QWiimote - Nintendo Wiimote interactions for Qt / Symbian applications

This article explains how to interact with a Nintendo Wiimote on Symbian devices using QML.

Introduction

The Wii Remote (also known as the "Wiimote"), is the primary controller for Nintendo's Wii console. A main feature of the Wii Remote is its motion sensing capability, which allows the user to interact with and manipulate on-screen items on screen using gestures and pointing.

The Wiimote is expandable through the use of attachments like the "Nunchuk", which complement the Wii Remote by providing functions similar to those in gamepad controllers. Another attachment is the Wii MotionPlus which allows the Wii to more accurately capture complex motion (according to Nintendo, the sensor in the device supplements the accelerometer and Sensor Bar capabilities of the Wii Remote to enable actions to be rendered identically on the screen in real time).

Nintendo Wiimote

Many tools have been developed to connect the Wiimote to other devices (instead of the Wii console). Most are targeted towards desktop applications, and have complex structure and code interfaces. Little work has been done in the mobile space, and most of that has focused on substituting the device event management with the controller events.

The Nintendo Wiiremote project delivers a simple QML wrapper to connect to the Nintendo Wiimote on a Symbian device and to get its raw data and events. This article explains how to get and use the `QWiimote` API.

Adding the library to your projects

Get the source

Download the QWiimote source from here

.pro

Add the following line to your application project (.pro) file to include QWiimote project files:

```
include(wiimote/wiimote.pri)
```

main.cpp

Includes and register `QWiimote` so that it can be used in your application QML:

```
#include <QtGui/QApplication>
#include "qmlapplicationviewer.h"

#ifdef QWIIMOTE
#include <QtDeclarative>
#include "wiimote/qwiimote.h"
#endif

Q_DECL_EXPORT int main(int argc, char *argv[])
{
    QScopedPointer<QApplication> app(createApplication(argc, argv));

#ifdef QWIIMOTE
    // The following registers the C++ QWiimote to be used in QML as Wiimote
    qmlRegisterType<QWiimote>("Wiimote", 1, 0, "Wiimote");
    qmlRegisterType<QWiimoteAccelerometerData>("Wiimote", 1, 0, "WiimoteAccelerometerData");
    qmlRegisterType<QWiimoteMotionPlusData>("Wiimote", 1, 0, "WiimoteMotionPlusData");
#endif

    QmlApplicationViewer viewer;
    viewer.setMainQmlFile(QLatin1String("qml/WiimoteDemo/main.qml"));
    viewer.showExpanded();

    return app->exec();
```

 (C) Copyright Nokia 2013. All rights reserved.

```
}
```

QML

Declare the Wiimote QML element:

```
import QtQuick 1.1
import com.nokia.symbian 1.1
import Wiimote 1.0

Page {
     Wiimote {
        id: wiimote
     }
}
```

Connecting/ Device state management

Signals

- `onPowerOn` - Emitted when activating Bluetooth on your device
- `onDeviceSelected` - Emitted when selected an item/device from the Bluetooth system menu list. Returns device address
- `onConnected` - Emitted when connected to Wiimote
- `onDisconnected` - Emitted when disconnected from Wiimote

Slots

- `start()` - Activate Bluetooth on your device if needed and shows Bluetooth devices detected nearby. Is the first method to call in order to begin pairing operations.

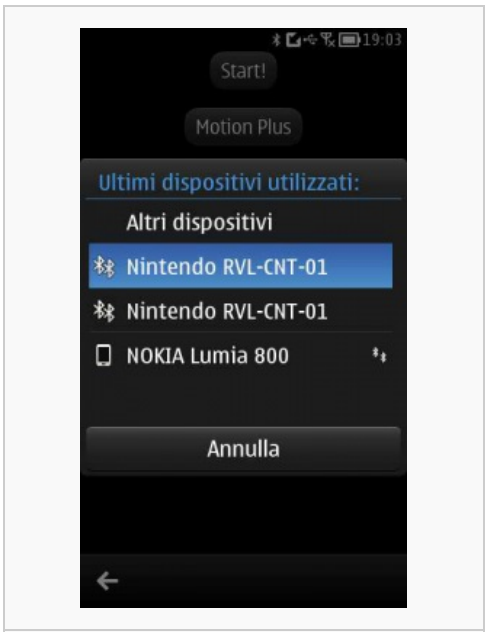
Code snippet

```
import QtQuick 1.1
import com.nokia.symbian 1.1
import Wiimote 1.0

Page {
    id: mainPage

    Button {
            text: "Start!"
            anchors.horizontalCenter: parent.horizontalCenter
            onClicked: wiimote.start();
     }

    Wiimote {
        id: wiimote

        onError : {
            console.log("Error code:"+error)
        }

        onPowerOn : {
            console.log("Bluetooth power error:"+error)
        }

        onDeviceSelected : {
            console.log("Selected device :"+address)
        }

        onConnected : {
            console.log("Connected to device")
            wiimote.setLed(Wiimote.Led1)
        }

        onDisconnected : {
            console.log("Disconnected from device")
            Qt.quit();
        }
    }
}
```

Pairing devices

To pair your device with Wiimote you need to call the `start()` slot. Press together the Wiimote's `1` and `2` buttons in order to make it discoverable.

