Qt Quick – Overview and basic GUI

Timo Strömmer, Feb 4, 2011

Contents

- Qt Quick overview
  - SDK installation notes
  - What is Qt Quick
  - Qt modules overview
- Programming with QML
  - Basic concepts
  - Structuring QML programs
  - Basic GUI elements and layouts
  - Mouse and keyboard interaction

Disclaimer

- Based on a 4-day course at Haaga-Helia
  - http://terokarvinen.com/courses/mobile-linux-development-with-qt
- Original slides and examples available at
  - http://terokarvinen.com/oldsite/otherauthors/qt/2011/?C=M;O=D

Qt SDK installation

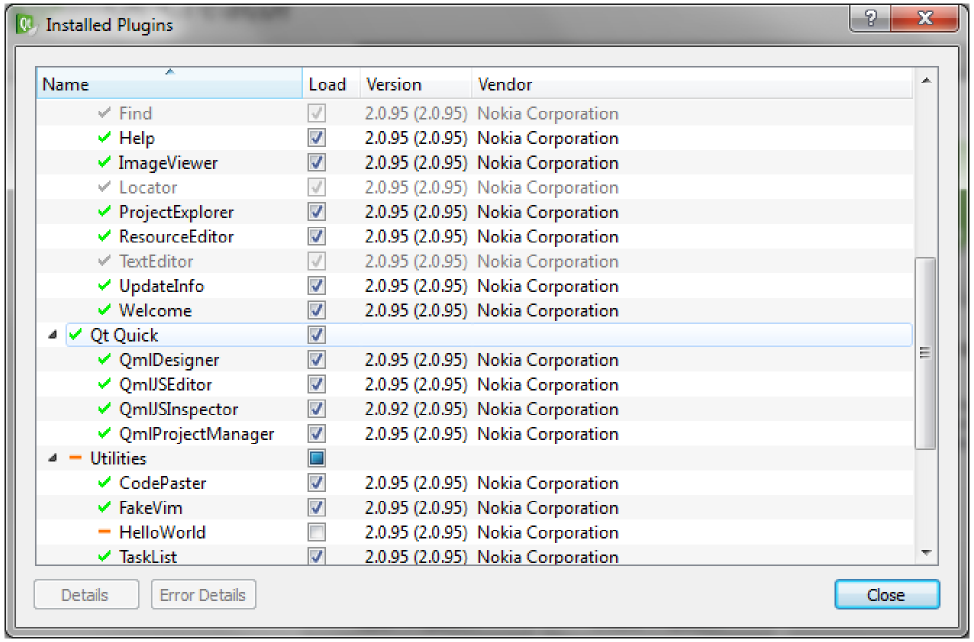
QT QUICK OVERVIEW

Qt SDK's

- Latest Qt SDK tech preview
  - http://www.forum.nokia.com/info/sw.nokia.com/id/da8df288-e615-443d-be5c-00c8a72435f8/Qt_SDK.html
- "Old" stuff:
  - http://qt.nokia.com/downloads/downloads
    - Latest Qt meant for desktop
  - http://www.forum.nokia.com/Develop/Qt/
    - Meant for mobile devices

Installation checks

- *Help / About plugins*
  - Tech preview should have *QmlDesigner* enabled

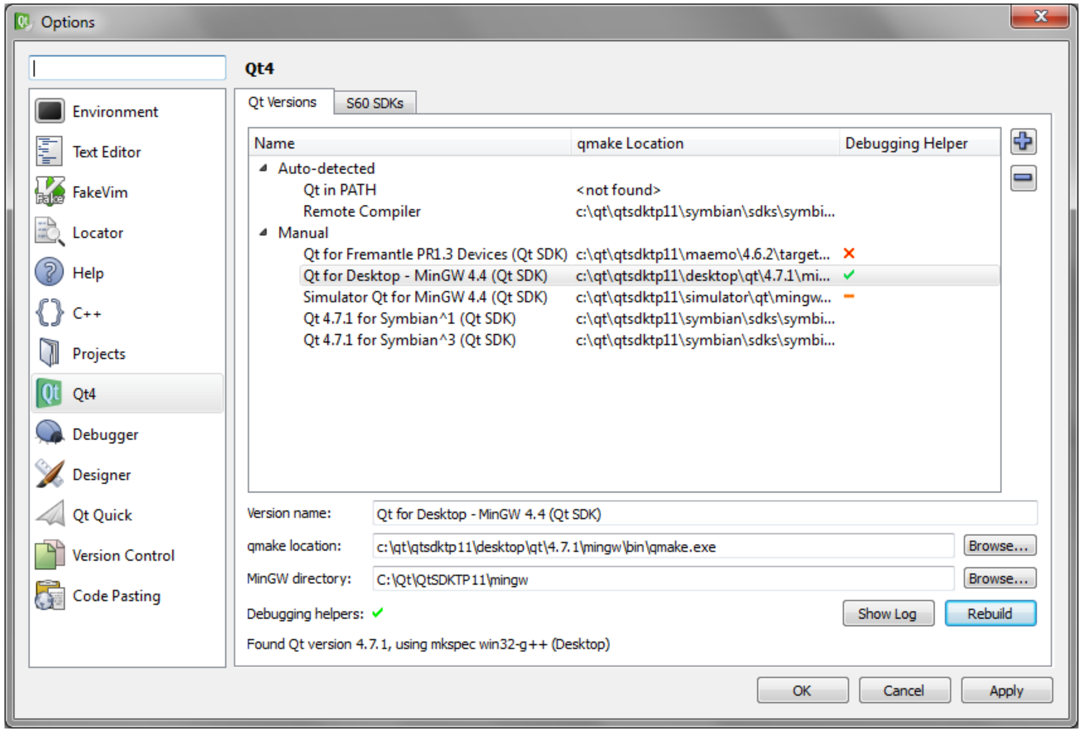


Installed Plugins

Name	Load	Version	Vendor
✓ Find	✓	2.0.95 (2.0.95)	Nokia Corporation
✓ Help	✓	2.0.95 (2.0.95)	Nokia Corporation
✓ ImageViewer	✓	2.0.95 (2.0.95)	Nokia Corporation
✓ Locator	✓	2.0.95 (2.0.95)	Nokia Corporation
✓ ProjectExplorer	✓	2.0.95 (2.0.95)	Nokia Corporation
✓ ResourceEditor	✓	2.0.95 (2.0.95)	Nokia Corporation
✓ TextEditor	✓	2.0.95 (2.0.95)	Nokia Corporation
✓ UpdateInfo	✓	2.0.95 (2.0.95)	Nokia Corporation
✓ Welcome	✓	2.0.95 (2.0.95)	Nokia Corporation
✓ Qt Quick	✓		
✓ QmlDesigner	✓	2.0.95 (2.0.95)	Nokia Corporation
✓ QmlJSEditor	✓	2.0.95 (2.0.95)	Nokia Corporation
✓ QmlJSInspector	✓	2.0.92 (2.0.95)	Nokia Corporation
✓ QmlProjectManager	✓	2.0.95 (2.0.95)	Nokia Corporation
– Utilities	■		
✓ CodePaster	✓	2.0.95 (2.0.95)	Nokia Corporation
✓ FakeVim	✓	2.0.95 (2.0.95)	Nokia Corporation
– HelloWorld	☐	2.0.95 (2.0.95)	Nokia Corporation
✓ TaskList	✓	2.0.95 (2.0.95)	Nokia Corporation

Installation checks

- *Tools / Options* and *Qt4*
  - Careful with *Qt in PATH* (4.6.x won't work)



Qt4

Name	qmake Location	Debugging Helper
Auto-detected		
Qt in PATH	<not found>	
Remote Compiler	c:\qt\qtsdktp11\symbian\sdks\symbi...	
Manual		
Qt for Fremantle PR1.3 Devices (Qt SDK)	c:\qt\qtsdktp11\maemo\4.6.2\target...	✗
Qt for Desktop - MinGW 4.4 (Qt SDK)	c:\qt\qtsdktp11\desktop\qt\4.7.1\mi...	✓
Simulator Qt for MinGW 4.4 (Qt SDK)	c:\qt\qtsdktp11\simulator\qt\mingw...	–
Qt 4.7.1 for Symbian^1 (Qt SDK)	c:\qt\qtsdktp11\symbian\sdks\symbi...	
Qt 4.7.1 for Symbian^3 (Qt SDK)	c:\qt\qtsdktp11\symbian\sdks\symbi...	

