Performance Tips & Tricks for Qt-QML Apps

Rajesh Lal, **Nokia**

High Performance QML Applications

Qt Quick

Qt UI Creation Kit

QML

Most Advanced UI

MeeGo Apps

High Performance

Agenda

What is Qt Quick

- QML
- IDE – Qt Creator
- Qt UI Designer
- Plugin for Mobility API
- Quick Components

The Fastest way to develop for **MeeGo**

What is QML

- Qt Meta-Object Language
- JavaScript based
- Qt Declarative runtime
- CSS-JavaScript like syntax

The **Most Advanced UI** technology for Mobile

QML

Powerful
Declarative
User Interface
language

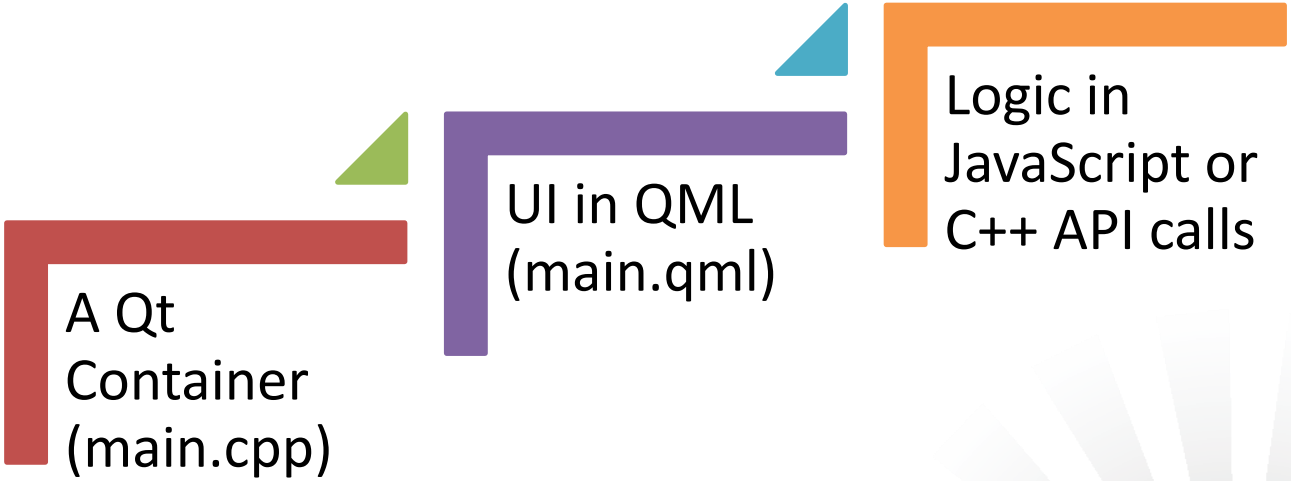
```
import QtQuick 1.0

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

QML Elements

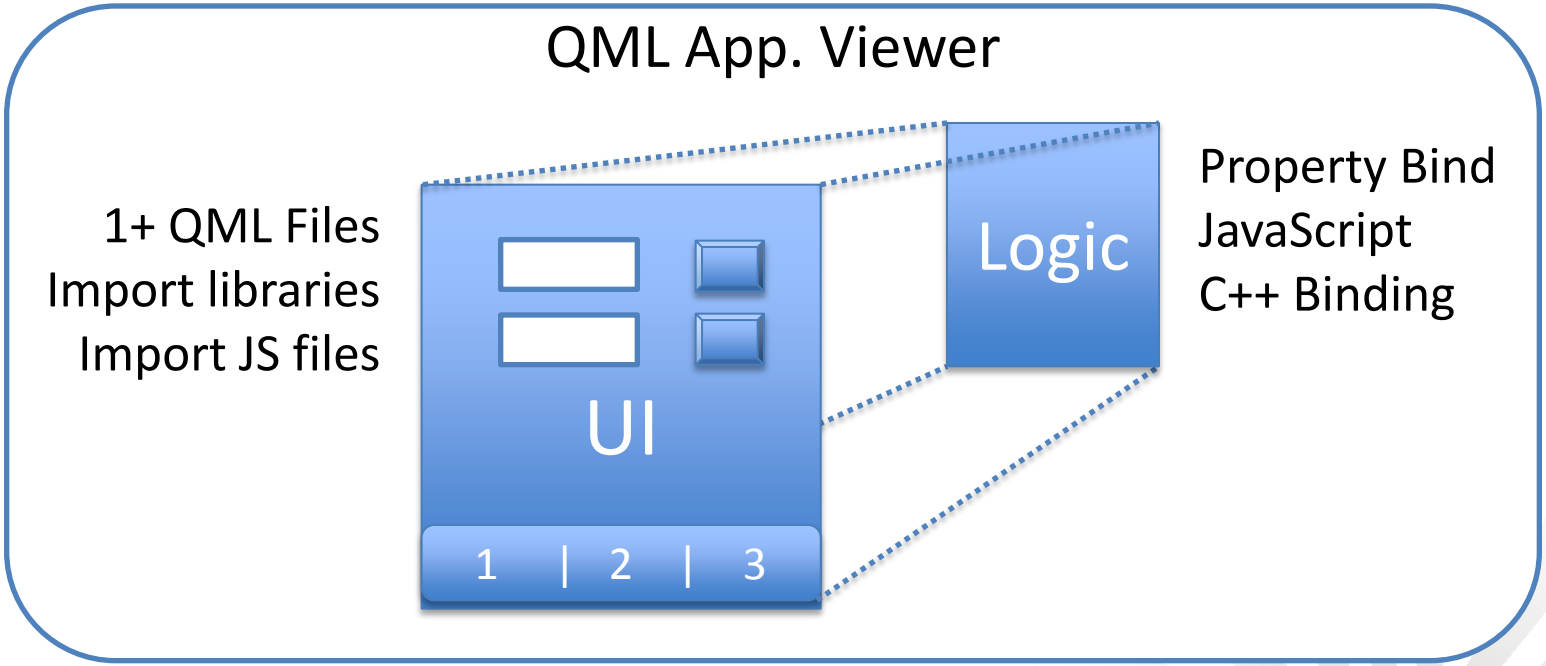
Graphics	Animation	Binding
Items	State	Property
Shapes	Transitions	JavaScript