 (C) Copyright Nokia 2013. All rights reserved.



System's menu for discovered Bluetooth devices

Wiimote pairing buttons

Selected the *Nintendo RVL-CNT-01* device the `onDeviceSelected()` and `onConnected()` signals are emitted.

Enabling actions

Once connected to device you can specify which kind of information you need.

- `setLed(WiimoteLed led)` - Sets the preferred Wiimote LED. Remember this operation needs to be made once the `onConnected()` signal is emitted

```
enum WiimoteLed {
       Rumble = 0x01,
       Led1   = 0x10,
       Led2   = 0x20,
       Led3   = 0x40,
       Led4   = 0x80
};
```

- `enableCoreButtons()` - Wiimote send just information on pressed buttons
- `enableCoreButtonsAndAccelerometer()` - Wiimote sends information on pressed buttons and accelerometer
- `enableMotionPlus()` - Enables the Motion Plus device
- `disableMotionPlus()` - Disable the motion Plus device
- `enableCoreButtonsAccelerometerExtention()` - Wiimote sends information on pressed button, accelerometer and gyroscope inside Motion Plus device

Once connected the Wiimote sets the `enableCoreButtons()` *action* automatically. If you need other information like accelerometer notifications you need to call `enableCoreButtonsAndAccelerometer()` *action*.

To access to Motion Plus data (the gyroscope) you need to call `enableCoreButtonsAccelerometerExtention()` *action*, but be careful to call the `enableMotionPlus()` *action* first.

Tip: Don't enable *actions* you don't need in order to save battery as Wiimote data flow is very high

Buttons

`QWiimote` provides a set of signals in order to manage buttons pressed on Wiimote.

Signals

- `buttonPressed(int button)` - Manages all buttons and useful for double pressing management as shown into overview
- `buttonLeft()`
- `buttonRight()`
- `buttonDown()`
- `buttonUp()`
- `buttonPlus()`
- `buttonTwo()`
- `buttonOne()`
- `buttonB()`
- `buttonA()`
- `buttonMinus()`

 (C) Copyright Nokia 2013. All rights reserved.

- `buttonHome()`

When a button in pressed on Wiimote two signals are emitted, `buttonPressed(int button)` and the corresponding button's signal. For example if left button is pressed `buttonPressed(int button)` and `buttonLeft()` signal are emitted.

```
enum WiimoteButton {
  ButtonLeft  = 0x0100,
  ButtonRight = 0x0200,
  ButtonDown  = 0x0400,
  ButtonUp    = 0x0800,
  ButtonPlus  = 0x1000,
  ButtonHome  = 0x0080,
  ButtonMinus = 0x0010,
  ButtonTwo   = 0x0001,
  ButtonOne   = 0x0002,
  ButtonB     = 0x0004,
  ButtonA     = 0x0008
};
```

Code snippet

```
import QtQuick 1.1
import com.nokia.symbian 1.1
import Wiimote 1.0

Page {
    id: mainPage

    Button {
            text: "Start!"
            anchors.horizontalCenter: parent.horizontalCenter
            onClicked: wiimote.start();
     }

     Button {
            text: "Activate Buttons"
            anchors.horizontalCenter: parent.horizontalCenter
            onClicked: wiimote.enableCoreButtons()
     }

    Wiimote {
        id: wiimote

        onButtonPressed : {
            console.log("pressed button: "+ button);
            if( button === (Wiimote.ButtonA|Wiimote.ButtonB))
                console.log("A + B pressed")
        }

        onButtonLeft : {
            console.log("pressed left button");
        }

        onButtonRight : {
            console.log("pressed right button");
        }

        onButtonUp : {
            console.log("pressed up button");
        }

        onButtonDown : {
            console.log("pressed down button");
        }

        onButtonHome : {
            console.log("pressed home button");
        }

        onButtonA : {
            console.log("pressed A button");
        }

        onButtonMinus : {
            console.log("pressed Minus button");
        }

}
```

Smooting raw data

Device sensor doesn't provide stable continuos data when device is still. This is because sensor are electrical devices bases on micro voltage changes that can be affected also by external conditions.
Is it possible to make up raw data using inner thresholds. If the new data has a value less then the fixed threshold then will not be updated.
That solution allows you to have stable data, but higher the threshold lower s the precision. If you don't feel with thresholds precalculated inside the project you can build your own data stabilizer using raw data.
To enable smooth data acquisition you have to use the `smooth` variable.