Version name: Qt for Desktop - MinGW 4.4 (Qt SDK)

qmake location: c:\qt\qtsdktp11\desktop\qt\4.7.1\mingw\bin\qmake.exe

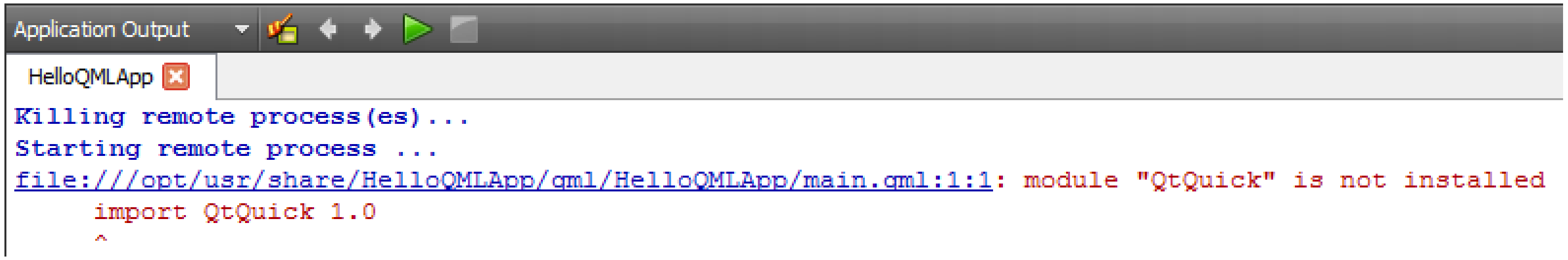
MinGW directory: C:\Qt\QtSDKTP11\mingw

Debugging helpers: ✓

Found Qt version 4.7.1, using mkspec win32-g++ (Desktop)

N900 environment setup

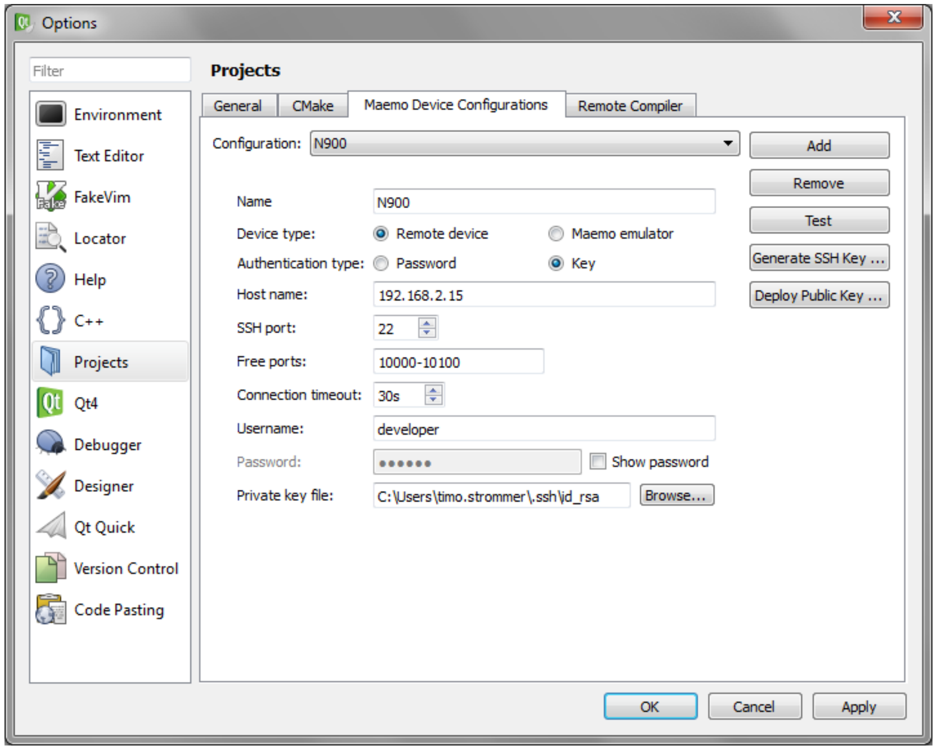
- N900 guide at:
  - http://wiki.forum.nokia.com/index.php/Set_up_Qt_for_Maemo_Environment
- N900 has older Qt version
  - Use "import Qt 4.7" in QML applications for now



```
Killing remote process(es)...
Starting remote process ...
file:///opt/usr/share/HelloQMLApp/qml/HelloQMLApp/main.qml:1:1: module "QtQuick" is not installed
     import QtQuick 1.0
     ^
```

N900 environment setup

- *Tools / Options* and *Projects / Maemo Device Configurations*



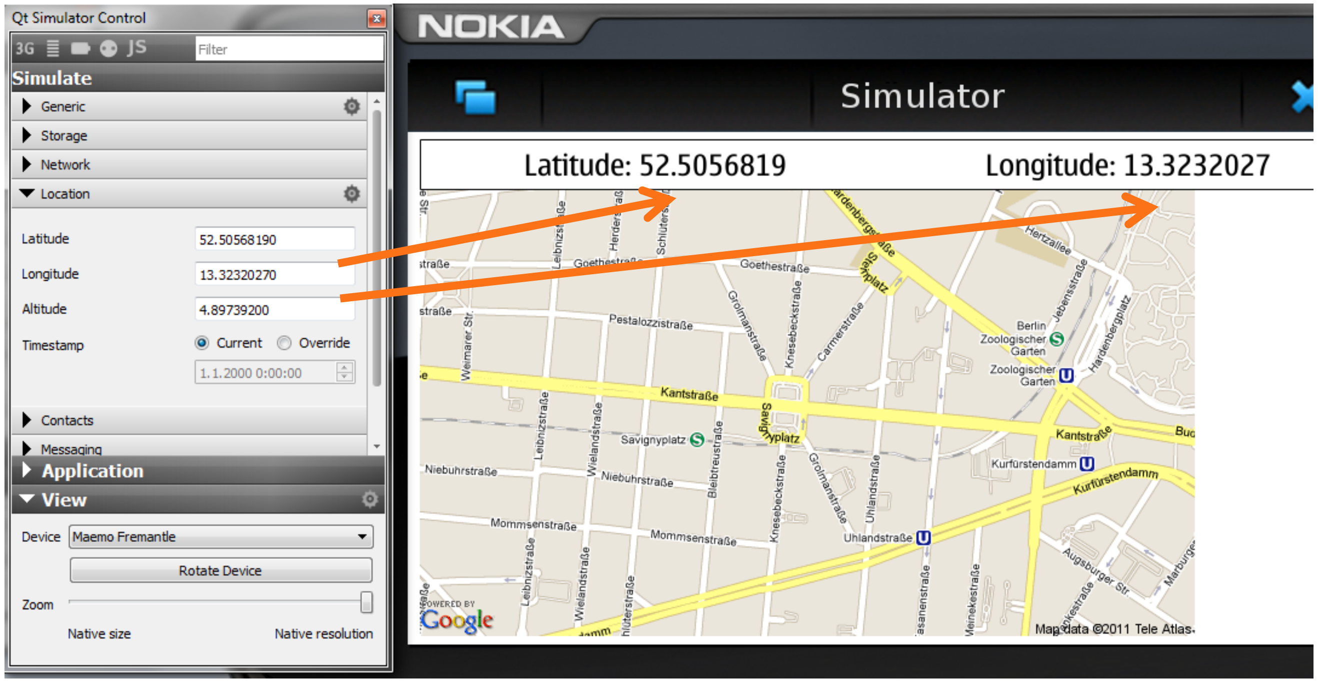
Projects

Configuration: N900

Name	N900	
Device type:	[✓ Remote device]	Maemo emulator
Authentication type:	Password	[✓ Key]
Host name:	192.168.2.15	
SSH port:	22	
Free ports:	10000-10100	
Connection timeout:	30s	
Username:	developer	
Password:	••••••	Show password
Private key file:	C:\Users\timo.strommer\.ssh\id_rsa	

