

< symbio >



SERIOUS ABOUT SOFTWARE

Qt Quick – Overview and basic GUI

Timo Strömmer, Jan 3, 2011

Contents



- Quick start
 - Environment installation
- Qt Quick overview
 - Qt Quick components
 - QML language overview
 - Qt modules overview
- Programming with QML
 - Basic concepts and GUI elements
 - GUI layouts



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



What's the difference between the Nokia Qt SDK and the Qt SDK?

The Nokia Qt SDK is a mobile specific version of the [Qt SDK](#) for targeting Nokia devices.

Nokia Qt SDK	Qt SDK	Feature/Component	Comments
✓	✓	Qt source	
✓	✓	Qt Creator	
✗	✓*	Qt binary build - Windows	* Qt SDK for Windows
✗	✓*	Qt binary build - Linux	* Qt SDK for Linux
✗	✓*	Qt binary build - Mac	* Qt SDK for Mac
✓	✗	Qt binary build - Symbian	
✓	✗	Qt binary build - Maemo	
✗*	✗*	Qt binary build - MeeGo	* under roadmap review
✗	✗	Qt binary build - Windows Mobile	
✗	✗	Qt binary build - Windows CE	
✓*	✓	Developer environment - Windows	* Symbian and Maemo support
✓*	✓	Developer environment - Linux	* Maemo support
✗*	✓	Developer environment - Mac	* Maemo support
✓	✓	Debugger support	
✓	✗*	Qt APIs for mobile development	* available as add-on
✓	✗	Build tool chain - Symbian/S60	
✓	✗	Build tool chain - Maemo	
✓	✗	Qt Simulator	

Installation

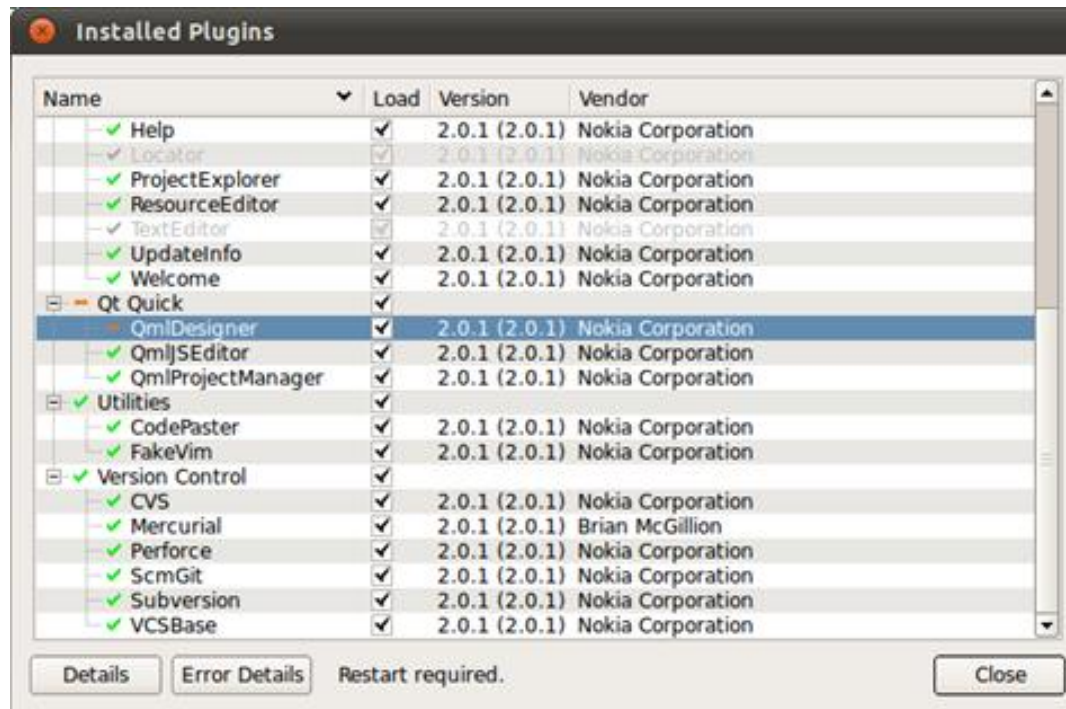


- Install Qt 4.7 from Ubuntu repositories
 - *sudo apt-get install build-essential libqt4-dev qt4-qmlviewer*
- Download and install the forum Nokia version of Nokia Qt SDK
 - Repository version doesn't have QML designer
- Run qtcreeator
 - `~/NokiaQtSDK/QtCreator/bin/qtcreeator`

Installation



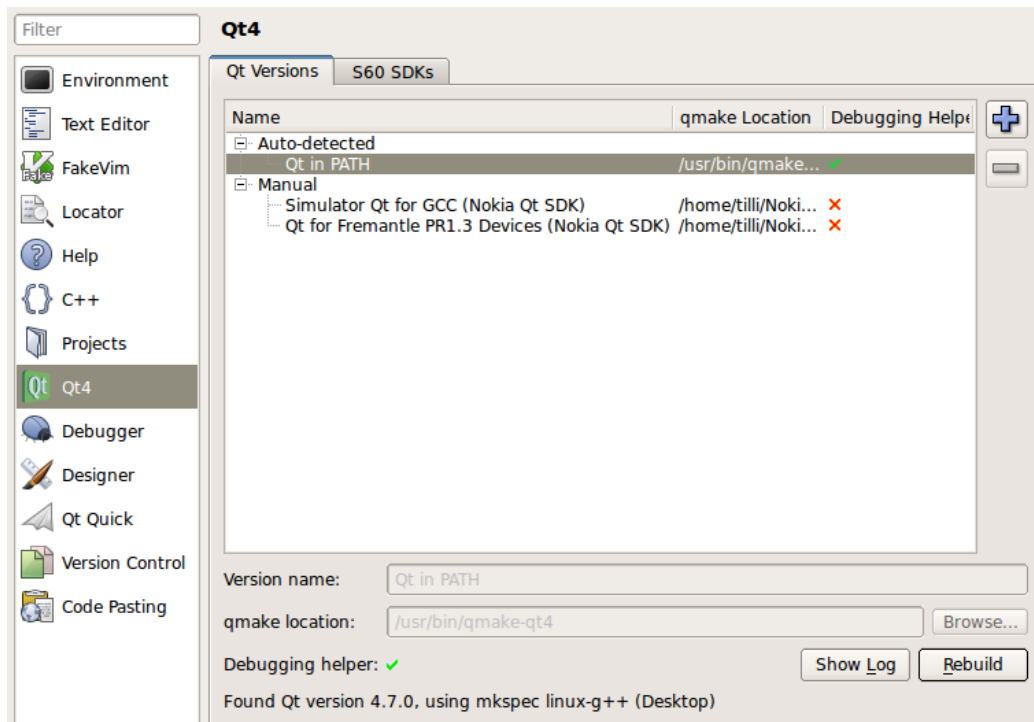
- Select *Help / About plugins* from menu
 - Enable QmlDesigner and re-start qtcreator



