Device independent layout for QML

This article needs to be updated: If you found this article useful, please fix the problems below then delete the {{ArticleNeedsUpdate}} template from the article to remove this warning.

Reasons: hamishwillee (21 Feb 2012)
Article provides only part of the layout/scalability/orientation story. Suggest it should be extended to include additional information from the linked SeeAlso section. I've also added some Comments in the talk page below.

QML components often need a *width* and *height* property, but setting explicit values causes issues when attempting to use the same code on different platforms. For example. a QML application that runs on a Nokia N900 may be designed for the 800 by 400 pixel screen, but the same application on a desktop may be used at 800 by 600 pixels, or it may be resized to be larger or smaller. The solution is to assign a size to the top-level item in your qml file and make all subsequent sizes relative to the main one. The root size should be made big enough so that the item, when viewed by itself in the qmlviewer for testing, is large enough and properly proportioned to be useful. But if all the other sizes are dependent on the root size, the item can be resized with impunity.

Take the following component (saved as *ColorRectangle.qml*, which creates a colored rectangle with a word in it:

```
import Qt 4.7

Item {
    width: 250   //these are the only explicit sizes set
    height: 250   //all others are relative
    property alias rectColor: rect.color
    property alias rectText: rectText.text
    Rectangle {
        id: rect
        color: "red"
        anchors.fill: parent
        Text {
            id: rectText
            anchors.centerIn: parent
            text: "RED"
            font.pixelSize: parent.height * 0.25
        }
    }
}
```

If this is viewed in qmlviewer, it shows up as a red square. Note that it can be resized, and the text grows and shrinks relative to the overall size.

Now we use this in another qml file called *Sizing.qml* which is saved in the same directory with *ColorRectangle.qml*:

```
import Qt 4.7

Item {
    width: 800    //these are the only explicit sizes set
    height: 600   //all others are relative

    Grid {
        width: parent.width
        height: parent.height
        columns: 2
        ColorRectangle {
            rectColor: "red"
            rectText: "RED"
            width: parent.width / 2  //our component is resized
            height: parent.height / 2
        }
        ColorRectangle {
            rectColor: "green"
            rectText: "GREEN"
            width: parent.width / 2
            height: parent.height / 2
        }
        ColorRectangle {
            rectColor: "orange"
            rectText: "ORANGE"
            width: parent.width / 2
            height: parent.height / 2
        }
        ColorRectangle {
            rectColor: "purple"
            rectText: "PURPLE"
            width: parent.width / 2
            height: parent.height / 2
        }
    }

    Rectangle {
        width: parent.width * 0.1
        height: width
        anchors.centerIn: parent
        color: "gray"
        radius: height/10
    }
}
```

When we run this in qmlviewer we see that we have resized our *ColorRectangle* objects to be one quarter of the total size, and the text size grows with the rectangle size. We can resize this window (it only lets us grow larger than the defined size, not smaller) and all the objects resize cleanly. This will look good at nearly any common aspect ratio or size combination, and because we made everything relative there is no further work to do in resizing.

We have also added an additional rectangle that sits on top of the others and is centered. This has its *height* set to be equal to its *width*, and no matter how we resize our window, it always looks square. Also, the *radius* property has been designated as a tenth of the "height" property, so that as it resizes the roundness of the corner will look the same.

(C) Copyright Nokia 2013. All rights reserved.

And object sizes can be related to more than just the parents--objects can refer to one another, as long as you are careful not to make the references circular. Object A can have a size of *parent.width / 3*; Object B can have a size of *A.width*; Object C can then have a size of *parent.width - A.width - B.width*. You can even set a property (e.g *property int buttonWidth: main.width/10* and set all of your button objects to have *width: buttonWidth*. However it is done, the key is to avoid using real numbers anywhere but in the root element.

One additional note: the sizes can be divided or multiplied, and whole numbers or decimals can be used to get the precise look you want.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.