Calibrating the magnetometer sensor in Qt

This snippet shows how to calibrate the device's magnetometer sensor by physically rotating the phone and how to handle the calibration level information in the Qt / QML code.

Calibrating the device

The device is calibrated by rotating the device through all of its axes. A good way to do this is to rotate the device in a figure of eight as indicated by the picture here. Rotating the device in the Möbius strip at the same time speeds up the calibration.

Calibrate by moving along the path of a Möbius strip

User experience considerations

An application should guide the user to perform the calibration when needed. Whether this is a part of the starting procedure of the application or performed only when needed, depends on the case. Either way, the procedure should be clear to the user and the user must be informed and notified when the calibration is complete. As a reference, the following Compass example does several things to guide the user properly, besides the figure and textual instruction, such as:

- vibra is used to indicate that the user is expected to perform the calibration
- progress of the calibration is shown
- when the calibration is ready, a sound is played and a "calibration ready" note is shown

after which the application automatically proceeds to the compass functions. Using an audio notification to indicate that the sensor has been calibrated is highly recommended since it is difficult for the user to monitor the screen while calibrating the device.

Detecting the calibration level in code

Preconditions

Qt Mobility 1.1 or higher is installed.

Qt / QML code

Add the Qt Mobility project configuration option in the .pro file as shown below

```
CONFIG += mobility
MOBILITY += sensors
```

compassfilter.h

```
#ifndef COMPASSFILTER_H
#define COMPASSFILTER_H

#include <QCompassFilter>
#include <QVariant>

QTM_USE_NAMESPACE


class CompassFilter :
```


(C) Copyright Nokia 2014. All rights reserved.

```
    public QObject,
    public QCompassFilter
{
    Q_OBJECT

public:
    CompassFilter();
    bool filter(QCompassReading *reading);

signals:
    void azimuthChanged(const QVariant &azimuth, const QVariant &calibrationLevel);
};

#endif // COMPASSFILTER_H
```

compassfilter.cpp

```
#include "compassfilter.h"

CompassFilter::CompassFilter()
{
}

bool CompassFilter::filter(QCompassReading *reading)
{
    // The azimuth and calibration level are retrieved through the
    // reading object. The calibration level has values 0..1 where
    // the 1.0 means that the sensor is fully calibrated.
    emit azimuthChanged(reading->azimuth(), reading->calibrationLevel());
    return false;
}
```

main.cpp

```
#include <QCompass>
#include "compassfilter.h"

    // ...

    QDeclarativeView view;
    view.setSource(QUrl("qrc:/qml/Ui.qml"));
    QObject *rootObject = dynamic_cast<QObject*>(view->rootObject());

    QCompass compass;
    CompassFilter filter;
    compass.addFilter(&filter);

    connect(&filter, SIGNAL(azimuthChanged(const QVariant&, const QVariant&)),
            rootObject, SLOT(handleAzimuth(const QVariant&, const QVariant&)));

    compass.start();

    //...
```

Ui.qml

(C) Copyright Nokia 2014. All rights reserved.

```
#import QtQuick 1.0

Rectangle {
    function handleAzimuth(azimuth, calibrationLevel) {
        if(calibrationLevel < 1.0) {
            // show calibration dialog with calibration level
        }
        else {
            // show compass with azimuth
        }
    }
}
```

Postconditions

The code snippet demonstrated how to calibrate the device by rotating it through all of its axes. The snippet also demonstrated how to retrieve the calibration level on a Qt / QML codes.

(C) Copyright Nokia 2014. All rights reserved.