Installation



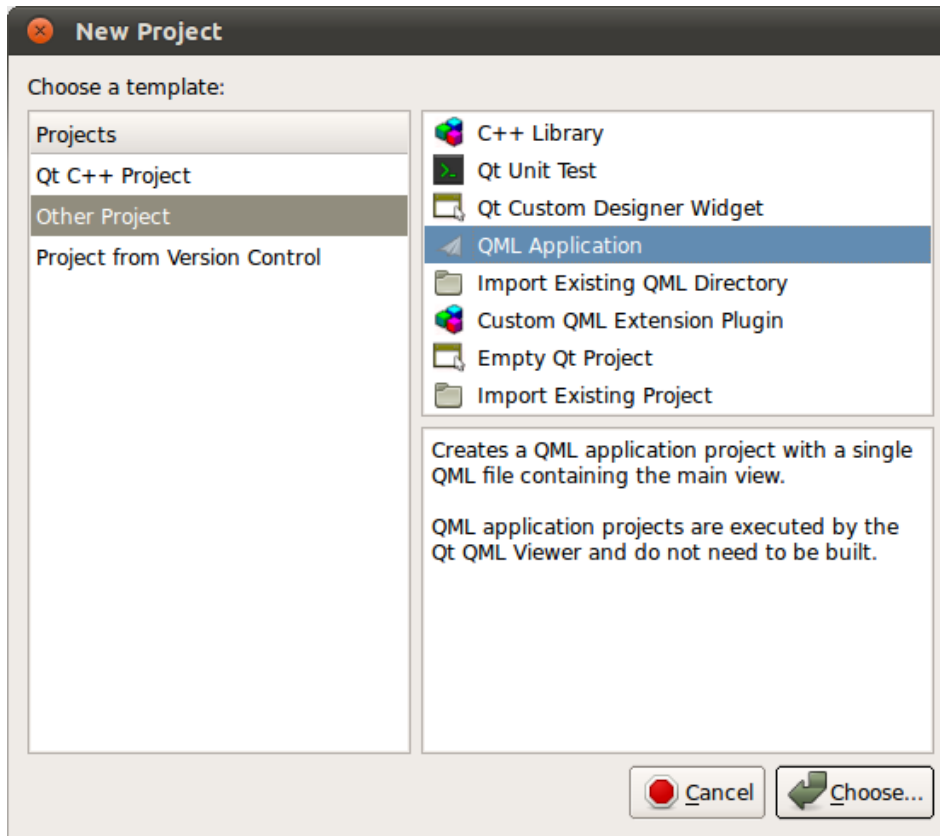
- Check *Tools / Options* that Qt libraries exist
 - Rebuild debug helper for C++ development



Quick start



- Select *File / New File or Project*



Quick start

< symbio >

New QML Project

Location
Summary

Introduction and project location

This wizard generates a QML application project.

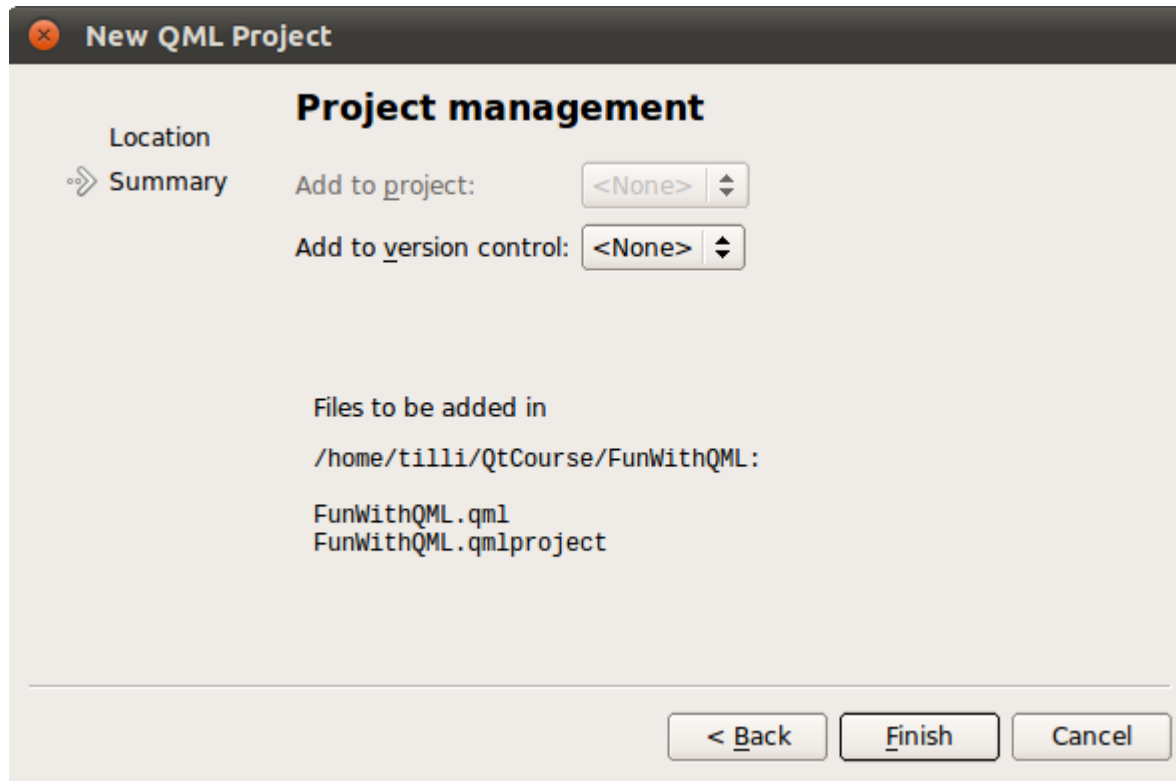
Name:

Create in:

