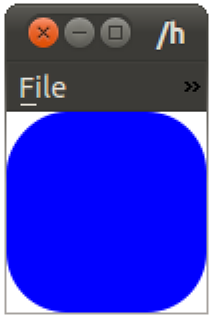
QML Item

- *Item* is a base for all GUI components
- Basic properties of an GUI item:
  - Coordinates*: x, y, z, width, height, anchors*
  - Transforms*: rotation, scale, translate*
  - Hierarchy: *children, parent*
  - Visibility: *visible, opacity*
  - *state* and *transitions*
- Does not draw anything by itself

Basic visual elements

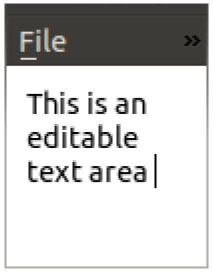
- *Rectangle* and *Image*
  - Basic building blocks
  - *Image* can be loaded from web
- *Text*, *TextInput* and *TextEdit*
  - For non-editable, single-line editable and multiline editable text areas
- And that's about it ☺
  - Qt components project is in progress

```
import Qt 4.7

Rectangle {
    width: 100
    height: 100
    color: "blue"
    radius: 30
}
```



```
Image {
    width: 100
    height: 100
    source: "http://qt.n
}
```

```
Rectangle {
    width: 100
    height: 100

    TextEdit {
        anchors.fill: parent
        anchors.margins: 10
        wrapMode: TextEdit.WordWrap
    }
```



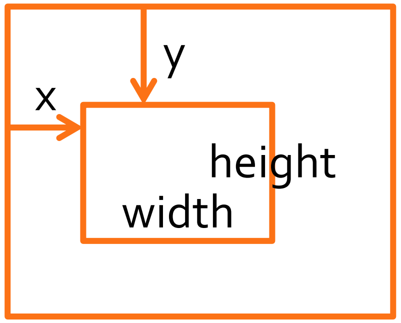
This is an editable text area

Item layouts

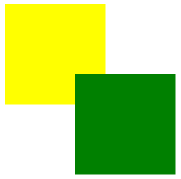
- Relative coordinates
- *Anchors* between items
- *Positioner* objects
  - *Row*, *Column*, *Flow*, *Grid*

Item coordinates

- Position is defined by *x* and *y*
    - Relative to parent item
- Size is defined by *width* and *height*

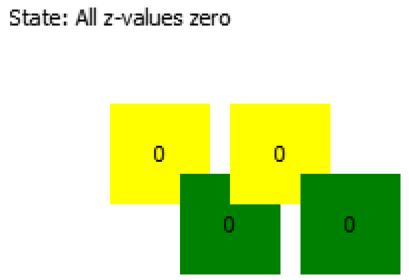


```
Rectangle {
    id: parentRect
    color: "yellow"
    x: 50; y: 50; width: 50; height: 50
    Rectangle {
        id: childRect
        color: "green"
        x: 35; y: 35; width: 50; height: 50
    }
}
```



Item coordinates

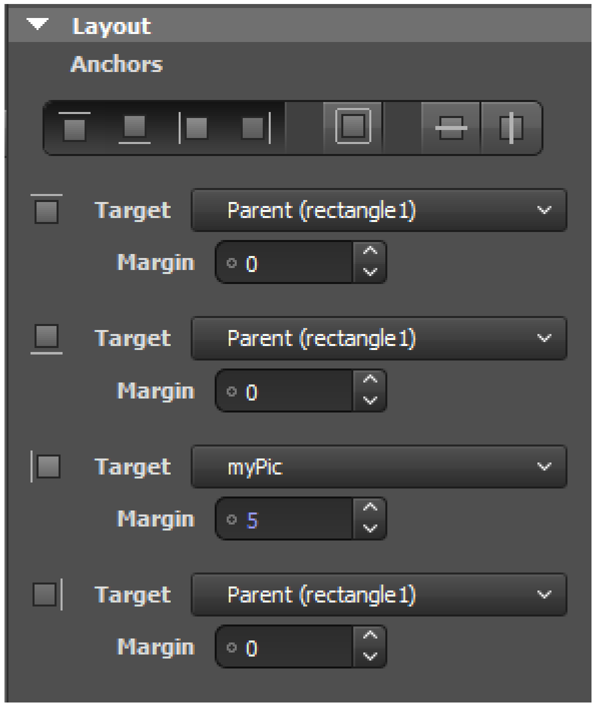
- *z* defines how overlapping areas are drawn
- Example in *Coordinates* directory


