Simple Qt timer application in QML

27 Mar 2011

Simple Qt Timer Application in QML - First Steps in QML - the real life experience

The result is here : File:QML First Steps - QmlTimer.zip

Introduction

We are told QML is an easy to learn rapid UI development language, so having studied the docs and tutorial videos for a day I thought I would have a go at porting a simple mobile Qt application to QML and see what could be acheived in a day of messing around and experimenting (plus some time to write it up here). Here is my experiences, I hope you find it useful as a tutorial example of real world development using QML.

This article is in no way a tutorial on QML nor a careful analysis or critique, but an example of what its like as an experienced programmer to pick up the new technology and how far you can get with a minimal investment in time.

References

To get started you need the Qt SDK 1.1 Technology preview from here http://labs.qt.nokia.com/2011/01/20/qt-sdk-1-1-technology-preview-released/

And then consult the documentation online, heres some links I found useful:

QtQuick: http://doc.qt.nokia.com/4.7-snapshot/qtquick.html

Getting started: Getting Started with Qt Quick and the Qt SDK v1.1

Tutorial: http://developer.qt.nokia.com/wiki/Qt_Quick_Tutorial (seems to be still in development)

QML Elements Reference: http://doc.qt.nokia.com/4.7-snapshot/qdeclarativeelements.html

and don't miss the QML presentations in : 50 hours of videos and here

Let us Go

The Qt application I chose is admittedly a fairly basic single screen example, it is called simpleTimer and is a digital timer with start/stop buttons and reset. The Qt application will hopefully be published soon in the OVI store in the meantime it will be freeware here eventually [digitisemylife.com site under construction] and includes up/down counting and an audio alarm. I didn't expect to get everything implemented in QML but aside from learning QML, I did want to see how easy QML state machines would be implement the orientation flip when the device is switched from portrait to landscape mode.

So here's the starting point a numeric display driven by the Timer element:

```
import QtQuick 1.0

Rectangle  {
    id: background;
    width: 360; height: 640
    color: "#343434"

    property int seconds : 0

    Rectangle {
        id : theTimer
        x : 20; y: 20
        width : 320 ; height: 280
```


(C) Copyright Nokia 2014. All rights reserved.

```
        color: "#707070"

        Text {

                text: seconds
                font.pointSize: 72; font.bold: true
                font.family: "Courier"
                anchors.centerIn: parent
            }
    }

    Timer {
        interval: 1000; running: true; repeat: true;
        onTriggered: seconds++;
    }
}
```

That seems simple enough, I created a rectangle set to the screen size of the device for the screen background, then another for the digits which contains some nice large text for the seconds elapsed property to be displayed. The QML Timer element is set to run continuously updating the seconds every 1000 millisecond.

Now every app needs buttons so until there are components for this kind of thing we need to make ourselves a QML button. There are lots of examples, but let us keep it as simple as possible. I added the following after the Timer above:

```
    Rectangle {
            id: button
            width: 160; height: 100
            x : 100 ; y: 340; radius: 10
            color: "#707070"

            Text { text: "Start"; anchors.centerIn: parent}

            MouseArea { id: mouseArea; anchors.fill: parent; }

            Rectangle {
                id: shade
                anchors.fill: parent; radius: 10; color: "black"; opacity: 0
            }

            states: State {
                name: "pressed"; when: mouseArea.pressed == true
                PropertyChanges { target:shade; opacity:0.4}
            }
    }
```

A simple rounded rectangle labeled "Start" with a mouse area for the click detection and a overlay rectangle which changes opacity when the mouse area is pressed to represent the button press. We could get a lot fancier here, I've experimented with gradient fills for a better appearance and font/size reduction on button press for more dynamic interaction, but let us keep it simple here so it is easy to see whats going on and how the code gets structured.

Heres what we have so far:

(C) Copyright Nokia 2014. All rights reserved.

Note: this is a full screen App so you'll need to make the following settings in projects;arguments : *-fullscreen -frameless*.

Let us make the button control the timer; all we need is an enabled property, a conditional in the timer and a setter in the mouse area here are the changes:

Add:

```
    property bool enabled : false
```

where the 'seconds' property is set. Modify our Timer triggered element to:

```
        onTriggered: {if (enabled) seconds++;}
```

and our mouse area definition in the button to:

```
            MouseArea { id: mouseArea; anchors.fill: parent; onClicked:
background.enabled = !background.enabled}
```

Later on I'll tidy this up, as I realized we can use the timer running property directly, but this will do for now.

