Porting Symbian Qt Apps to Nokia N9

This article explains how to port Symbian Qt applications to the Nokia N9, which is based on the MeeGo 1.2 Harmattan platform. Note also that most of the methods in this article **can also be used to port from Harmattan to Symbian**.

30 Oct 2011

Prerequisites

- Basics of mobile Qt development (Symbian or Harmattan)
- Familiarity with the Qt SDK

There are some differences between Symbian and MeeGo 1.2 Harmattan development processes. If you haven't yet developed for Harmattan we recommend you read Getting started with Harmattan (in *MeeGo 1.2 Harmattan Development Documentation*). This covers, among other things, selecting the correct target and setting up the Harmattan emulator (QEMU) or the device.

Introduction

Symbian and Harmattan both use the Qt as their primary app development framework. Although Qt is a "cross platform application and UI framework", there are differences between the platforms and the way that Qt apps may be written. Even if the apps do not use native code, these differences can impact the porting effort.

Some of the main issues to consider when porting are:

- Symbian applications may be written using pure Qt C++ with `QWidget` or `QGraphicsView` classes. Harmattan does not officially support `QWidget` based application development - the standard widgets are unattractive and there is no support for the Harmattan UI paradigms. Using QWidgets in Symbian has also been deprecated from Qt 4.7.4 release. Although QWidgets work as earlier it is highly recommended to not to use them anymore, see Symbian platform notes.
- Symbian apps written using *Qt Quick* may use *Qt Quick Components for Symbian* for a native look and feel. Harmattan developers use the Harmatttan-specific Qt Quick Components for the platform look and feel. These may not run or behave the same on Harmattan, and there are some components that exist in only one of the plaforms or the other. However, many of the incompatibilities will be fixed in the future releases. For a comparison between the Symbian and Harmattan components, see *UI migration from N9 to updated Symbian style*.
- The Nokia N9 (Harmattan) has a higher resolution than comparable Symbian devices (854 x 480 vs. 640 x 360 [640 x 480 in Nokia E6-00]). Symbian apps that don't follow the scalability guidelines may run into scalability issues.
- Symbian C++ native code can require a lot of porting effort, and may need to be re-written.
- Symbian applications that use Qt Multimedia should work on Harmattan. If there is a requirement for low latency audio the code may need to be ported to Harmattan's native Pulse Audio framework.

Given these issues, there are a number of porting approaches you can use. These are outlined, and expanded upon, in the following sections.

Porting methods

In general, regardless of the type of application, there are three approaches to porting:

1. Rewrite the application.
2. Make the necessary changes to the application code base to make it fully cross-platform (so the same code base works on both Symbian and Harmattan).
3. Some combination of the two previous approaches: extract the cross-platform parts of the application and rewrite the platform-specific parts to Harmattan.

Rewriting the application (engine and/or UI) is advisable if the application contains a lot of code that is impossible or expensive to port. This applies to applications either fully or mostly written in Symbian C++. It may also apply to `QWidget`-based UIs, as these would be easier to reimplement in Qt Quick than to rewrite for Harmattan. We don't discuss this approach any further in this article, because with Qt Quick a complete rewrite of the application is rarely required.

The second approach, making the application fully cross-platform, may be feasible if the application's user interface already compensates for some of the differences between platforms - for example, a UI that is already scalable, or that doesn't contain platform-specific components. A Qt application that only uses Qt Quick but not Qt Quick Components falls to this category. This approach is also often suitable for OpenGL ES games and applications, although some tweaks may be required. Nokia Developer examples ported using this approach include:

- Bubble Level (Qt Quick application, no Qt Quick Components)]
- Compass (Qt Quick application, no Qt Quick Components)]
- Guitar Tuner (Qt Quick application, no Qt Quick Components)]
- Qoat of the Hill (OpenGL ES game)]

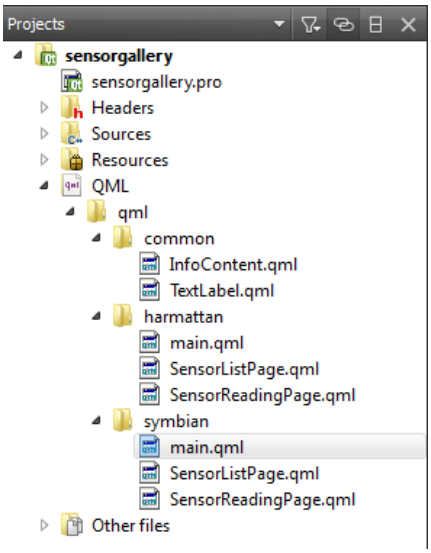
The third approach, extracting the cross-platform parts of the application and rewriting the platform specific parts to Harmattan, is often the best approach. This is especially effective for Qt Quick Components based apps, because so much of the UI can be shared. We cover this approach step-by-step in the following section.