☒ Use as default project location

Quick start

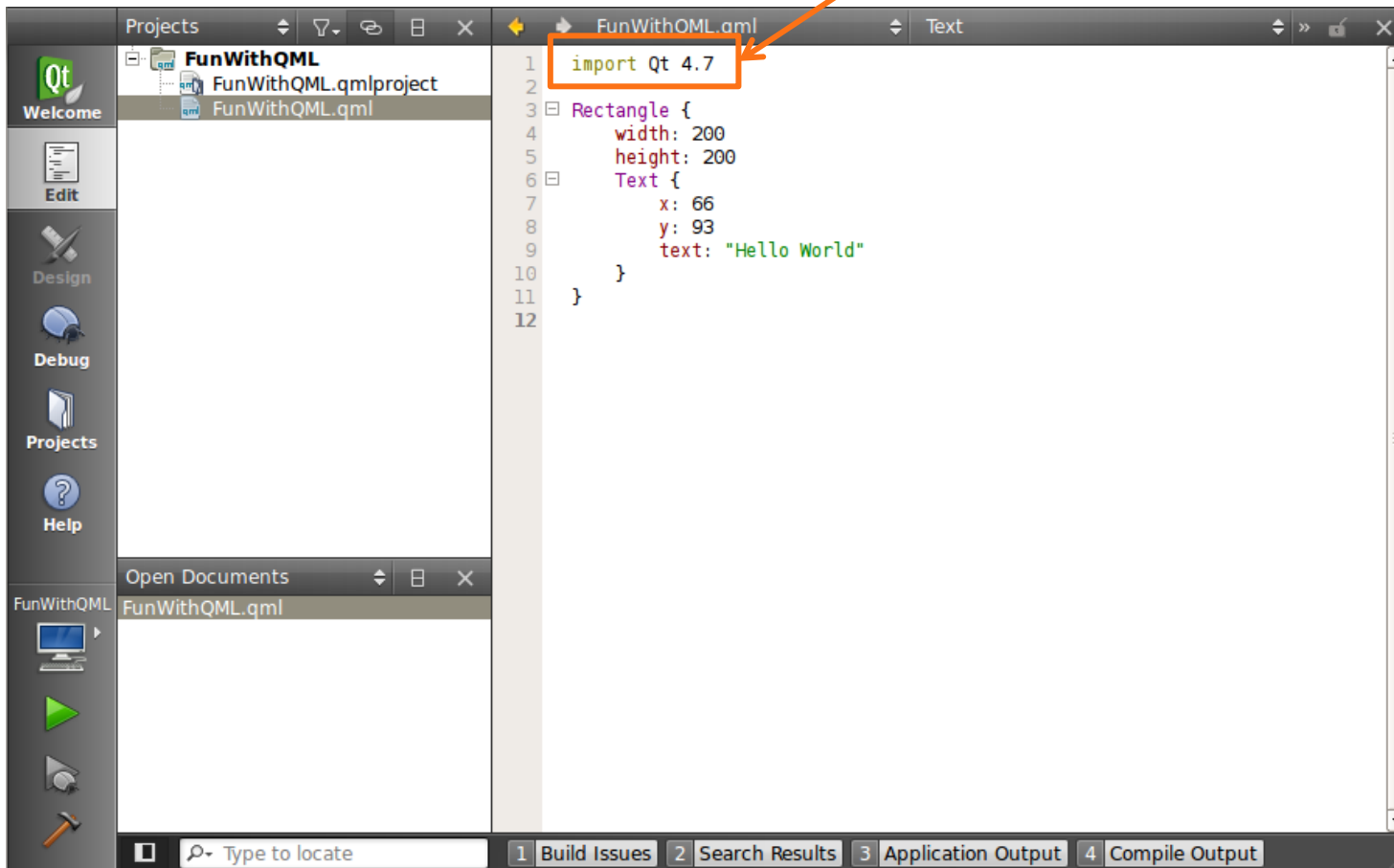
< symbio >



Quick start

< symbio >

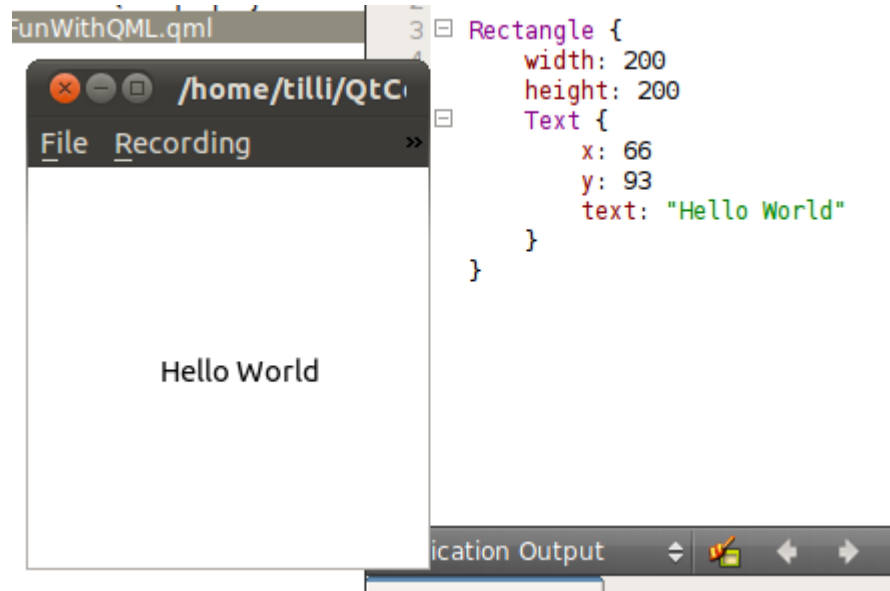
Will be changed to
import QtQuick 1.0



Quick start

< symbio >

- Run the program with Ctrl+R



Exercise



- Try it out, create and run a QML application project
 - Add three other text entries
 - Add an image to the middle



```
Image {  
    width: 50; height: 50  
    source: "http://qt.nokia.com/products/qt-addons/images/template/infocircle3d2.png/image_tile"  
}
```

Overview

QT QUICK

What is Qt Quick

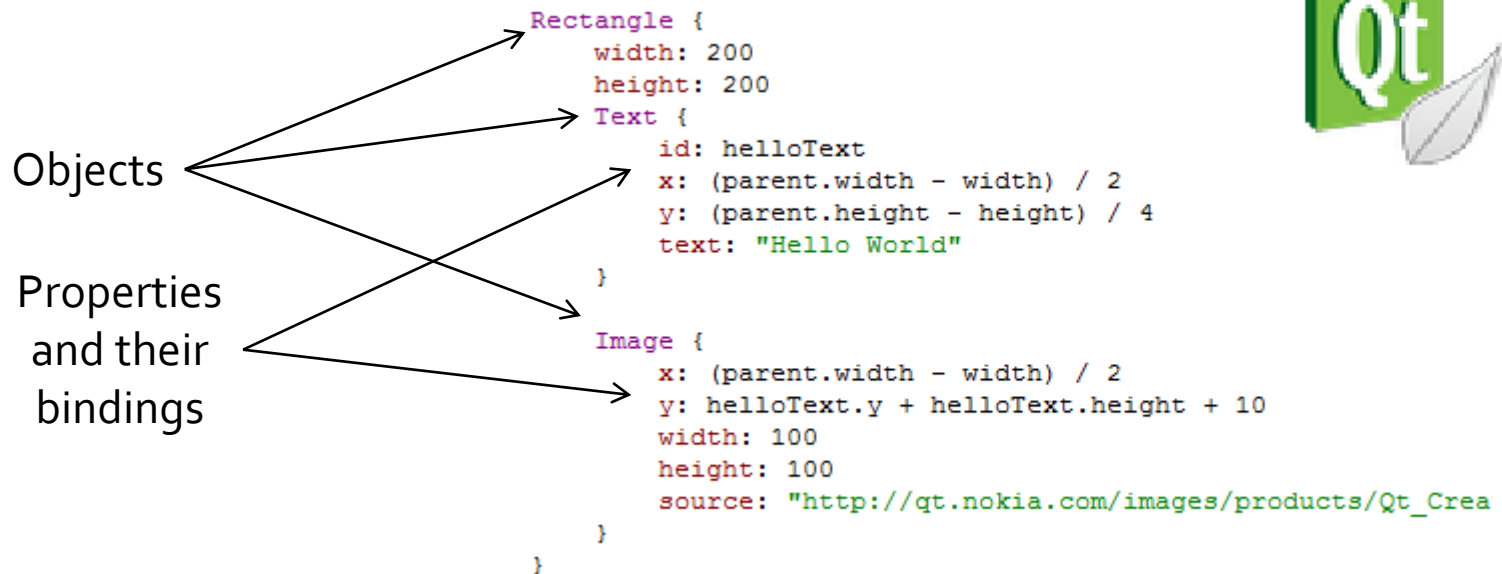


- QML – a language for UI design and development
- Qt declarative – Module for integrating QML and Qt C++ libraries
- Qt Creator tools – Complete development environment
 - QML design and code
 - C++ integration
 - Packaging and deployment

QML overview



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



QML overview



- Contrast with an *imperative language*

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

Property bindings are
statements that get evaluated
whenever property changes

```
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);
```