Run it, you should see the timer can be started and stopped now. But our button says start regardless of whether we're running or not. We'll fix this, but first of all we clearly need to make our button into a reusable component. So I copied it into a separate QML file called . . . of course Button.qml, and added a signal to get the clicked event out, as well as a property alias to be able to change the text. Here's the full implementation:

```
import QtQuick 1.0

Rectangle {
        id: button
```


(C) Copyright Nokia 2014. All rights reserved.

```
        width: 180; height: 100; radius: 10
        color: "#707070"

        property alias text: label.text
        signal clicked

        Text { id: label; text: "Start"; anchors.centerIn: parent}

        MouseArea { id: mouseArea; anchors.fill: parent; onClicked: button.clicked()}

        Rectangle {
            id: shade
            anchors.fill: parent; radius: parent.radius; color: "black"; opacity: 0
        }

        states: State {
            name: "pressed"; when: mouseArea.pressed == true
            PropertyChanges { target:shade; opacity:0.4}
        }
}
```

The nice thing about QML is we can just run this as it is and we get a button that can be clicked in the QmlViewer. Not much use, it appears full screen in the simulator but nevertheless very handy for debugging and testing. In our main file the Button code we've removed gets replaced with a reference to the new Button - note there are no import changes QML just finds the new file if it is in the same directory as the main QML file. Also it is time to tidy up some of those absolute position references so let us have a go at setting anchors on our display and button elements so that the positioning is now all relative:

```
import QtQuick 1.0

Rectangle  {
    id: background;
    width: 360; height: 640
    color: "#343434"

    property int seconds : 0
    property bool enabled : false


    Rectangle {
        id : theTimer
        anchors.left: parent.left; anchors.leftMargin: 20
        anchors.top: parent.top; anchors.topMargin:20
        width : 320 ; height: 280
        color: "#707070"

        Text {

                text: seconds
                font.pointSize: 72; font.bold: true
                font.family: "Courier"
                anchors.centerIn: parent
            }
    }

    Timer {
        interval: 1000; running: true; repeat: true;
```


(C) Copyright Nokia 2014. All rights reserved.

```
        onTriggered: {if (enabled) seconds++;}
    }

    Button{
        id: startButton
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.top: theTimer.bottom; anchors.topMargin: 20
        onClicked: background.enabled = !background.enabled
    }


    states: State {
        name: "enabled"; when: enabled
        PropertyChanges {target: startButton; text:"Stop"}
    }
}
```

So far so good, but it is still pretty unusable as an App. We need to make our timer display a reusable component as well and also let us put in some more buttons. First of all the display follows the same process as the button; we put it in another QML file called . . . Display.qml, here it is:

```
import QtQuick 1.0

Rectangle {
    id : display
    width : 320 ; height: 280
    color: "#707070"

    function count()
    {
        seconds++;
    }
    function reset()
    {
        seconds = 0;
    }

    property int seconds

    Text {
            text: seconds
            font.pointSize: 72; font.bold: true
            font.family: "Courier"
            anchors.centerIn: parent
        }
}
```

When we run the main file (with the reference to display added) it still works, but did you spot the basic error above! There is no property called 'seconds' in the Display component even though it is used there. It works because in QML all children have access to the parents properties, so anything at the top level is effectively a global variable, have to watch that in future, but it is handy for now and probably exactly right for rapid prototyping. We'll fix this in a mo, but for now let us add our extra buttons to QmlTimer:

```
    Button{
        id: resetButton
```


(C) Copyright Nokia 2014. All rights reserved.

```
        text: "Reset"
        height: 70; radius: 5
        anchors.left: parent.left
        anchors.bottom: parent.bottom
        onClicked: theTimer.reset();
    }
    Button{
        id: exitButton
        text: "Exit"
        height: 70; radius: 5
        anchors.right: parent.right
        anchors.bottom: parent.bottom
        onClicked: Qt.quit()
    }
```

Moving on, now we have the display as a component let us have two! One for minutes and one for the seconds. To save space and lots of code in this article, let us at the same time also sort out our seconds property which was in the wrong place earlier, improve our implementation of the enabled function and add a separator to the display. Here's what we have now in the main file:

```
import QtQuick 1.0

Rectangle  {
    id: background;
    width: 360; height: 600
    color: "#343434"


    Item {
        id : theDisplay
        anchors.left: parent.left; anchors.leftMargin: 20
        anchors.top: parent.top; anchors.topMargin:20
        width : 320; height : 180

        property int pointSize : 42

        Display {
            id : minutes
            anchors.top: parent.top
            anchors.left: parent.left
            anchors.right: point.left;
            height : parent.height
            pointSize: parent.pointSize

        }
        Point {
            id : point
            anchors.top: parent.top;
            anchors.centerIn: parent
            width : 30; height : parent.height
            pointSize: parent.pointSize

            onCountOut : seconds.countIn()
        }
```


(C) Copyright Nokia 2014. All rights reserved.

```
        Display {
            id : seconds
            anchors.top: parent.top
            anchors.left: point.right
            anchors.right: parent.right
            height : parent.height
            pointSize: parent.pointSize

            onCountOut : minutes.countIn()
        }
    }

    Timer {
        id:ticker
        interval: 100; running: false; repeat: true;
        onTriggered: point.countIn()
    }

    Button{
        id: startButton
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.top: theDisplay.bottom; anchors.topMargin: 50
        text: "Start"

        onClicked: ticker.running = !ticker.running
    }


    Button{
        id: resetButton
        text: "Reset"
        height: 70; radius: 5
        anchors.left: parent.left
        anchors.bottom: parent.bottom
        onClicked: {seconds.reset(); minutes.reset();}
    }
    Button{
        id: exitButton
        text: "Exit"
        height: 70; radius: 5
        anchors.right: parent.right
        anchors.bottom: parent.bottom
        onClicked: Qt.quit()
    }
}
```

and the Display.qml file contains:

```
import QtQuick 1.0

Rectangle {
    id : display
    width : 320 ; height: 280
    color: "#707070"

    function countIn()
```