Text	Transform.	C++

What is a Qt QML App



Qt Quick Application

Qt-QML App



Qt Container(main.cpp)

What is Performance

- Short response time
- High Throughput
- Low utilization of resources
- High availability
- High Speed
- Instant
- Smoothness

7 Tips for
High Performance QML Application

Tip 1
Divide and Rule

1: Divide and Rule

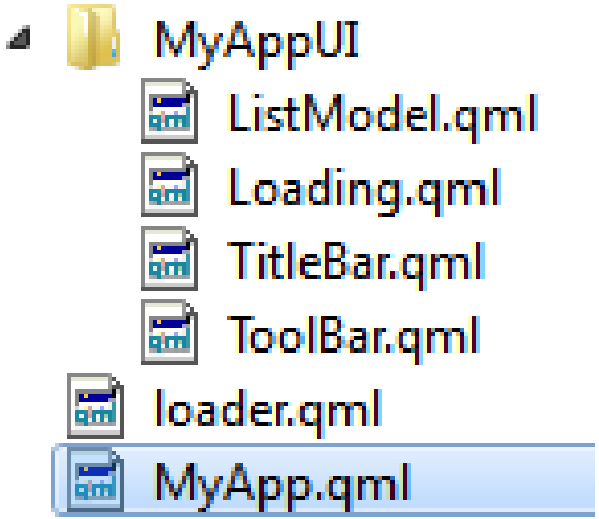
- Divide application UI into multiple QML files
- Each Logical entity as a separate QML file
- Think Object Oriented Programming
- Do not use 1 Large QML file

1: Divide and Rule

Rule

- Use main.qml as the main UI
  - With different States & Transitions
  - Create Multiple Views .QML files
  - Import folders containing other QML files
  - Local instances of imported QML elements

1: Divide and Rule