Qt Simulator

- Simulator target can be used to test N900 or Symbian projects without real device



Latitude 52.50568190
Longitude 13.32320270
Altitude 4.89739200
Timestamp [✓ Current] Override
1.1.2000 0:00:00
Device Maemo Fremantle

Latitude: 52.5056819 Longitude: 13.3232027

What is Qt Quick

QT QUICK OVERVIEW

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
    - Design, code, package, deploy

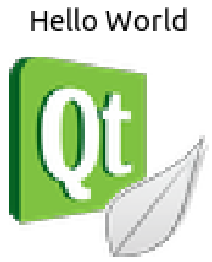
QML overview

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

Objects

Properties and their bindings

```
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*

```
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

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);
```

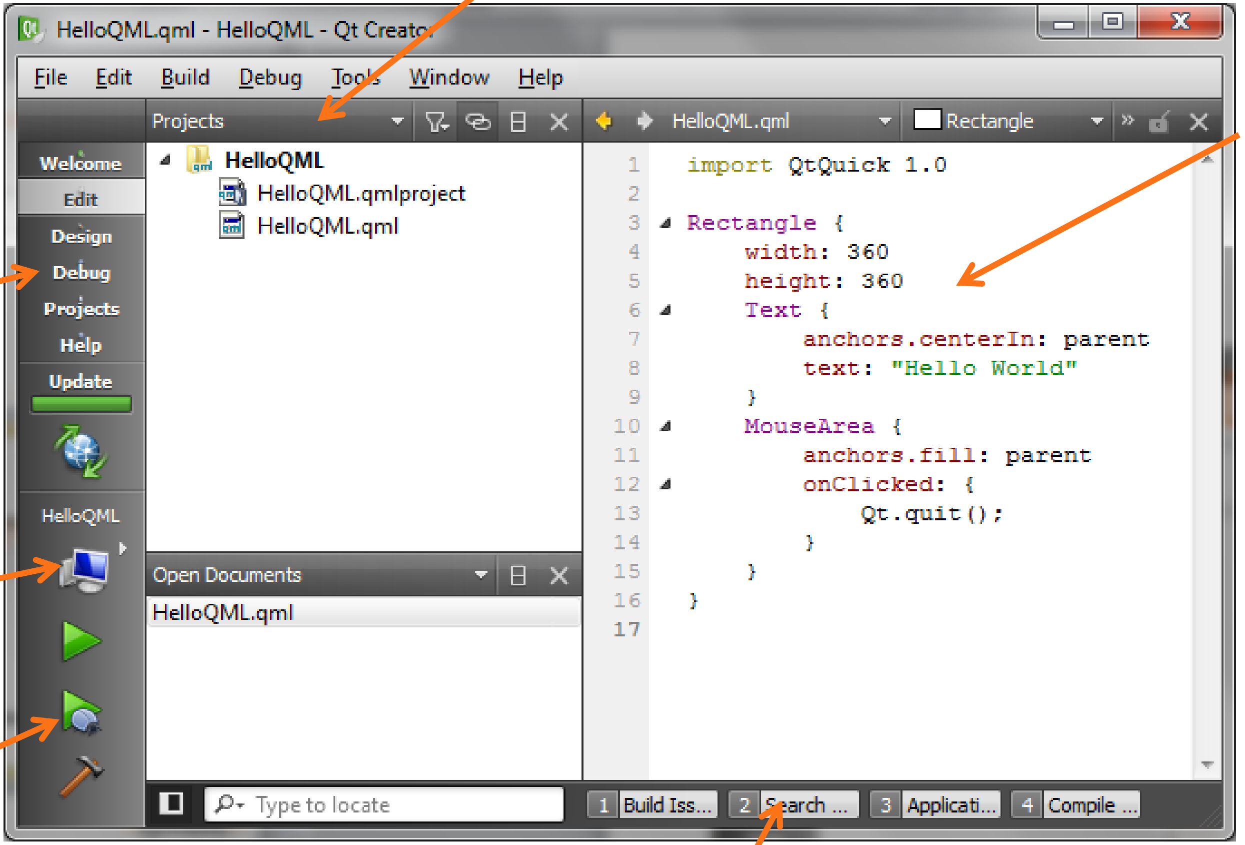
Qt Declarative

- 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

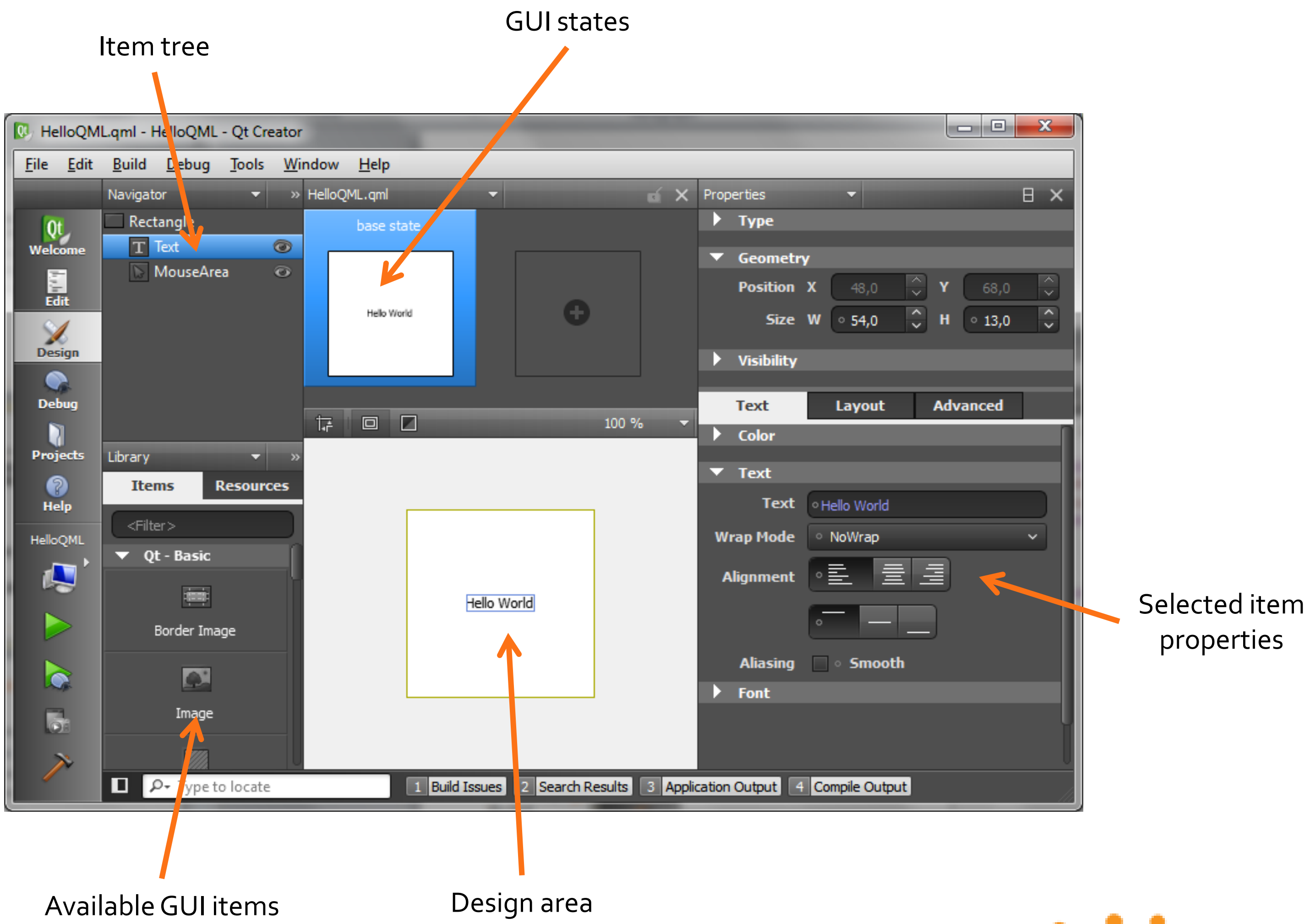
- Qt Creator integrates C++ and QML development into single IDE
    - Designers for visual editing
        - QML designer
        - Widget UI designer
    - QML and C++ code editors
    - Same code can be run at desktop or device

Qt Creator