Statements are
evaluated once

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



- 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



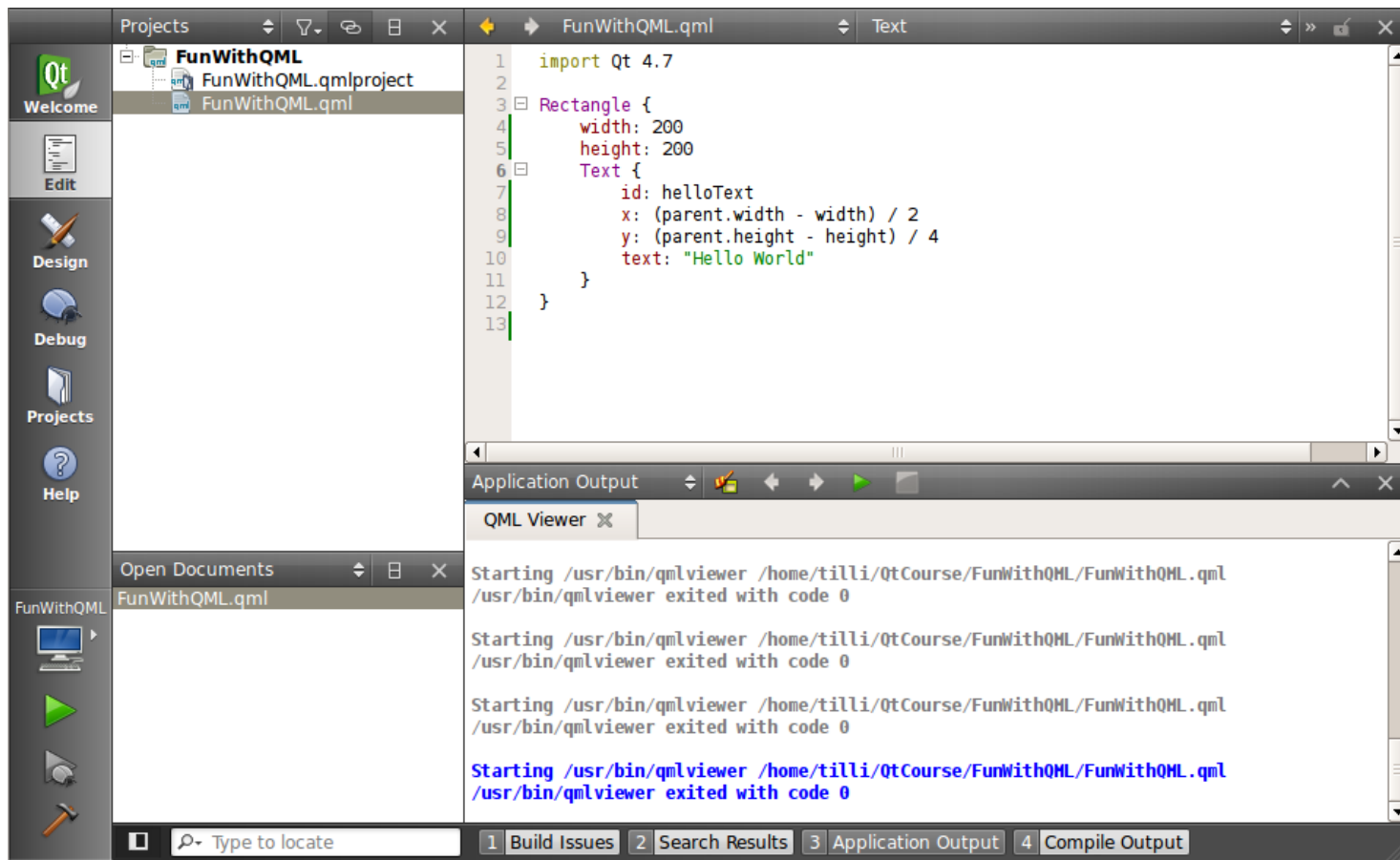
Qt Creator intro

< symbio >

- This is interactive part...
 - QML editor
 - QML designer
 - Project management
 - Adding / removing / renaming files
 - Session management