```
import "MyAppUI" 1.0 as MyAppUI

// Main UI Elements
MyAppUI.ListModel{id:listModel;}
MyAppUI.ListModel{id:historyModel;}
MyAppUI.TitleBar { ...}
MyAppUI.Loading { id:loading; anchors.centerIn: parent;
MyAppUI.ToolBar { id: toolBar; height: 84; ...}
```

Tip 2
Load and Unload

2: Load and Unload

- Use Loader to control the memory usage
- Load the absolute Minimum at the startup
- Dynamically Load and Unload UI Components
- Create Components when required

2: Load and Unload

```
import QtQuick 1.0

Item {
    id:splashscreen
    width: 200; height: 200
    Loader { id: pageLoader }

    MouseArea {
        anchors.fill: parent
        onClicked: pageLoader.source = "MyApp.qml"
    }
}
```

Tip 3
Use Asynchronous Threads

3: Asynchronous Threads

- Use WorkerScripts for Remote API Calls
- Use Threads for time consuming operations
- Load large images asynchronously
- Use caching for remote data/image (cacheBuffer in listview/Gridview)

3: Asynchronous Threads

Login call in MyApp.QML

```
// API Call Scripts
MyAppScript.Login{
    id:workerlogin;
    onMessage:
    {   ...}
}}
```

MyAppScript/Login.QML file

```
import Qt 4.7

WorkerScript {
        id: myWorker
        source: "js/doLogin.js"
}
```

3: Asynchronous Threads

Js/dologin.js

```
WorkerScript.onMessage = function(msg) {
    var doc = new XMLHttpRequest();
    doc.onreadystatechange = function() {
        if (doc.readyState == XMLHttpRequest.HEADERS_RECEIVED) {
        } else if (doc.readyState == XMLHttpRequest.DONE) {
            var response = doc.responseText;
            //console.log('response' + response);
            if (response=="")
                WorkerScript.sendMessage("")
            else
                {
            var myObject = eval('(' + response + ')');
                WorkerScript.sendMessage({returnObj:myObject,
                                          username:msg.username,pwd:msg.password})
            }

        }
    }
    var username = msg.username;
    var password= msg.password;
    var uri = "http://meego.com?blah/finderApiJson.svc?method=auth&service=auth&v=1";
    var txtjson = '[{ "class": "com.finder.NumberPasswordAuthCredential",
    "password": "' + password + '", "phoneNumber": "' + username + '"}]';
    doc.open("POST", uri);
    doc.send(txtjson);
}
```

Tip 4
Optimize for Images
(Greatest User of Memory)

4: Optimize for Images

- Optimize Images to reduce memory footprints
- Specify exact size of the image
- Avoid resizing/scaling an image in QML
- Lazy load large images (asynchronous=true)
- Don’t cache large images (cache=false)
- Smooth filter = better visual quality, but slower

Tip 5
Beware of String Operations

5: Beware of String Operations

- Strings are immutable
- multiple '+' operator = multiple memory allocations
- Use StringBuilder for more efficient strings
- Define
  - #define QT_USE_FAST_CONCATENATION
  - #define QT_USE_FAST_OPERATOR_PLUS

Tip 6
Know States, Transitions, & Animations

6: Know States, Transitions, & Animations

- In transition, animated area should be small
- Animate different items sequentially when possible
- Avoid multiple timers during animation
- Avoid JavaScript operations during animation
- Use ScriptAction & StateChangeScript

Tip 7
Follow Best Practices

7: Best Practices

- For best performance use C++ instead of JavaScript
- Insert properties at top of element declarations
- Create application logic outside QML
- Don’t build multiple layer of QML hierarchies
- Use Qt's i18n for internationalisation & localisation

Tools

QML Performance Monitor
http://www.youtube.com/watch?v=XdI9C53uJw8

Valgrind
http://valgrind.org/docs/manual/mc-manual.html

QML Performance Guidelines
http://doc.qt.nokia.com/4.7/qdeclarativeperformance.html

Thank You

@rajeshlalnokia