Table of content

Using QtMobility sensors and QML from PySide

This PySide tutorial shows how to use QtMobility APIs to read the accelerometer from Python, scale and smoothen the resulting data and expose it to a QML application in order to keep an image always upright. In the future, Qt Mobility 1.2 (still not released as of December 2010) will have QML Plugins [doc.qt.nokia.com], but right now we have to write some glue code in Python (and one might want to do more with the accelerometer data than just using it in the UI layer, so this will still be relevant when Qt Mobility 1.2 is out).

UnderMeSensi.py

Importing the required modules

This is basically the same as in the previous tutorials (PySide modules needed, the QtOpenGL module is optional) with the new addition of the QtMobility Sensors API. On your N900, you have to install the python-qtmobility metapackage in order to get the right modules.

```
import sys

from PySide import QtCore, QtGui, QtDeclarative, QtOpenGL
from QtMobility import Sensors
```

The listener / controller

Next, we need to define a QObject subclass that takes care of receiving events from the accelerometer, scaling and smoothing the value and finally exposing the calculated rotation value as property so that we can access it from within our QML UI:

```
class Listener(QtCore.QObject):
    def __init__(self):
        QtCore.QObject.__init__(self)
        self._initial = True
        self._rotation = 0.

    def get_rotation(self):
        return self._rotation

    def set_rotation(self, rotation):
        if self._initial:
            self._rotation = rotation
            self._initial = False
        else:
            # Smooth the accelermeter input changes
            self._rotation *= .8
            self._rotation += .2*rotation

        self.on_rotation.emit()

    on_rotation = QtCore.Signal()
    rotation = QtCore.Property(float, get_rotation, set_rotation, \
            notify=on_rotation)

    @QtCore.Slot()
    def on_reading_changed(self):
        accel = self.sender()
        # Scale the x axis reading to keep the image roughly steady
        self.rotation = accel.reading().x()*7
```

Putting it all together

We create a new QAccelerometer from the Sensors module, which abstracts away the underlying system accelerometer and sends us easy-to-use events. We then create an instance of our listener class, and connect the readingChanged signal of the accelerometer (which gets called every time the reading changes, obviously) to the listener's on_reading_changed slot. We also have to tell the accelerometer to start reading the sensor and send out events.

We then only need to set up our QDeclarativeView as usual, and expose our listener object to the QML context, so that we can access it from the UI:

```
app = QtGui.QApplication(sys.argv)

accel = Sensors.QAccelerometer()
listener = Listener()
accel.readingChanged.connect(listener.on_reading_changed)
accel.start()

view = QtDeclarative.QDeclarativeView()
glw = QtOpenGL.QGLWidget()
view.setViewport(glw)
view.setResizeMode(QtDeclarative.QDeclarativeView.SizeRootObjectToView)
view.rootContext().setContextProperty('listener', listener)
view.setSource(__file__.replace('.py', '.qml'))
view.showFullScreen()

app.exec_()
```

UnderMeSensi.qml

This one is really trivial: Have an enclosing rectangle (which fills the whole screen) and then an image centered into it that shows the PySide logo, gets scaled a bit (so that it fits the screen nicely) and finally has its rotation property set to the rotation property of listener (this is the key part here – it will update the image's rotation everytime the listener's rotation property changes).

```
import Qt 4.7

Rectangle {
    width: 800
    height: 480

    Image {
        source: "images/pysidelogo.png"
        fillMode: Image.PreserveAspectFit
        width: parent.width/2
        height: parent.height/2
        anchors.centerIn: parent
        rotation: listener.rotation
    }
}
```

How the example app looks like

Copy the files UnderMeSensi.py and UnderMeSensi.qml to your N900 and download the file logo.png [pyside.org] as images/pysidelogo.png (or use a custom image of your choosing and set the source:

path/URL in the QML file correctly. On an N900, it looks like this:

```
~/thp-qml-examples $ python UnderMeSensi.py
loaded the Generic plugin
loaded the N900 plugin
```

The example app in action: Video of the example on YouTube [youtu.be]