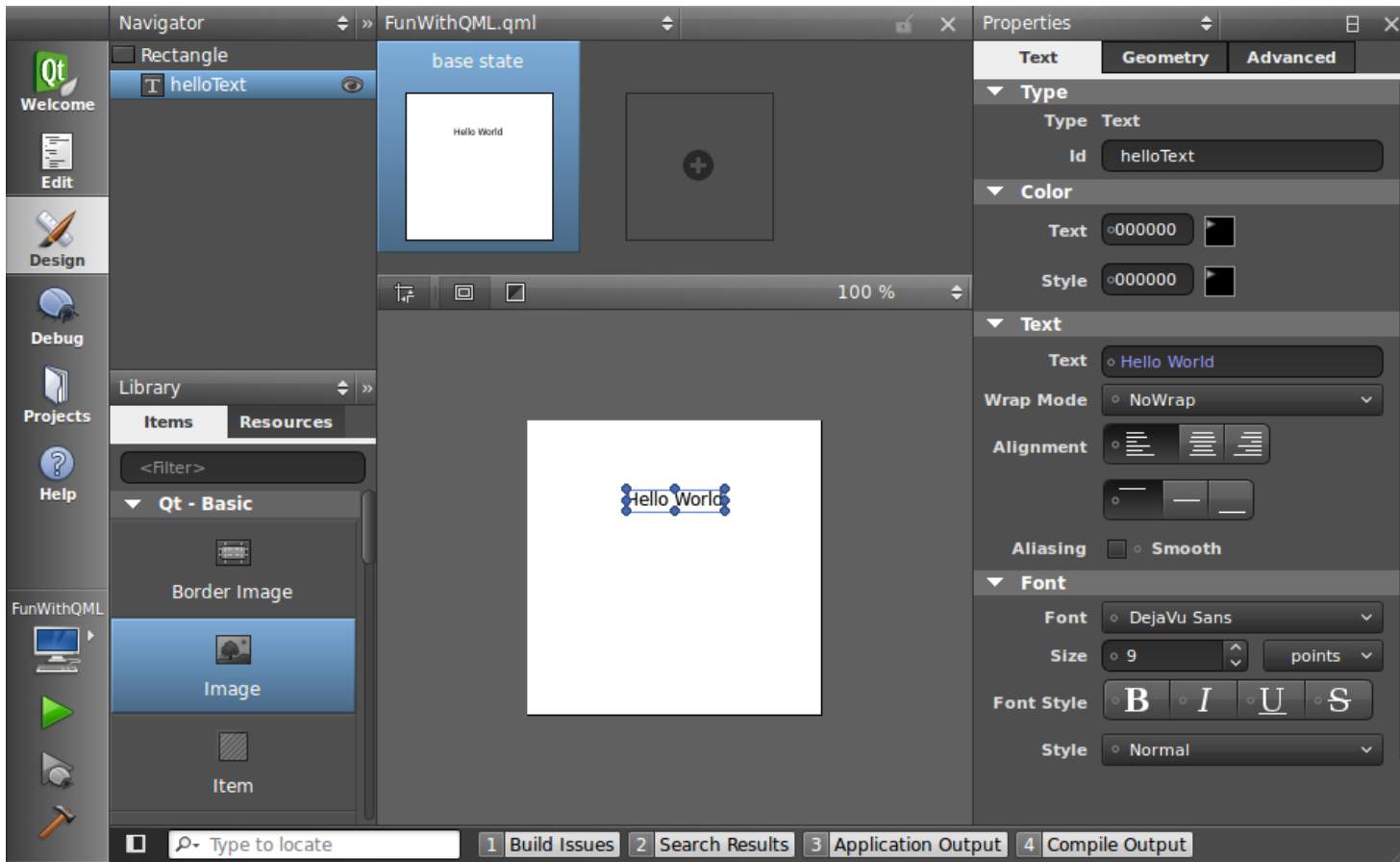
QML editor

< symbio >



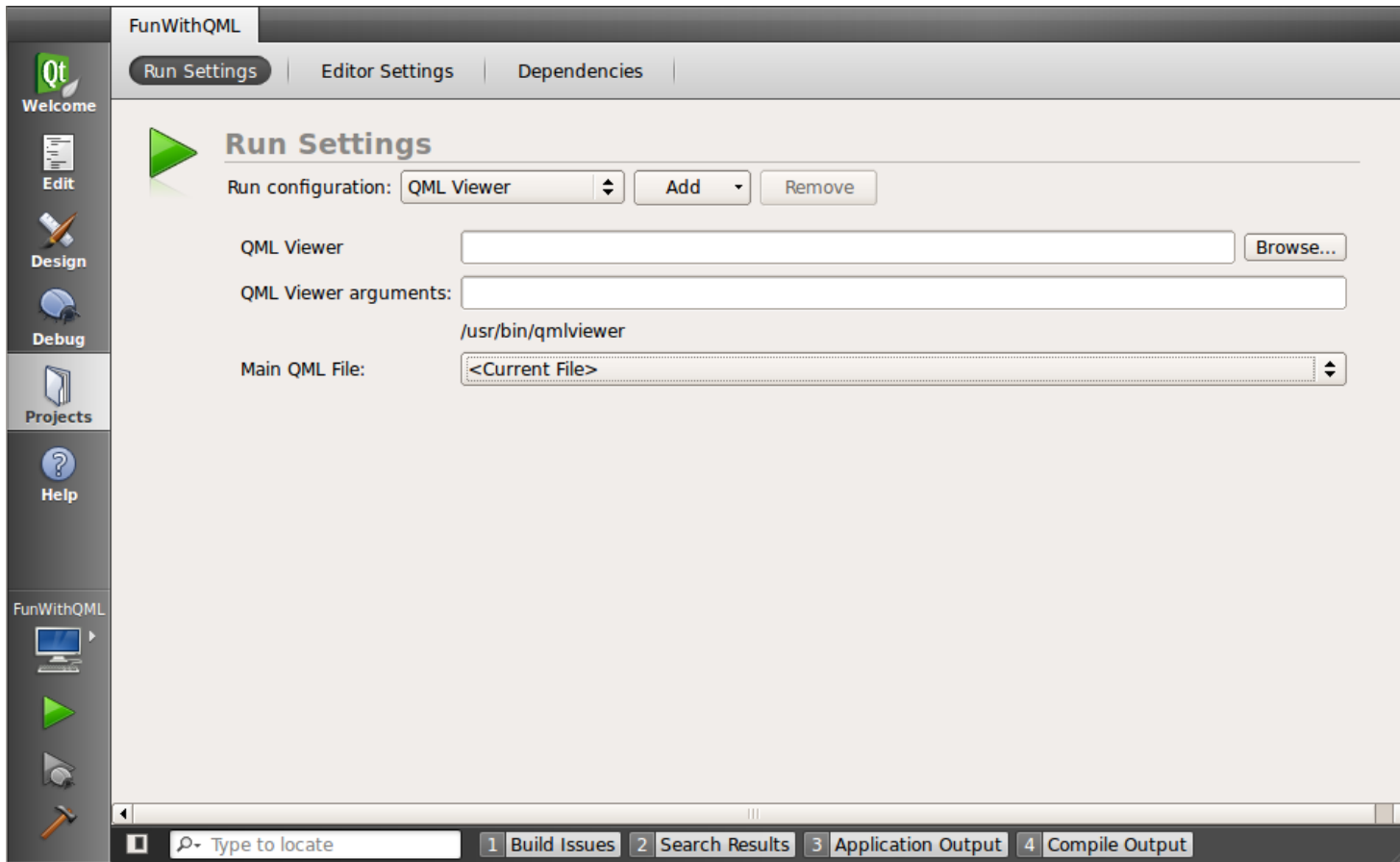
QML designer

< symbio >



QML project properties

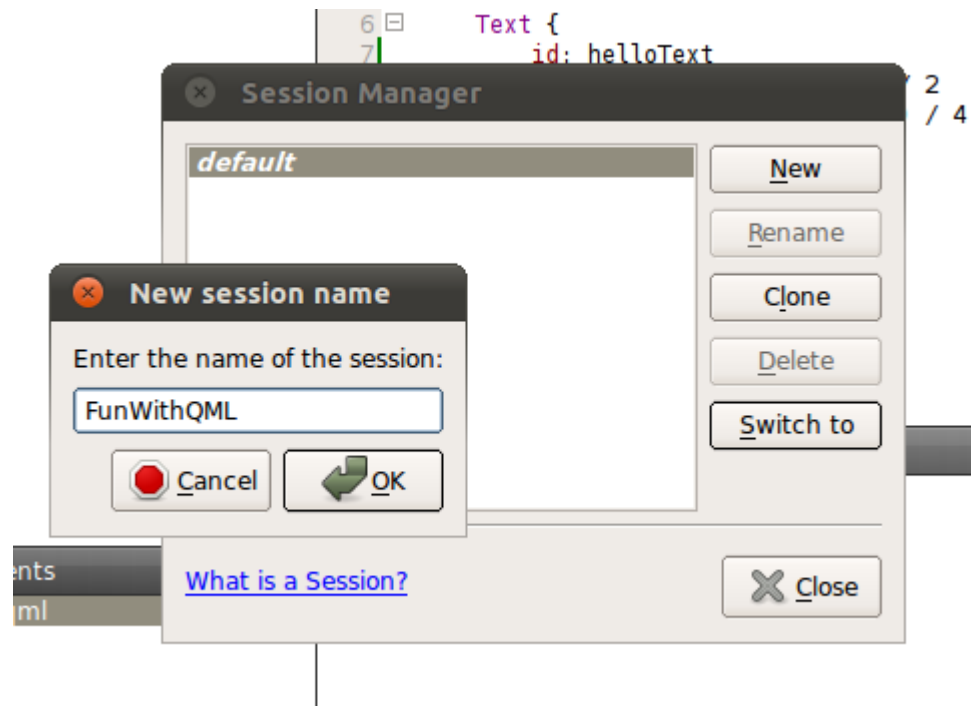
< symbio >



Session management



- File -> Sessions -> Session Manager



Designer exercise

< symbio >

- Create a new project
 - Make a similar UI as was done in previous exercise



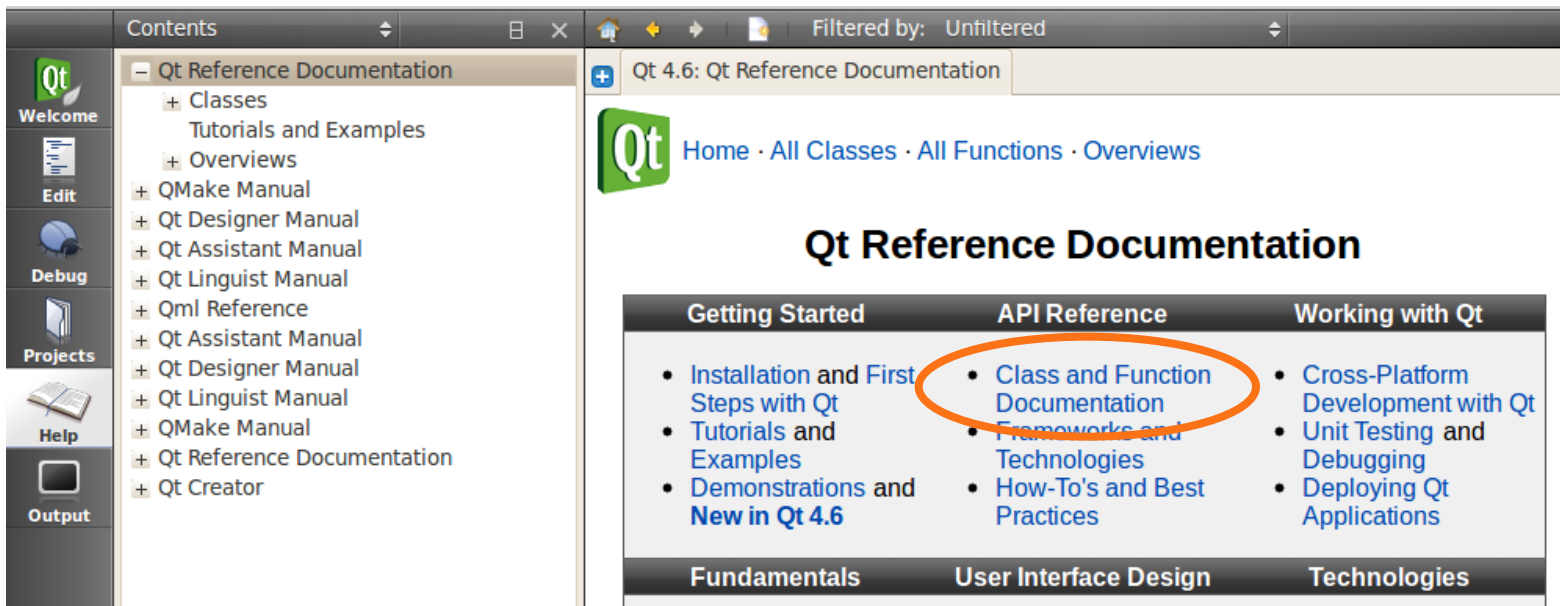
Overview of what's in there

QT MODULES



Qt modules walkthrough < symbio >

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



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



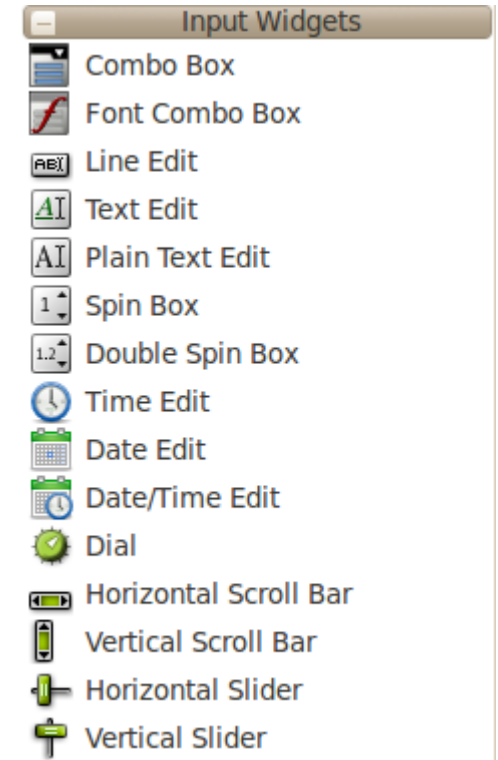
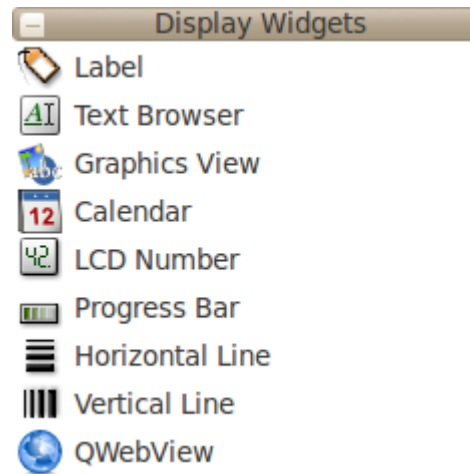
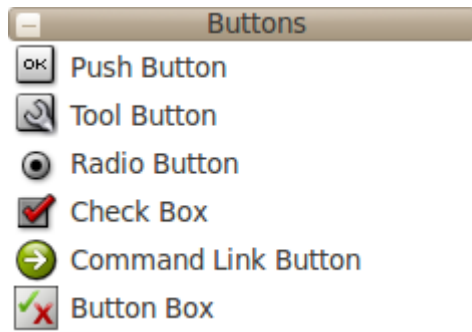
- Frameworks not discussed in this course
 - Multithreading (QFuture & related)
 - I/O devices (QIODevice, QFile & related)
 - State machines (QStateMachine & related)