State: All z-values zero

Item anchors

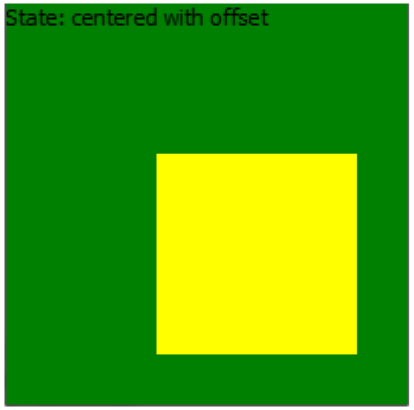
- Each GUI item has 6 *anchor lines* (+1 for text)
  - Side anchors:
    - *top, bottom, left, right*
    - *fill* special anchor
  - Center anchors:
    - *verticalCenter, horizontalCenter*
  - Text has *baseline* anchor

```
Rectangle {
    id: rectangle2
    color: "blue"
    anchors.left: myPic.right
    anchors.right: parent.right
    anchors.bottom: parent.bottom
    anchors.top: parent.top
    anchors.leftMargin: 5
```


Layout
Anchors
Target Parent (rectangle1)
Margin 0
Target Parent (rectangle1)
Margin 0
Target myPic
Margin 5
Target Parent (rectangle1)
Margin 0

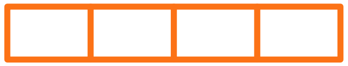
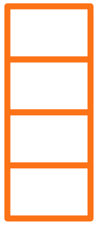
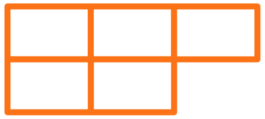
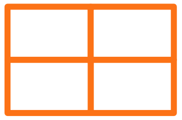
Item anchors

- Anchors may contains spacing
  - Side anchors have *margins*
    - *topMargin, bottomMargin, leftMargin, rightMargin*
    - *margins* special value
  - Center anchors have *offset*
    - *verticalCenterOffset, horizontalCenterOffset*
- Example in *Anchors* directory



Positioners

- Four positioner types:
  - *Row* lays out child items horizontally
  - *Column* lays them vertially
  - *Flow* is either horizontal or vertical
    - *Row* or *Column* with wrapping
  - *Grid* is two-dimensional
- Child item doesn’t need to fill the ”slot”

Positioners

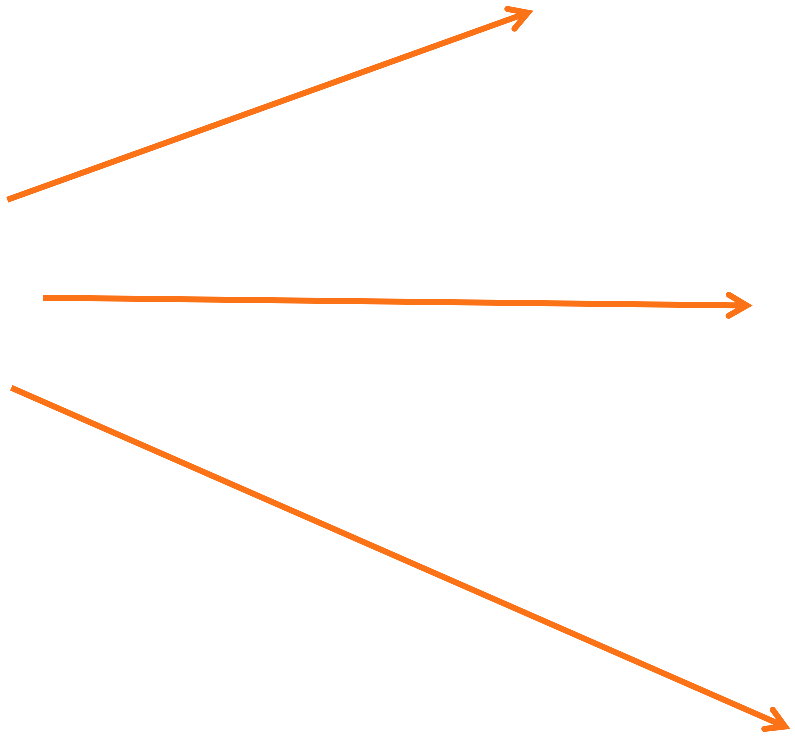
- Positioners inherit from *Item*
  - Thus, have for example anchors of their own
  - Can be nested inside other positioners
- Positioners have *spacing* property
  - Specifies the distance between elements, quite similarly as *margins* of anchors
    - Same spacing for all child item

- Example in *Positioners* directory

Getting started with QML

PROGRAMMING EXERCISE

Day 1 exercise - layouts



References

- JavaScript:
  - https://developer.mozilla.org/en/JavaScript/
- QML elements:
  - http://doc.qt.nokia.com/4.7/qdeclarativeelements.html

SERIOUS ABOUT SOFTWARE