Pull to Refresh feature in QML

This article explains how to implement Pull to Refresh feature in QML apps, though the method is a bit heavy, but it's quite foolproof and elegant

Introduction

The code is simple and self-explanatory. The extra heaviness is due to the added ability to cancel a pull after pulling down, and also to avoid accidental unintentional pulling requests. There's a timer which induces a latency to get confirmed request from the pull. Might sound simple, but there were quite a few pit falls.... I'm open to modifications and further improvements. Feel free to contribute and suggest :)

Instructions

- You've to put the whole Item { } as a child of the ListView{} where you wan to implement pull to refresh.
- You can edit the two values of 600 depending on how much latency you want
- ListView id is taken as "itemView". It has to be changed according to your project

CODE

```
Item {
    property bool puller : false
    id: pull
    width: parent.width
    opacity: (pullImage.rotation*-1)/135
    y: (itemView.contentY+pullImage.height+10)*-1
    Image {
        id: pullImage
        smooth: true
        source: invertedTheme? "images/refreshI.svg" : "images/refresh.svg"  //change the invertedTheme checker property name
        anchors.left: parent.left
        anchors.leftMargin: pullImage.height/2
        rotation: 2*360*itemView.contentY/itemView.height
        onRotationChanged: {
            if(pullImage.rotation*-1>135){
                if(!pullTimer.running && !pull.puller)
                pullTimer.restart()
            }
            if(pullImage.rotation*-1<135){
                if(!pullTimer.running && pull.puller)
                pullTimer.restart()
            }
        }
        Timer{
            id:pullTimer
            interval:600  //latency value
            onTriggered: {
                if(pullImage.rotation*-1>135){
                    pull.puller = true
                }
                else {
                    pull.puller = false
                }
            }
        }
    }
    Label{
        id: pullLabel
        platformInverted: wTheme
        text: "Pull and hold to refresh.."
        anchors.left: pullImage.right
        anchors.leftMargin: pullImage.height/2
    }
    Rectangle{
        id:pullRect
        smooth: true
        opacity:0.5
        anchors.bottom: pullLabel.bottom
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.bottomMargin: -10
        color: "steel blue"
        height: 1
        width: (pullImage.rotation*-1>135)?parent.width:0
        Behavior on width {
            NumberAnimation{
                duration: 600  //latency value. Keep same as earlier
            }
        }
        onWidthChanged: {
            if(width == 0)
                pullLabel.text = "Pull and hold to refresh.."
            if(width == parent.width)
                pullLabel.text = "Release to refresh.."
        }
    }
}


//Put the following code in the listView element as the onMovementEnded property

onMovementEnded: {
    if(pull.puller){
        console.log("REFRESH EVENT")
        REQUIRED_Function() // Change it as needed
    }
    pull.puller = false
```


(C) Copyright Nokia 2013. All rights reserved.

```
    pullTimer.stop()
}
```

Another sample of ready for usage ListView extended component

This is a bit simplified version with clearer code. Moreover no third-party graphic resources are used, only 'toolbar-refresh' from Symbian Qt Components. The following source code could be placed into ListViewEx.qml file (or any like that), though ListViewEx component would be available for usage.

```
import QtQuick 1.1
import com.nokia.symbian 1.1

ListView {
    id: root

    property int latency: 600
    property int rotationThreshold: 135
    property string pullMessageString: "Pull and hold to refresh..."
    property string releaseMessageString: "Release to refresh..."
    property bool platformInverted: false

    signal refreshEvent()

    Item {
        property bool __puller : false

        id: pull
        width: parent.width
        opacity: -pullImage.rotation / root.rotationThreshold
        y: -(root.contentY + pullImage.height + labelRow.spacing)

        Row {
            id: labelRow
            anchors.left: parent.left
            anchors.leftMargin: spacing
            spacing: pullImage.width / 2
            width: pullImage.width + pullLabel.width + spacing

            Image {
                id: pullImage
                smooth: true
                source: privateStyle.toolBarIconPath("toolbar-refresh", root.platformInverted)
                rotation: 2 * 360 * root.contentY / root.height
                onRotationChanged: {
                    if (pullImage.rotation < -root.rotationThreshold){
                        if (!pullTimer.running && !pull.__puller)
                            pullTimer.restart()
                    }
                    else if (pullImage.rotation > -root.rotationThreshold){
                        if (!pullTimer.running && pull.__puller)
                            pullTimer.restart()
                    }
                }

                Timer{
                    id: pullTimer
                    interval: root.latency

                    onTriggered: {
                        if(pullImage.rotation < -root.rotationThreshold)
                            pull.__puller = true
                        else
                            pull.__puller = false
                    }
                }
            }

            Label {
                id: pullLabel
                platformInverted: root.platformInverted
                text: {
                    if (pull.__puller)
                        return root.releaseMessageString

                    return root.pullMessageString
                }
            }
        }
    }

    onMovementEnded: {
        if (pull.__puller)
            root.refreshEvent()

        pull.__puller = false
        pullTimer.stop()
    }
}
```


(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.