

Implementing custom orientation changes animation with QML

This code example shows how to override the default QML orientation change animation with your own *custom* orientation change. The orientation change is detected with `QOrientationSensor`.

Overview

Qt Quick Components provide a standard animation between orientations on Symbian (in Maemo the orientation is already locked to the landscape mode). If the orientation is not locked, QML apps can monitor the change through the `QML Window` element, or by observing the width and height of the main window and seeing which one is the greatest. This approach should be perfectly acceptable for almost all Qt apps.

This code example shows how you can change the default animation if you need to. This is achieved by first locking the orientation to portrait mode to prevent the default animation occurring. We then use the `QOrientationSensor` to detect when the orientation has changed, provide this data to QML using Signals and Slots, and perform the animation in QML.

Pros and cons of this approach:

- + Nice animation to the orientation change
- + No black screen during the orientation change
- Custom implementation is required to switch to landscape orientation when HW-keyboard is used
- Additional (though user-grantable) capability `ReadDeviceData` is required to allow reading of sensors on Symbian
- Qt Mobility libraries are required to detect the orientation

Download the sources of this snippet [here](#).

Preconditions

- Qt 4.7 or higher is installed on your platform
- QtMobility 1.0.2 or higher is installed on your platform

orientation.pro

The declarative module is used, as well as the Qt Mobility sensors. QML code will be placed inside Qt resource system for easier deployment to the devices.

```

QT             += core gui declarative

TARGET        = orientation
TEMPLATE      = app

SOURCES       += main.cpp
HEADERS       += orientationfilter.h
OTHER_FILES   += Orientation.qml
RESOURCES     = resources.qrc

CONFIG        += mobility
MOBILITY      += sensors

symbian {
    # To lock the application to landscape orientation
    LIBS += -lcone -leikcore -lavkon
}

unix:!symbian {
    maemo5 {
        target.path = /opt/usr/bin
    } else {
        target.path = /usr/local/bin
    }
    INSTALLS += target
}

```

main.cpp

The `main.cpp` creates the `QOrientationSensor` and the self-implemented `OrientationFilter` which will signal the changes of the orientation. This signal is connected to the QML documents root elements function `orientationChanged`. The root element is retrieved by using code `view.rootObject()`.

The width and height of the root element are set to be resized to `QDeclarativeViews` sizes with the code `view.setResizeMode(QDeclarativeView::SizeRootObjectToView)`.

```

#include <QApplication>
#include <QDeclarativeView>
#include <QGraphicsObject>
#include <QDesktopWidget>
#include <QOrientationSensor>
#include "orientationfilter.h"

QTM_USE_NAMESPACE

#ifdef Q_OS_SYMBIAN
    // Lock orientation in Symbian
    #include <eikenv.h>

```

```

#include <eikappui.h>
#include <aknenv.h>
#include <aknappui.h>
#endif

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

#ifdef Q_OS_SYMBIAN
    // Lock orientation in Symbian
    CAknAppUi* appUi = dynamic_cast<CAknAppUi*> (CEikonEnv::Static()->AppUi());
    TRAP_IGNORE( if(appUi) { appUi->SetOrientationL(CAknAppUi::EAppUiOrientationPortrait); } );
#endif

    QDeclarativeView view;
    view.setSource(QUrl("qrc:/Orientation.qml"));
    view.setResizeMode(QDeclarativeView::SizeRootObjectToView);

    QOrientationSensor sensor;
    OrientationFilter filter;
    sensor.addFilter(&filter);

    // Connect the Qt signal to QML function
    QObject::connect(&filter, SIGNAL(orientationChanged(const QVariant&)), view.rootObject(), SLOT(orientationChanged(const QVariant&)));
    sensor.start();

    // Show application in full screen on all platforms with corresponding device resolution
    view.setGeometry(QApplication::desktop()->screenGeometry());
    view.showFullScreen();

    return app.exec();
}

```

orientationfilter.h

The self-implemented `OrientationFilter` is derived from `QObject` to get the possibility to emit signals and from `QOrientationFilter` to read the values of the sensor. The sensor reading is placed to `QVariant` so that the value can be passed to JavaScript function in QML code.

```

#ifndef ORIENTATIONFILTER_H
#define ORIENTATIONFILTER_H

#include <QOrientationFilter>

QTM_USE_NAMESPACE

class OrientationFilter : public QObject, public QOrientationFilter
{
    Q_OBJECT
public:
    bool filter(QOrientationReading *reading) {
        emit orientationChanged(reading->orientation());

        // don't store the reading in the sensor
        return false;
    }

signals:
    void orientationChanged(const QVariant &orientation);
};

#endif // ORIENTATIONFILTER_H

```

Orientation.qml

The UI holds six Text elements inside a Column representing orientations of the device. Current orientation text will be shown as bold and white color as the others are shown in black without bold. The view will be rotated to the current orientation by altering the rotation property of the view. Also the width and height of the view are updated to match the corresponding orientation.

```

import Qt 4.7

Item {
    id: base

    function orientationChanged(orientation) {
        highlightindex(orientation)

        if(orientation == 1) {
            view.rotation = 0
            view.width = base.width; view.height = base.height
        }
        else if(orientation == 2) {
            view.rotation = 180
            view.width = base.width; view.height = base.height
        }
        else if(orientation == 3) {
            view.rotation = 270
            view.width = base.height; view.height = base.width
        }
        else if(orientation == 4) {
            view.rotation = 90
            view.width = base.height; view.height = base.width
        }
    }

    function highlightindex(orientation) {
        var count = texts.children.length

        for(var i=0; i<count; i++) {
            if(i == (orientation - 1)) {
                texts.children[i].color = "white"
                texts.children[i].font.bold = true
            }
            else {

```

```
        texts.children[i].color = "black"
        texts.children[i].font.bold = false
    }
}
Rectangle {
    id: view

    Behavior on rotation { RotationAnimation { direction: RotationAnimation.Shortest; duration: 500; easing.type: Easing.OutBoun
    Behavior on width { NumberAnimation { duration: 500 } }
    Behavior on height { NumberAnimation { duration: 500 } }

    anchors.centerIn: parent
    width: base.width; height: base.height
    color: "blue"

    Column {
        id: texts

        anchors.centerIn: parent
        spacing: 10

        Text { text: "Top up" }
        Text { text: "Top Down" }
        Text { text: "Left Up" }
        Text { text: "Right Up" }
        Text { text: "Face Up" }
        Text { text: "Face Down" }
    }
}
}
```

Postconditions

The snippet demonstrated the implementing of nice animation to orientation change with QML language. The orientation data was provided by Qt Mobility's `QOrientationSensor` class. The retrieved data was transferred via Qt Signals and Slots to QML code where the current orientation was nicely visualized.