```
import QtQuick 1.0

Rectangle {
    width: 360
    height: 360
    Text {
        anchors.centerIn: parent
        text: "Hello World"
    }
    MouseArea {
        anchors.fill: parent
        onClicked: {
            Qt.quit();
        }
    }
}
```

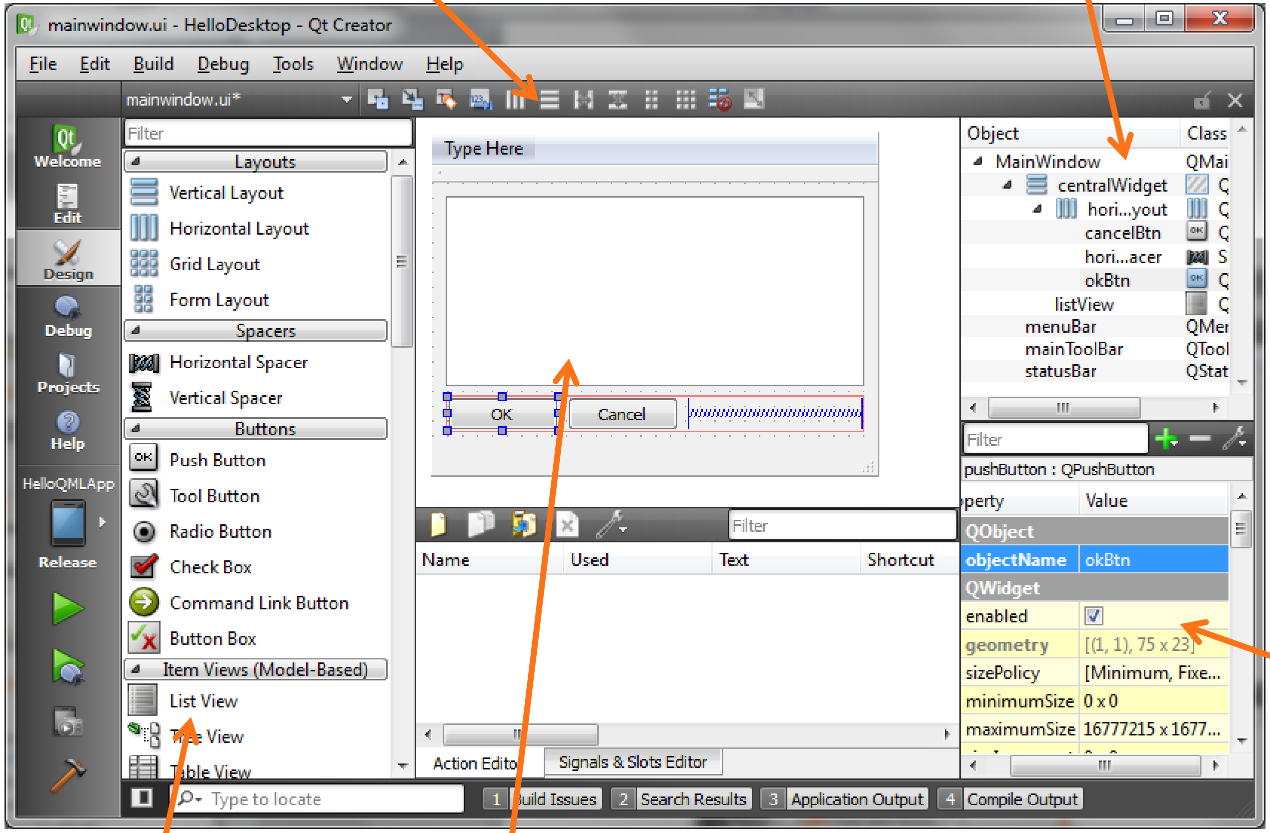
QML designer



Position	X	48,0	Y	68,0
Size	W	54,0	H	13,0

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

Widget UI designer



Object	Class
MainWindow	QMai
centralWidget	Q
hori...yout	Q
cancelBtn	Q
hori...acer	S
okBtn	Q
listView	Q
menuBar	QMer
mainToolBar	QTool
statusBar	QStat

pushButton : QPushButton

Property	Value
QObject	
objectName	okBtn
QWidget	
enabled	✓
geometry	[(1, 1), 75 x 23]
sizePolicy	[Minimum, Fixe...
minimumSize	0 x 0
maximumSize	16777215 x 1677...

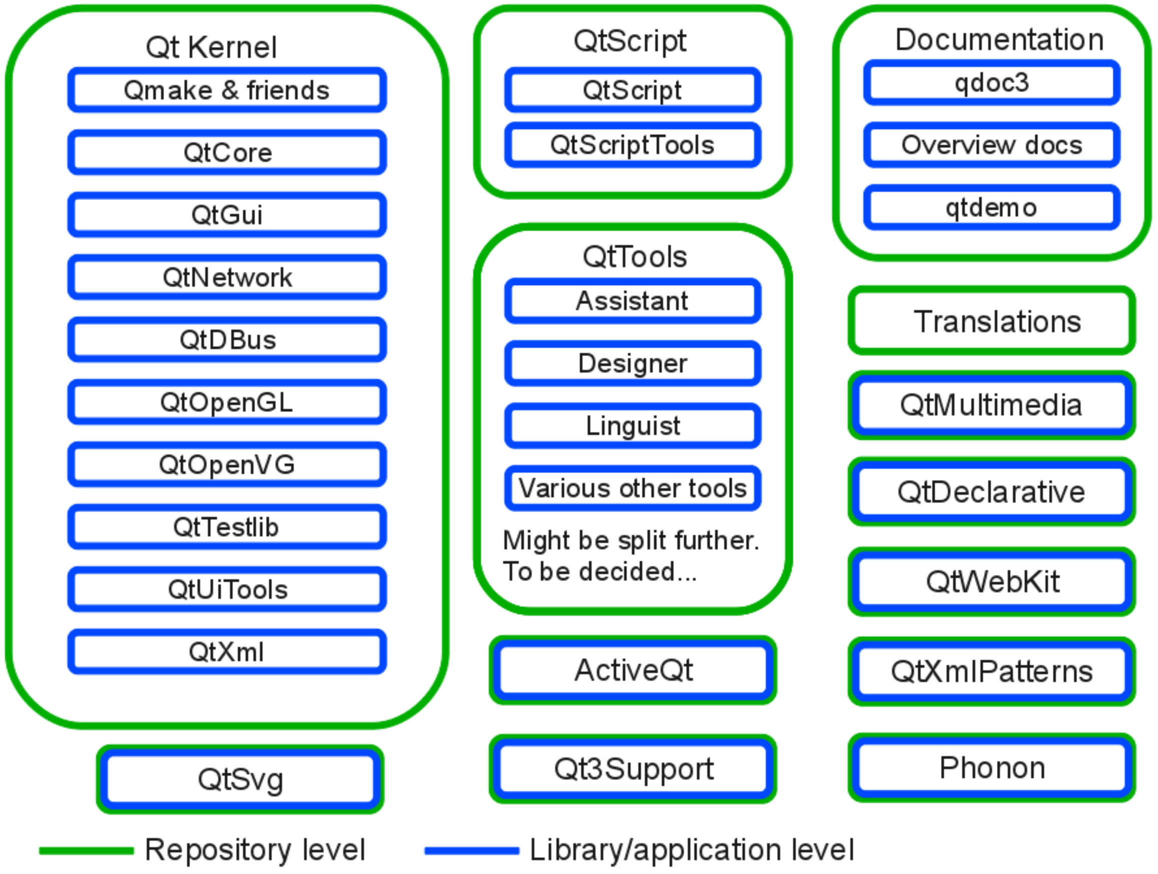
Qt Quick projects

- Qt Quick UI
  - Just QML, no deployment options
    - See also http://qml.haltu.fi/
- Qt Quick Application
  - QML packaged into C++ application
  - Deployment to device from QtCreator
- QML extension plug-in
  - C++ library loaded by QML runtime

Qt modules

QT QUICK OVERVIEW

Modularization project



Mobile development

- Qt Mobility API's
  - Device peripherals and frameworks
  - Latest release 1.1 (also tech preview 1.2):
    - 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

Mobility API's

	S60 5th Edition	Symbian	Maemo 5	Harmattan	Windows XP/Vista	Linux	Mac OS X
Service Framework (in-process)	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Messaging	🟩	🟩	🟩	🟨	🟩	🟩	▨
Bearer Management	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Publish and Subscribe	🟩	🟩	🟩	🟨	🟩	🟩	🟩
Contacts	🟩	🟩	🟩	🟨	▨	▨	▨
Location	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Multimedia	🟩	🟩	🟩	🟨	🟩	🟩	🟩
System Information	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Sensors	🟩	🟩	🟩	🟩	▨	▨	▨
Versit(vCard)	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Versit(Organizer)	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Camera	🟩	🟩	🟩	🟨	▨	▨	▨
Service Framework(OOP)	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Organizer	🟩	🟩	🟩	🟨	▨	▨	▨
Landmarks	🟩	🟩	🟩	🟨	🟩	🟩	🟩
Document Gallery	🟨 *)	🟩	🟩	🟨	▨	▨	▨
Maps/Navigation	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Feedback	🟨	🟨	🟨	🟨	▨	▨	▨

Basic concepts

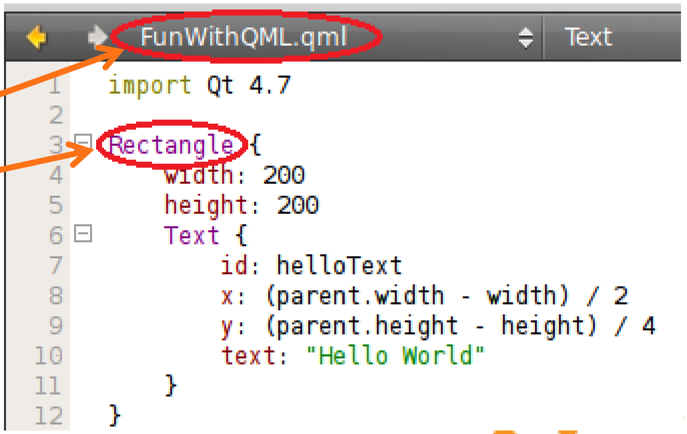
QML PROGRAMMING

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