Code snippet

```
import QtQuick 1.1
import com.nokia.symbian 1.1
import Wiimote 1.0

Page {
    id: mainPage

    Wiimote {
        id: wiimote
        smooth:true
    }
```

 (C) Copyright Nokia 2013. All rights reserved.

```
}
```

Accelerometer

To use the accelerometer first call `enableCoreButtonsAndAccelerometer()` and then implement the `accelerometerChanged()` handler (`onAccelerometerChanged()`) to process the raw data from the Wiimote.
`accelerometerChanged()` returns a `QWiimoteAccelerometerData` object having the following member fields:

- `x` value of x-axis gravity
- `y` value of y-axis gravity
- `z` value of z-axis gravity

Code snippet

```
import QtQuick 1.1
import com.nokia.symbian 1.1
import Wiimote 1.0

Page {
    id: mainPage

    Button {
            text: "Start!"
            anchors.horizontalCenter: parent.horizontalCenter
            onClicked: wiimote.start();
     }

     Button {
            text: "Activate Buttons and Accelerometer"
            anchors.horizontalCenter: parent.horizontalCenter
            onClicked: wiimote.enableCoreButtonsAndAccelerometer()
     }

    Wiimote {
        id: wiimote
        smooth: true
        onAccelerometerChanged: {
            console.log(accelerometer.x)
            console.log(accelerometer.y)
            console.log(accelerometer.z)
        }
    }
}
```

Motion Plus

To use the gyroscope inside MotionPlus follow the following flow:

1. `enableMotionPlus()`
2. `enableCoreButtonsAccelerometerExtention()`
3. implement the `motionPlusChanged()` handler (`onMotionPlusChanged()`) to process the raw data from the Wiimote.

`motionPlusChanged()` returns a `QWiimoteMotionPlusData` object having the following member fields:

- `yaw` the yaw value
- `roll` the roll value
- `pitch` the pitch value
- `magnitude` the vector module of three axis. Useful when you need to catch the strength of the movement independent of direction
- `yawSlowMode` true if yaw movement is slow, false if very fast
- `rollSlowMode` true if roll movement is slow, false if very fast
- `pitchSlowMode` true if pitch movement is slow, false if very fast
- `extentionConnected` true if there is an extention controller connected to Motion Plus

Code snippet

```
import QtQuick 1.1
import com.nokia.symbian 1.1
import Wiimote 1.0

Page {
        id: mainPage

        Button {
            text: "Start!"
            anchors.horizontalCenter: parent.horizontalCenter
            onClicked: wiimote.start();
        }

        Button {
            text: "Motion Plus"
            anchors.horizontalCenter: parent.horizontalCenter
            onClicked: wiimote.enableMotionPlus()
        }

        Button {
            text: "Acceleration + Extension report"
            anchors.horizontalCenter: parent.horizontalCenter
            onClicked: wiimote.enableCoreButtonsAccelerometerExtention()
        }

        Wiimote {
```

 (C) Copyright Nokia 2013. All rights reserved.

```
                id: wiimote
               smooth: true

               onMotionPlusChanged: {
                     console.log(gyroscope.yaw)
                     console.log(gyroscope.roll)
                     console.log(gyroscope.pitch)
                     console.log(gyroscope.magnitude)
                     console.log(gyroscope.yawSlowMode)
                     console.log(gyroscope.rollSlowMode)
                     console.log(gyroscope.pitchSlowMode)
               }
         }
}
```

Battery level

In order to retrieve battery information, you can use the `battery` property

```
import QtQuick 1.1
import com.nokia.symbian 1.1
import Wiimote 1.0

Page {
     Wiimote {
        id: wiimote
        onConnected : {
            wiimote.setLed(Wiimote.Led1)
            console.log("Battery level"+wiimote.battery)
        }
     }
}
```

Further improvements

The project is still under development so while some features are already available (and almost frozen) others function are still coming.

In the near future I plan to release:

1. IR-Camera support

Summary

This article describes how to interact with Wiimote. For more information on hacking the remote, please refer to dedicated project's section : Wiimote Hacking

 (C) Copyright Nokia 2013. All rights reserved.

 (C) Copyright Nokia 2013. All rights reserved.