GUI module

< symbio >

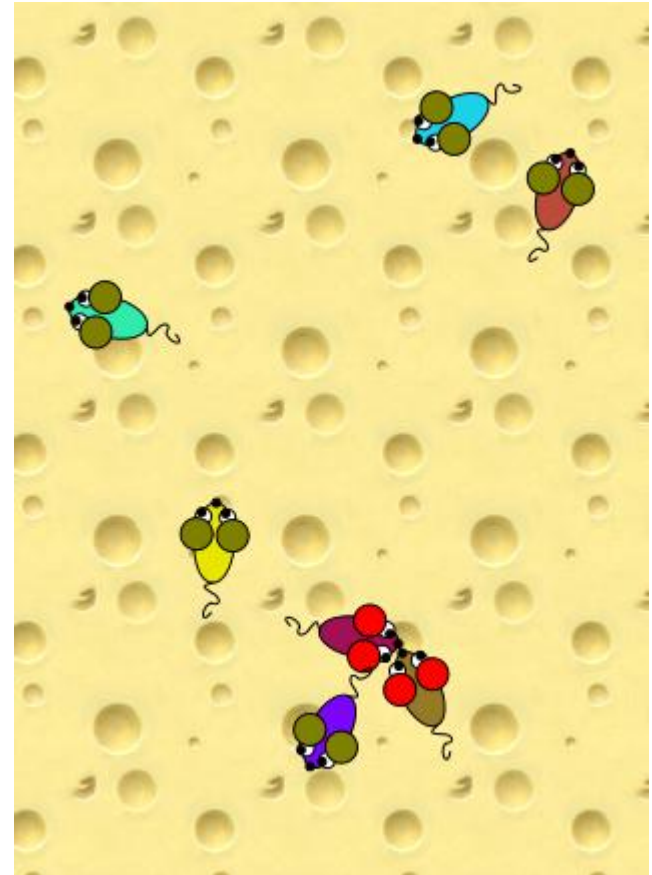
- “Traditional” widgets



GUI module

< symbio >

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



Network module

<symbio>

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



Multimedia modules

< symbio >

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

Scripting modules

<symbio>

- 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

- 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
 - *libqtm-...* packages with *apt-get install*
 - Works in Qt Simulator on PC
 - QML integration in progress

Basic concepts

QML PROGRAMMING

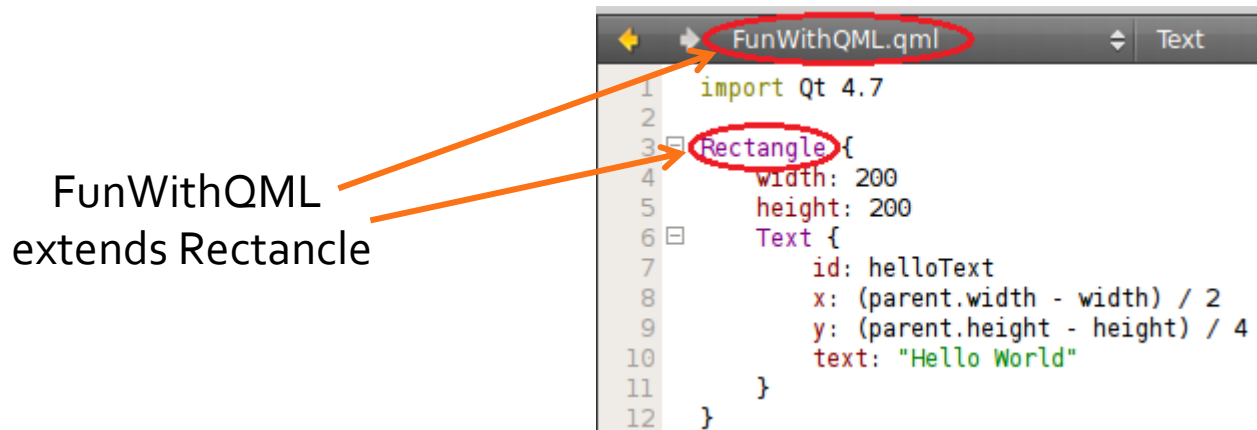


- Based on ECMA-262 specification
 - Operating environment differs from the usual web browser
 - DOM vs. QtDeclarative
 - Supports v5 features (notably JSON)
- Declarative concepts added on top
 - Quite a lot can be done without any "scriptiness"