```
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()
    }
}
```

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

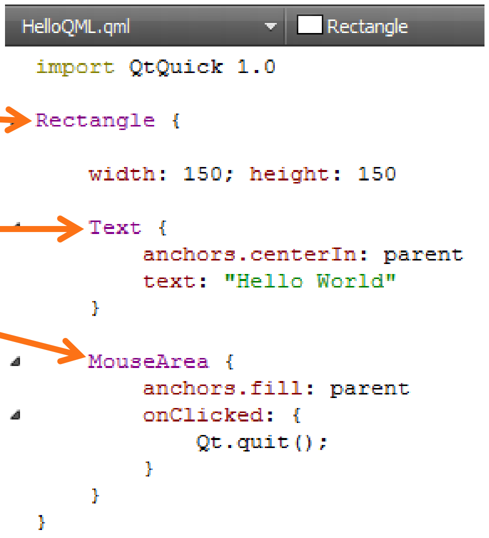
id property is used when referencing instances

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

Object tree

- QML program is run in QML engine
  - *QDeclarativeEngine* class at C++ side
- Engine has a single *root*
  - Any number of *children*
  - *QDeclarativeContext* at C++ side



```
import QtQuick 1.0
Rectangle {
    width: 150; height: 150
    Text {
        anchors.centerIn: parent
        text: "Hello World"
    }
    MouseArea {
        anchors.fill: parent
        onClicked: {
            Qt.quit();
        }
    }
}
```

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

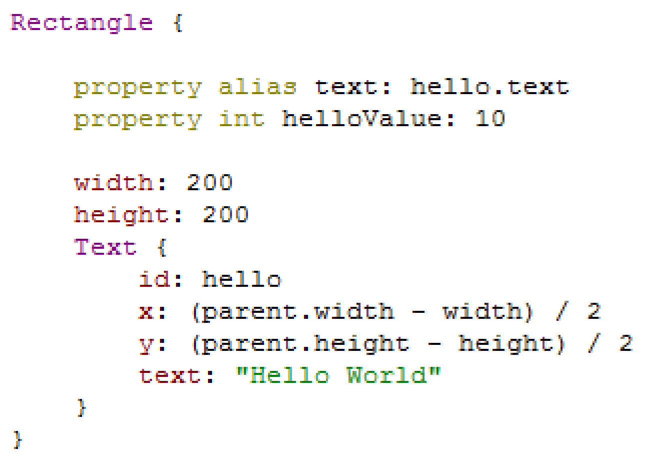
- The basics of properties:
  - *id* is used to reference an object
  - *parent* references the parent object
  - *default* property can be used without a name
    - *data* list is default property of items (like *Rectangle)*

```
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*
  - *variant* for generic data
    - Including JavaScript objects