(C) Copyright Nokia 2014. All rights reserved.

```
    {
        if (seconds == 59)
        {
            seconds = 0;
            countOut();
        }
        else
            seconds++;
    }
    function reset()
    {
        seconds = 0;
    }

    property int seconds

    signal countOut
    property int pointSize : 72

    function formatOutput()
    {
        if (seconds < 10)
            return '0' + seconds
        else
            return seconds
    }


    Text {
            text: formatOutput()
            font.pointSize: pointSize; font.bold: true
            font.family: "Courier"
            anchors.centerIn: parent
        }
}
```

Button.qml is unchanged.

You notice I implemented the display separator as a Point component, I'll leave the implementation out here, as this article is getting a bit long, but you can take a look at the source if you download the attached zip file for the complete 'App'. Point and Display have countOut signals on overflow so, we simply thread the signal through our components to get a minutes and seconds timer with a flashing colon.

Heres what we have now

(C) Copyright Nokia 2014. All rights reserved.

Switching Orientation

OK so running out of time now, but how about the orientation switch. If running in the QMLVIewer there is a runtime.Orientation property which I didn't try, as it is Viewer specific. I believe something similar would need implementing in QDeclarativeView if this QML was wrapped by your own binary. Also that would probably need the Mobility API and hence affect the signing privileges required. Lets keep it simple (ish), we can just detect the size change of the background to trigger our state change from portrait to landscape like so:

```
    states: State {
        name: "landscape"; when: background.width >= background.height
...
```

Seems to work. So there's a few properties to change and re-anchoring thats required, this is what I ended up with:

```
    states: State {
        name: "landscape"; when: background.width >= background.height
        PropertyChanges { target: resetButton; width:120}
        AnchorChanges {target:resetButton; anchors.right: background.right;
anchors.left: undefined; anchors.bottom: undefined;anchors.top: background.top}
        PropertyChanges { target: exitButton; width:120; y:290}
        AnchorChanges {target:exitButton; anchors.bottom: undefined}

        PropertyChanges { target: startButton; width:120; height:70; y:145;
anchors.topMargin: 0}
        AnchorChanges {target:startButton; anchors.right: background.right;
anchors.left: undefined; anchors.bottom: undefined;anchors.top:
undefined;anchors.horizontalCenter:undefined}
```


(C) Copyright Nokia 2014. All rights reserved.

```
        PropertyChanges { target: theDisplay; height: 320; width: 480; pointSize : 66}
    }
```

We can see the advantage of putting the Display in one 'box' now; less properties to change.

Try this in the simulator and switch the orientation, we now have a nice big display in landscape mode with clearly visible timer characters even from a distance and good use of the display area.

and here it is in landscape:

Almost forgot its quite simple to add a basic transition to make the orientation switch look a little neater. Lets put this in for now:

```
    transitions:
        Transition { id:trans
            from: "*"; to: "landscape"; reversible : false
        AnchorAnimation { duration: 800 }
        NumberAnimation {properties: "y, width, height, pointSize"; easing.type:
Easing.InCurve; duration: 800}
    }
```

OK so I'll have to stop there - time is up.There's a few more things I would like to have done like down counting, and better fancier buttons, and transitions on switching back to portrait but nevertheless given the time limit I set myself and patent QML amateur status thats not bad. Another day would probably sort it and then we'd have to call the designers in !

Lessons

So what caused me problems, well it took a little while to get used to the signal mechanism, I kept wanting to use the connect method to wire things together and although there is one in QML in this case its not necessary, also there is no formal SLOT type definition, it turns out all accessible javascript functions can be slots.

Also the states handling nearly got messy when I had the 'enabled' state and wanted to add the 'landscape' state; the design required these to be independent state machines but the QML loader objected to states being declared twice. However, in the end it was a better implementation without the 'enabled' state anyway so the slight refactoring above cured that.

Anchoring elements at the bottom of the screen after the orientation switch didn't work, the screen height seems to be wrong, so I had to put in an absolute y position for the Exit button until this can be investigated further.

But I suppose the most important lesson is :- if I can do it, so can you. Just a couple of days of your time invested in QML is all it takes to get started with (almost) a real application like this. The tools work well and the language seems well thought out and very forgiving.

There is still lots to learn of course and its also still quite early for QML, I think a set of reusable components (buttons..) are on the way, and of course it may not be the best thing to use as the sole language for large formal type safe applications, but nevertheless for the case its designed for; User interfaces and rapid prototyping, I found it really is 'Quick' and quite fun.

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.