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



Components



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

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

```
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()  
    }  
}
```

id property is used when referencing instances

Components

< symbio >

- 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

Simple values

Bindings

```
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>

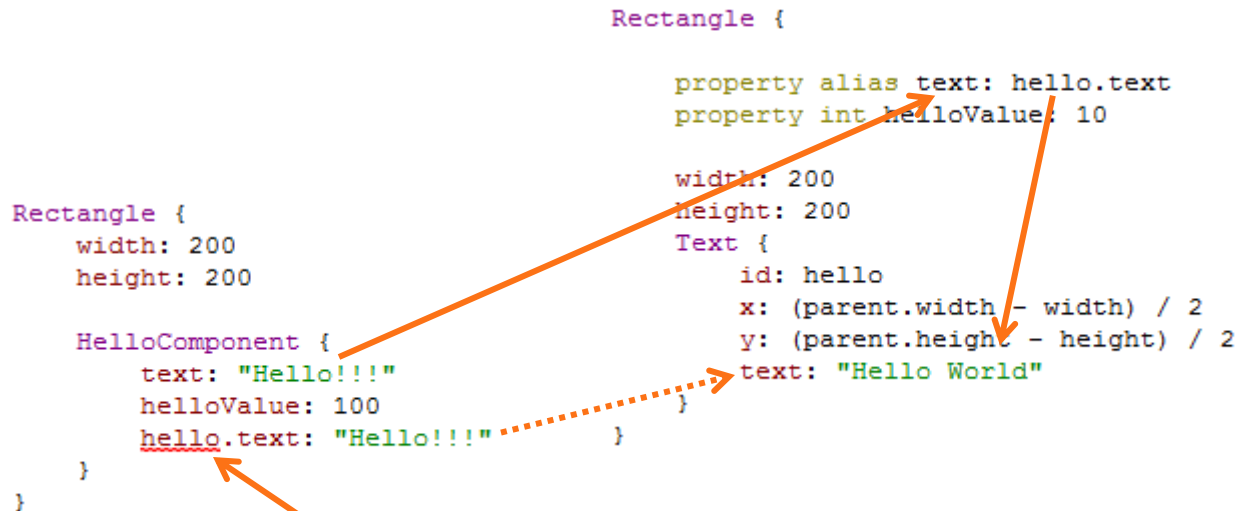
Alias properties



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

The diagram illustrates the relationship between two Rectangle components. The left component contains a HelloComponent with properties text, helloValue, and hello.text. The right component has properties for aliasing (property alias text: hello.text, property int helloValue: 100) and a Text component with id: hello and text: "Hello World". An orange arrow points from the text property of the HelloComponent to the property alias text in the second Rectangle. Another orange arrow points from the property alias text to the id: hello property of the Text component. A dotted orange arrow points from the hello.text property of the HelloComponent to the text property of the Text component.

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

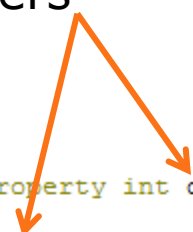
Mouse click
signal handler

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

Signals



- Property changes may be bound to signals
 - *on<Property>Changed* syntax
 - Note the capital case, similarly as with other signal handlers



```
property int detachCount: 0
onDetachCountChanged: {
    console.log("Rectangles detached: " + detachCount)
}
```

Signals



- Signals can be defined with *signal* keyword

Custom signal

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

Calling the signal

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

Custom signal handler

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

Functions

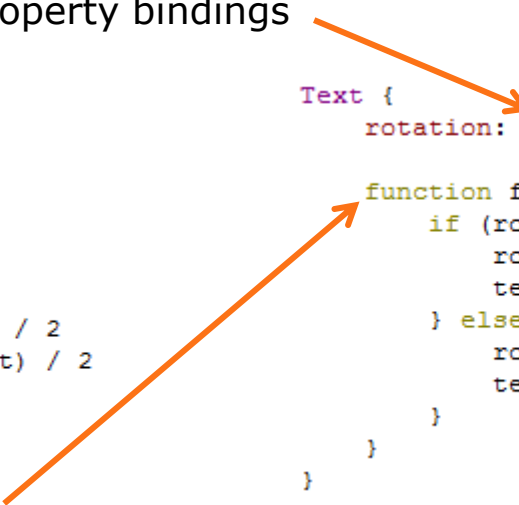


- A component may export functions that can be called from other components
 - Note: Not *declarative* way of doing things
 - Destroys property bindings

```
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()  
  }  
}
```

Diagram illustrating function calls in a declarative context:

- The `FlipText` component (inside `Rectangle`) calls the `flip()` function.
- The `flip()` function is defined within the `Text` component.
- The `Text` component has a `rotation` property that is bound to `parent.rotation`.

Two orange arrows originate from the text "Destroys property bindings" in the list above. One arrow points from the `onClicked: flipText.flip()` line in the `MouseArea` component to the `flip()` function definition in the `Text` component. The other arrow points from the `rotation: parent.rotation` line in the `Text` component to the `parent.rotation` property access in the `flip()` function.

Visual GUI items

QML GUI BASICS

- *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

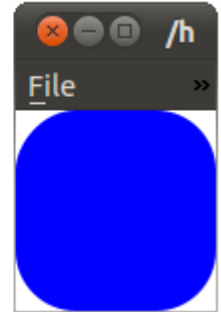
< symbio >

- *Rectangle* and *Image*

- Basic building blocks
- *Image* can be loaded from web