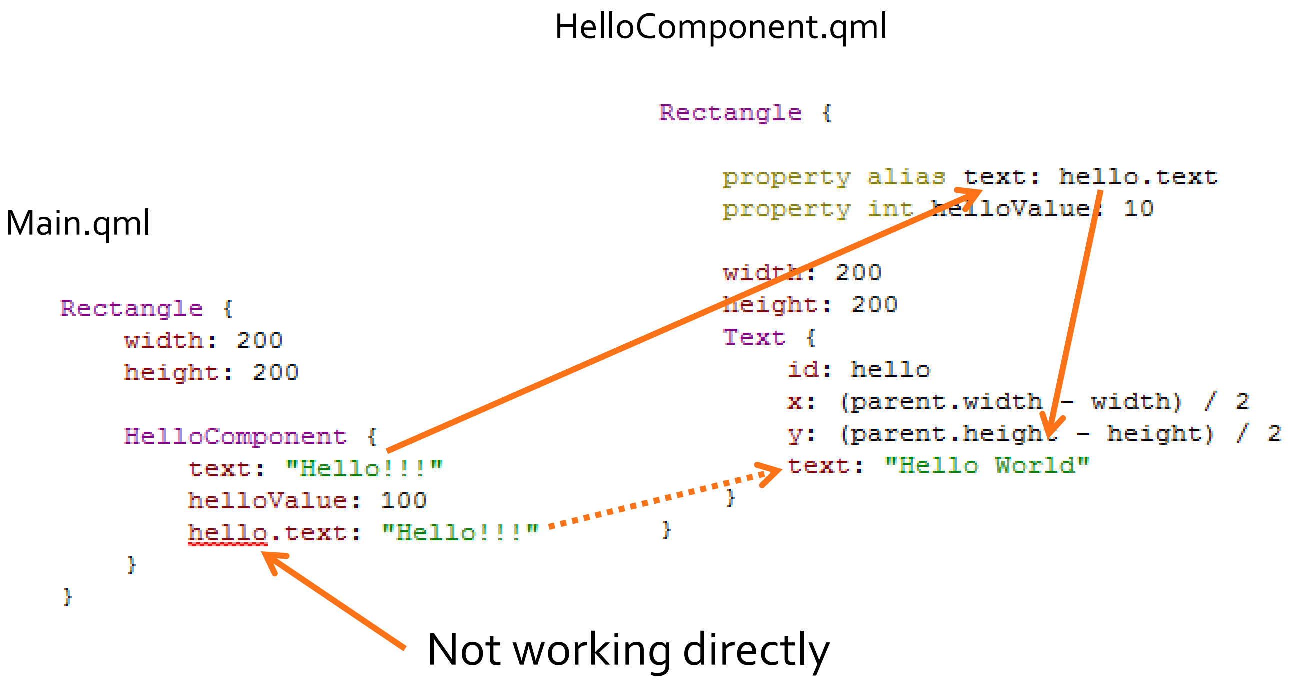


```
Rectangle {

    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-snapshot/qdeclarativebasictypes.html

Alias properties

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



HelloComponent.qml

```
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"
    }
}
```

Main.qml

```
Rectangle {
    width: 200
    height: 200

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

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

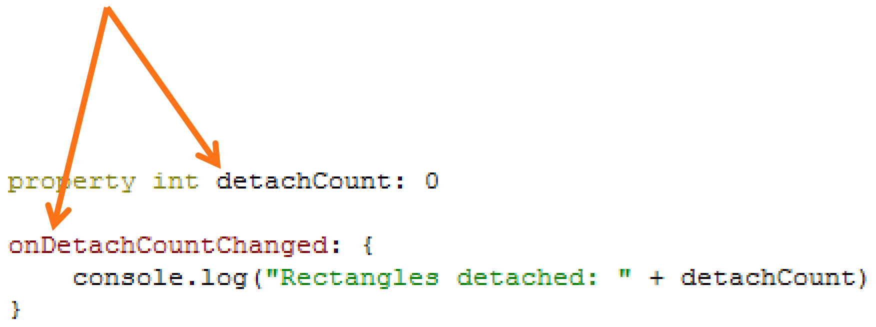
- A component may emit signals, which are processed in *signal handlers*
  - Signal handlers follow *onSignalName* syntax

Mouse click signal handler

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

Signals

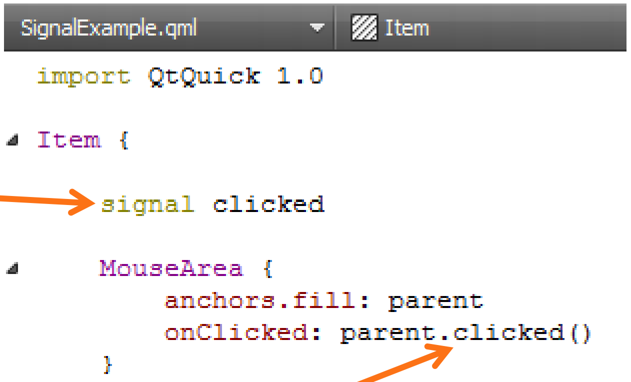
- Property changes may be bound to signal handlers
  - *on<Property>Changed* syntax



```
property int detachCount: 0

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

Signals

- New signals can be defined with *signal* keyword



```
import QtQuick 1.0

Item {

    signal clicked

    MouseArea {
        anchors.fill: parent
        onClicked: parent.clicked()
    }
```

Custom signal

Custom signal handler

Calling the signal

```
SignalExample {
    anchors.fill: parent
    onClicked: console.log("Click delegated here...");
}
```

Functions

- A component may export functions that can be called from other components
  - Note: Not *declarative* way of doing things
    - JavaScript 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()
    }
}
```

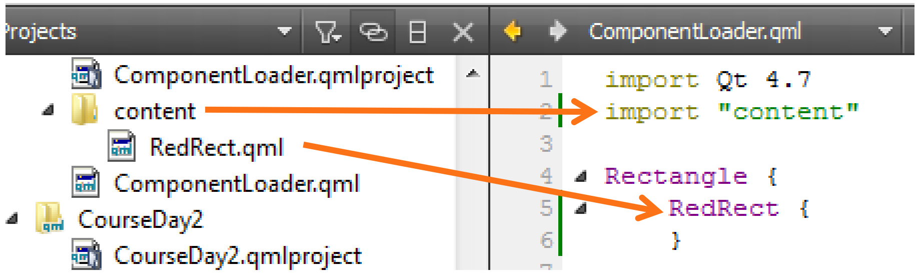
```
Text {
    rotation: parent.rotation

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

Component and script files, dynamic object loading

STRUCTURING QML PROGRAMS

Component files

- The *import* statement can be used to reference QML files in other directories
  - Single file import
  - Directory import



```
import Qt 4.7
import "content"

Rectangle {
    RedRect {
    }
```

- Imported directory can be *scoped*

```
import Qt 4.7
import "content" as Content

Rectangle {
     Content.RedRect {
     }
```

Script files

- The *import* statement also works with JavaScript
  - Can import *files*, not directories
  - Must have the *as* qualifier

```
import "js/startup.js" as Startup

Rectangle {
    Component.onCompleted: Startup.loadItems(rootRect);
```

Property scopes

- Properties of components are visible to child components
  - But, considered bad practice

Main.qml

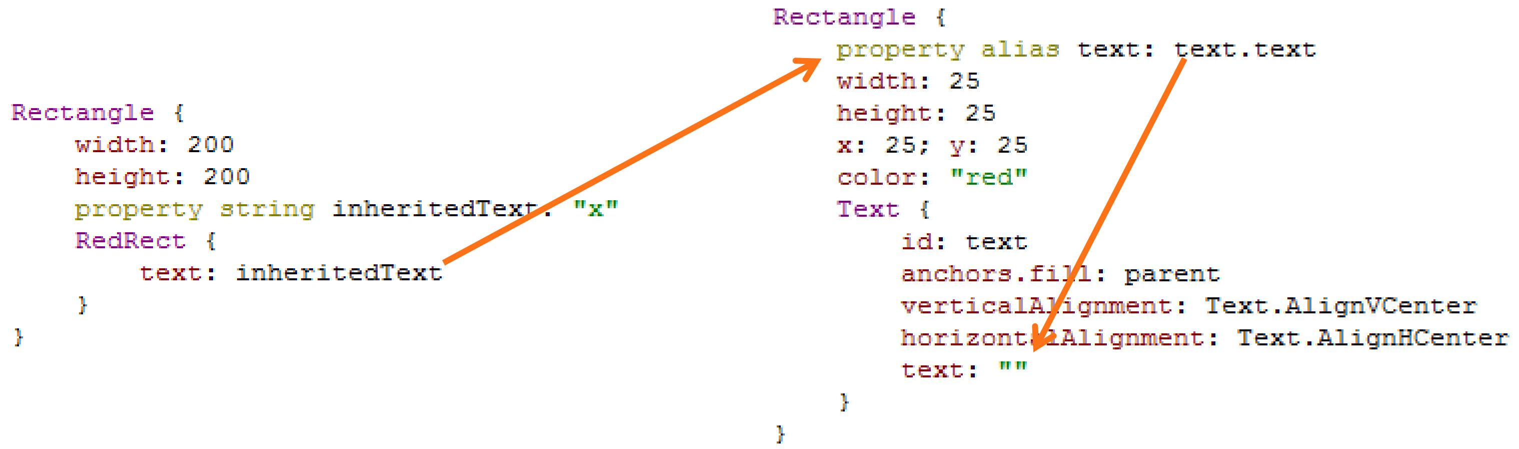
```
Rectangle {
    width: 200
    height: 200
    property string inheritedText: "x"
    RedRect { }
}
```

RedRect.qml

```
Rectangle {
    width: 25
    height: 25
    x: 25; y: 25
    color: "red"
    Text {
        anchors.fill: parent
        verticalAlignment: Text.AlignVCenter
        horizontalAlignment: Text.AlignHCenter
        text: inheritedText
    }
}
```

Property scopes

- Instead, each component should provide an API of it's own

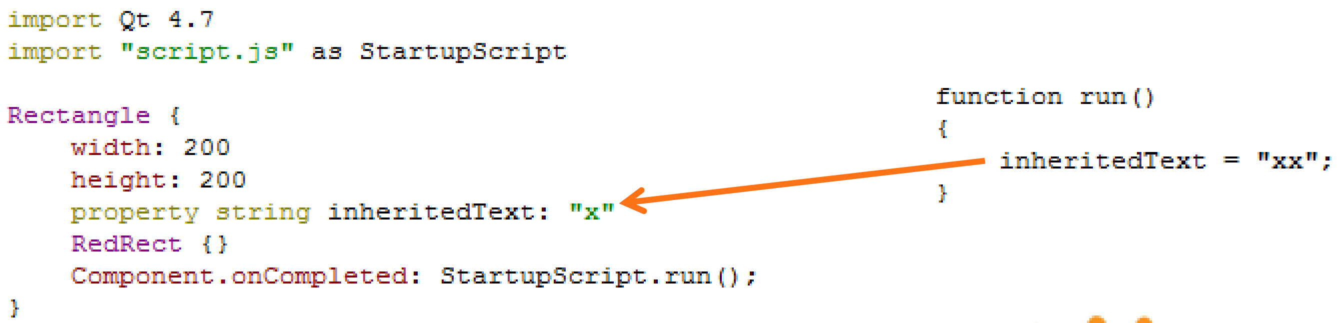


```
Rectangle {
    width: 200
    height: 200
    property string inheritedText: "x"
    RedRect {
        text: inheritedText
    }
}
```

```
Rectangle {
    property alias text: text.text
    width: 25
    height: 25
    x: 25; y: 25
    color: "red"
    Text {
        id: text
        anchors.fill: parent
        verticalAlignment: Text.AlignVCenter
        horizontalAlignment: Text.AlignHCenter
        text: ""
    }
}
```

Script scopes

- Same scoping rules apply to scripts in external JavaScript files
  - i.e. same as replacing the function call with the script
  - Again, not good practice as it makes the program quite confusing

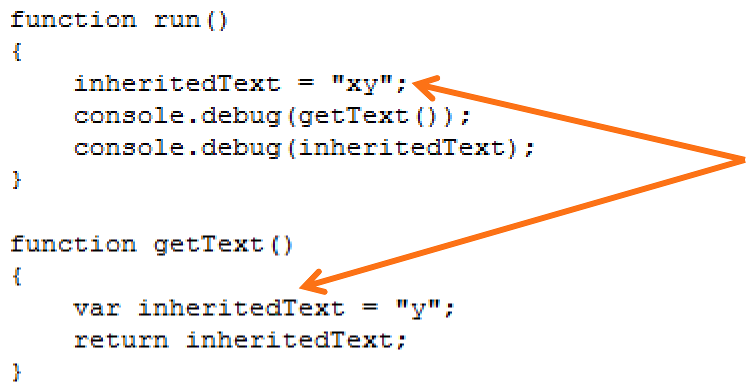


```
import Qt 4.7
import "script.js" as StartupScript

Rectangle {
    width: 200
    height: 200
    property string inheritedText: "x"
    RedRect {}
    Component.onCompleted: StartupScript.run();
}
```

```
function run()
{
    inheritedText = "xx";
}
```

JavaScript scoping

- If script function declares variables with same name, the script variable is used



```
function run()
{
    inheritedText = "xy";
    console.debug(getText());
    console.debug(inheritedText);
}

function getText()
{
    var inheritedText = "y";
    return inheritedText;
}
```

getText uses local variable
run uses inherited one

Inline components

- Components can be declared *inline*
  - *Component* element
  - Useful for small or private components
    - For example data model delegates
  - *Loader* can be used to create instances
    - *Loader* inherits *Item*
    - Can be used to load components from web
- Example in *ComponentLoader* directory

```
Component {
    id: helloComponent
    Text { text: "Loaded from: " + helloComponent.url }
}
```

Dynamic loading

- In addition to *Loader*, components can be loaded dynamically via script code
  - *Qt.createComponent* loads a *Component*
    - File or URL as parameter
  - *component.createObject* creates an instance of the loaded component
    - Parent object as parameter
  - *Qt.createQmlObject* can be used to create QML objects from arbitrary string data
- Example in *ScriptComponents* directory

Visual GUI items

QML GUI BASICS

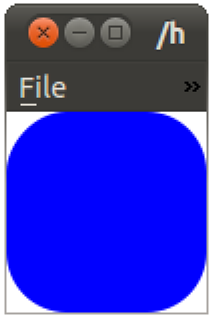
QML Item

- *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

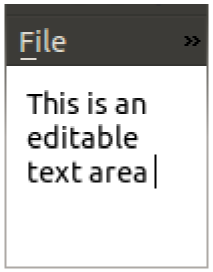
- *Rectangle* and *Image*
  - Basic building blocks
  - *Image* can be loaded from web
- *Text*, *TextInput* and *TextEdit*
  - For non-editable, single-line editable and multiline editable text areas
- And that's about it ☺
  - Qt components project is in progress