```
import Qt 4.7

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



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



- *Text*, *TextInput* and *TextEdit*

- For non-editable, single-line editable and multiline editable text areas

```
Rectangle {
    width: 100
    height: 100

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



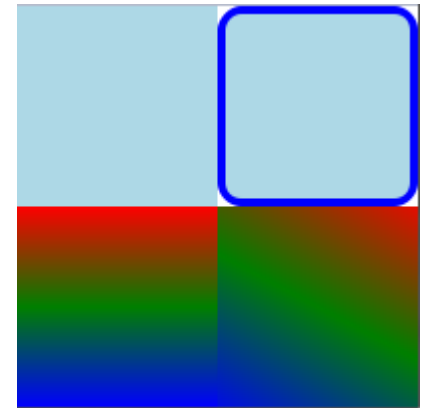
- And that's about it 😊

- Qt components project is in progress

Rectangle

< symbio >

- *Rectangle* is the basic visual element
 - *color* and *gradient*
 - *border* and *radius*
- Color can be a name or hex representation
- Gradient is always vertical
 - Can be rotated
- Example in *Rectangles* directory



Image

<sy**mbio**>

- *Image* element for standard image formats
 - *jpg, png, svg...*
 - File or URL *source*
 - Loads web images asynchronously
 - Image has *sourceSize*
 - Scaled to *width* and *height* by default
 - Inherited by *AnimatedImage*
- Example in *Images* directory

- *Text* element supports plain and rich text
 - HTML markup support
 - Fonts and styles from platform
- Text alignment
 - *verticalAlignment, horizontalAlignment*
- Multiline text wrapping supported

Text.AlignTop

Text.AlignVCenter

Text.AlignBottom

Text.AlignLeft

Text.AlignHCenter

Text.AlignRight

TextInput



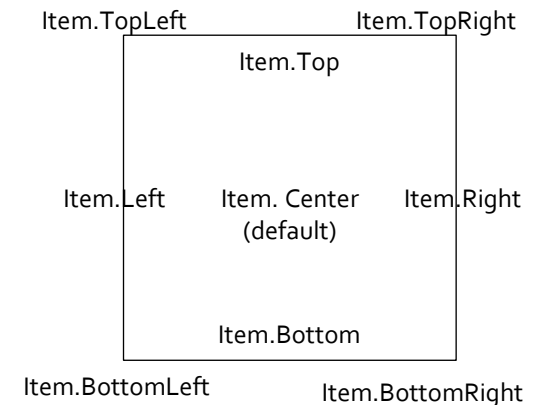
- *TextInput* is used for single-line input
 - Font and styles as in *Text*
 - Text selection supported
 - Integrates with system clipboard
 - Echo modes for password input
 - Text input validators
 - *IntValidator*, *DoubleValidator*, *RegExpValidator*
- Example in *TextInput* directory

- *TextEdit* for multi-line editing
 - Supports both plain and rich text
 - Mostly similar as *Text* and *TextInput*
 - Note *readOnly* and *selectByMouse* flags
 - *Text* that can be copied to system clipboard

Item transformations



- Each *Item* has two basic transformations
 - *rotation*
 - Around z-axis in degrees
 - *scale*
 - smaller < 1.0 < larger
 - Both relative to *transformOrigin*
 - "Stick through the screen"
- Additionally, item has *transform* list



Item transformations



- Transform objects allow more options
 - *Rotation in 3-D*
 - Around arbitrary axis (x, y, z)
 - *Scale*
 - Separate scale factors for x and y axis
 - *Translate*
 - Moves objects without affecting their x and y position
- Combination of any above
 - With arbitrary origin points

Putting the blocks together

ITEM LAYOUTS



Item layouts

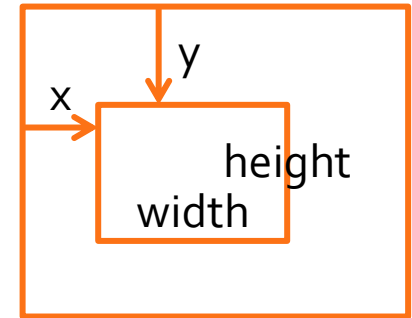
- Relative coordinates
- *Anchors* between items
- *Positioner* objects
 - *Row, Column, Flow, Grid*



Item coordinates

<sybio>

- 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  
  }  
}
```

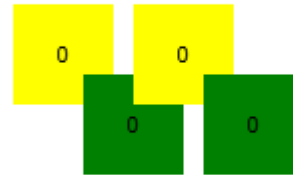


Item coordinates

< symbio >

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

State: All z-values zero

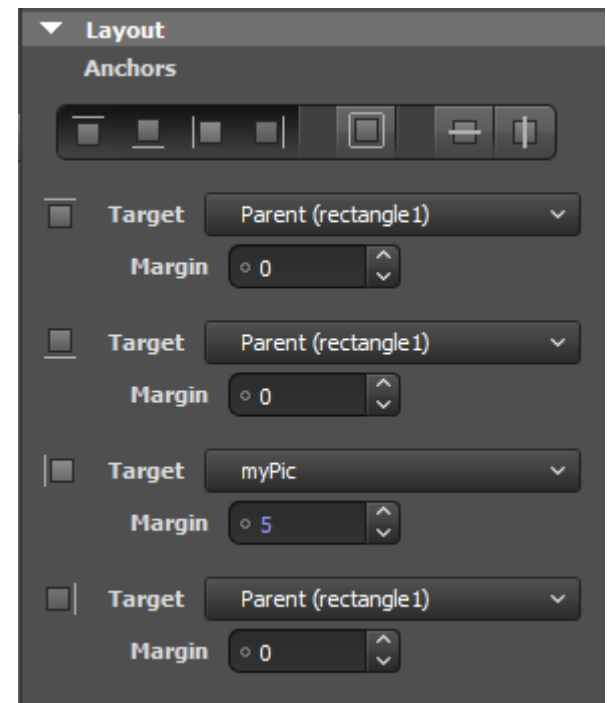


Item anchors



- Each 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

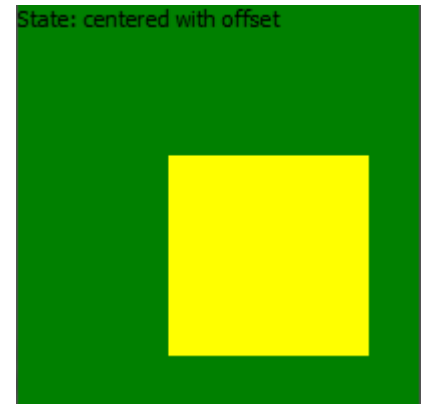
```
Rectangle {  
    id: rectangle2  
    color: "blue"  
    anchors.left: myPic.right  
    anchors.right: parent.right  
    anchors.bottom: parent.bottom  
    anchors.top: parent.top  
    anchors.leftMargin: 5  
}
```



Item anchors

< symbio >

- 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

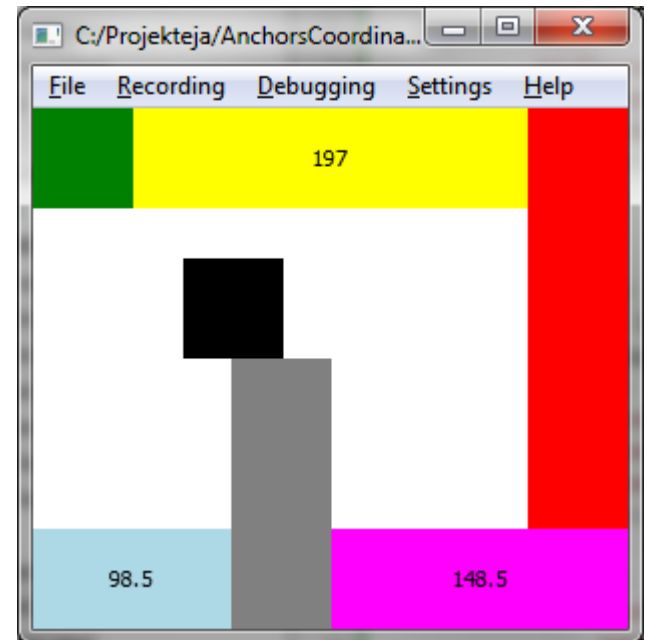


Anchors and coordinates

< symbio >

- Anchoring rules
 - Can only anchor to *parent* or *siblings*
 - Anchors will always overwrite *x* and *y*
 - *width* or *height* needed with single anchor
 - *width* or *height* overwritten when both sides anchored

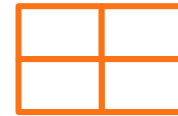
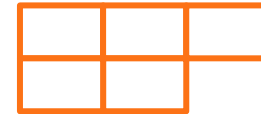
- Example in *AnchorsCoordinates*



Positioners

< symbio >

- Four positioner types:
 - *Row* lays out child items horizontally
 - *Column* lays them vertically
 - *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

< symbio >

- Create a QML application
 - Build following layout
- Add some interaction
 - When *Submit* is pressed, status bar text changes to whatever has been typed into text input
 - If a color is clicked, status bar text changes to represent that color

- "red", "green" etc.

Text input and button



< symbio >



SERIOUS ABOUT SOFTWARE