```
import Qt 4.7

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



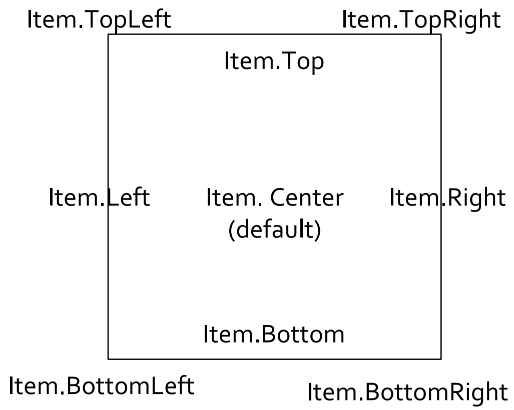
```
Image {
    width: 100
    height: 100
    source: "http://qt.n
}
```

```
Rectangle {
    width: 100
    height: 100

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



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

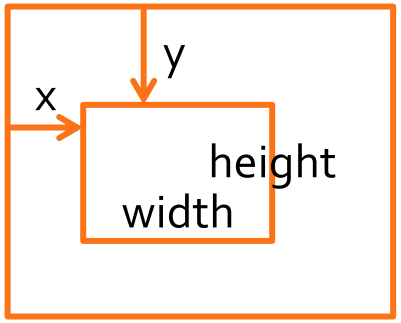
ITEM LAYOUTS

Item layouts

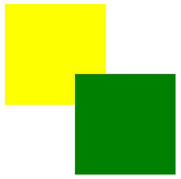
- Relative coordinates
- *Anchors* between items
- *Positioner* objects
  - *Row*, *Column*, *Flow*, *Grid*

Item coordinates

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



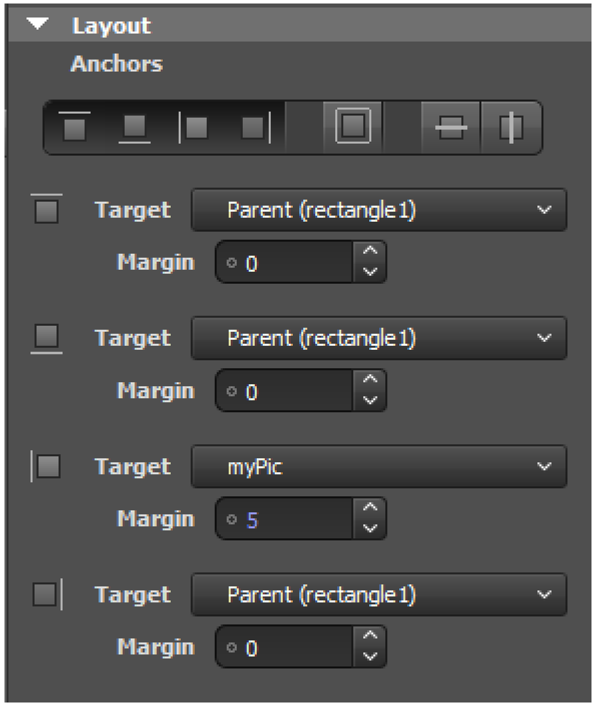
```
Rectangle {
    id: parentRect
    color: "yellow"
    x: 50; y: 50; width: 50; height: 50
    Rectangle {
        id: childRect
        color: "green"
        x: 35; y: 35; width: 50; height: 50
    }
}
```



- Stacking order is controlled by *z*
  - Example in *Coordinates* directory

Item anchors

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



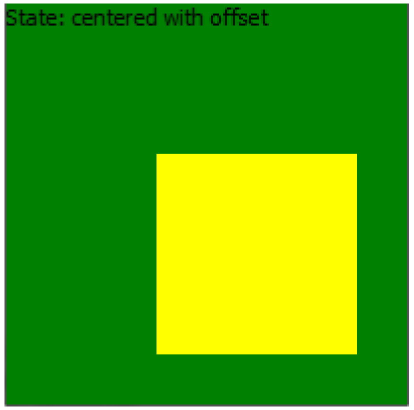
Layout
Anchors

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

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

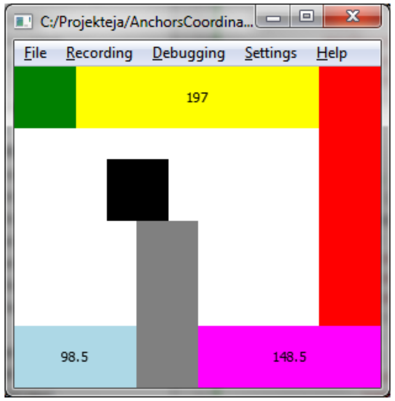
Item anchors

- 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



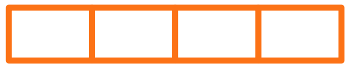
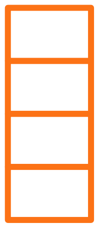
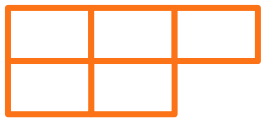
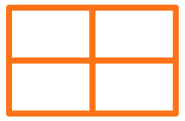
Anchors and coordinates

- 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

- 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

Handling mouse and keyboard input

USER INTERACTION

Mouse and key events

- Mouse and keys are handled via *events*
  - *MouseEvent* contains position and button combination
    - Posted to *Item* under cursor
  - *KeyEvent* contains key that was pressed
    - Posted to *Item*, which has the *active focus*
  - If item doesn’t handle it, event goes to parent
    - When *accepted* properties is set to *true*, the event propagation will stop
  - Events are *signal parameters*

Mouse input

- *MouseArea* element
  - Works for desktop and mobile devices
    - Although, some signals will not be portable
  - *pressed* property
    - Any mouse button (*pressedButtons* for filtering)
    - Finger-press on touch screen
  - Position of events:
    - *mouseX* and *mouseY* properties
    - *mouse* signal parameter

```
MouseArea {
    onClicked: {
        clickX = mouseX
        clickY = mouseY
    }
}
```

```
MouseArea {
    onClicked: {
        clickX = mouse.x
        clickY = mouse.y
    }
}
```

Mouse drag

- *MouseArea* can make an item *draggable*
  - Works with mouse and touch
- Draggable items may contain children with mouse handling of their own
  - The child items must be children of the *MouseArea* that declares dragging
    - *MouseArea* inherits *Item*, so may contain child items
    - *drag.filterChildren* property
- Example in *MouseDrag* directory

Keyboard input

- Each *Item* supports keyboard input
  - *Keys* and *KeyNavigation* attached properties
    - *Keys.on<Key>Pressed* signals
    - *KeyNavigation.up / down / left / right* properties
  - Key events arrive to item with *activeFocus*
    - Can be forwarded to other items
    - Ignored if none of items is focused
  - Setting focus property to *true* to get focus

Keyboard input

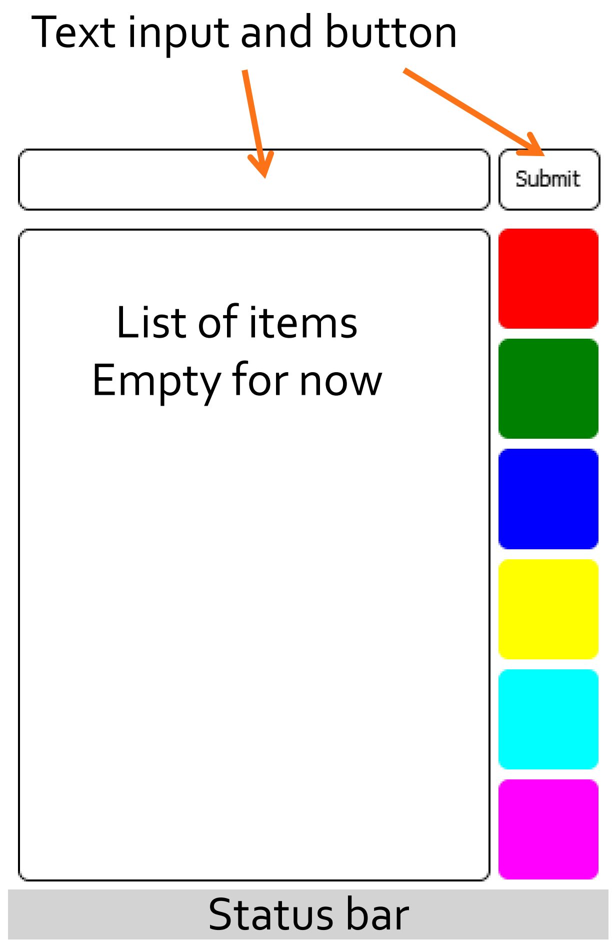
- *FocusScope* element can create focus groups
  - Needed for re-usable components
    - Internals of component are not visible
  - Invisible item, similarly as *MouseArea*
    - One item within each *FocusScope* may have focus
    - Item within the *FocusScope*, which has focus gets key events
- Example in *KeyboardFocus* directory

Getting started with QML

PROGRAMMING EXERCISE

Exercise - layouts

- 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.



<symbio>

SERIOUS ABOUT SOFTWARE