Note: It can be difficult to choose between the second and third approach. While there are maintenance benefits in having a single code-base, it can often be easier (and result in a better UI experience) to rewrite the platform-specific code. If in doubt, we recommend the third approach. For more discussion and advice, please read this Qt Quick components blog article by Attila Csipa.

Porting step-by-step

Step 1: Refactoring project structure

First, create separate folders for common, Symbian-specific, and Harmattan-specific UI components, as shown in the following figure. Note that the structure may not be displayed in the Active projects window of the Qt SDK until the project file modifications in step 2 are completed.



Next identify and extract common components (the files whose code fully works or can easily be made to work on both platforms) and move them into the 'common' folder. Here's an example a custom `TextLabel` element that works on both platforms:

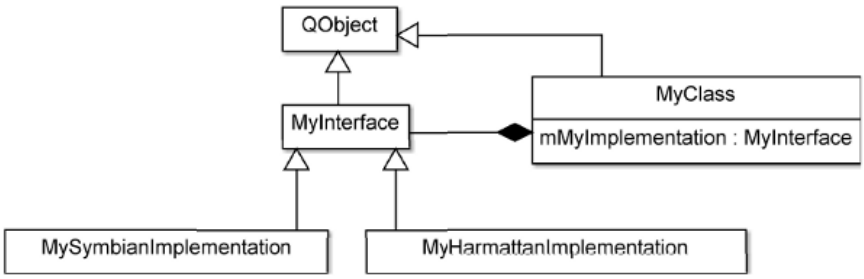
```
import QtQuick 1.0

Rectangle {
    property alias text: labelText.text
    ...

    gradient: Gradient {
        GradientStop { position: 0.0; color: "#555555" }
        GradientStop { position: 1.0; color: "#222222" }
    }

    // Inner rectangle to make borders
    Rectangle {
        ...
    }

    Text {
        id: labelText
        ...
    }
}
```

Common elements must be composed only from other common elements. For scalability, use relative positioning and sizes and let the parent element (which can be a platform-specific element) define the size of the child (note that you can still define the *default* size for your common elements). Finally, using anchors and layout elements like Grid, Row, and Column, make it easy to create scalable application views.

After the common parts have been extracted, the Symbian-specific QML and Javascript code can be moved to the 'symbian' folder.

Copy the files from the 'symbian' folder to the 'harmattan' folder. This will not provide a working Harmattan UI (unless you're really lucky or talented) but this is enough for now; we'll get to the required modifications in the later steps.

The application engine structure (implemented with Qt/C++) doesn't usually require changes. Platform-specific variations can usually be handled using `#ifdef` blocks. Should an implementation of a class be completely different for Harmattan, utilising interface classes and dynamic binding usually solves the issue:




```
...

#include <QObject>
#include "myinterface.h"

class MyClass : public QObject
{
    Q_OBJECT

public:
    explicit MyClass(QObject *parent = 0);
```

```
    ...

private:
    MyInterface *mMyImplementation;
};

...
```

In the header file declare a member variable pointer to a type of `MyInterface` in `MyClass` .

```
...

// Include the correct header depending on the target platform.
#ifdef Q_OS_SYMBIAN
    #include "mysymbianimplementation.h"
#else
    #include "myharmattanimplementation.h"
#endif

...

/*!
  Constructor.
*/
MyClass::MyClass(QObject *parent)
    : QObject(parent),
      mMyImplementation(0)
{
#ifdef Q_OS_SYMBIAN
    mMyImplementation = new MySymbianImplementation(this);
#else
    mMyImplementation = new MyHarmattanImplementation(this);
#endif
}

...
```

In the source file (.cpp) construct a different instance depending on the target platform.

For information on how to define a Harmattan-specific code scope, see here.

Step 2: Project file and application deployment

Add the Harmattan configurations into the project file. The following block contains the configurations in the project file of the RentBook example (some parts omitted):

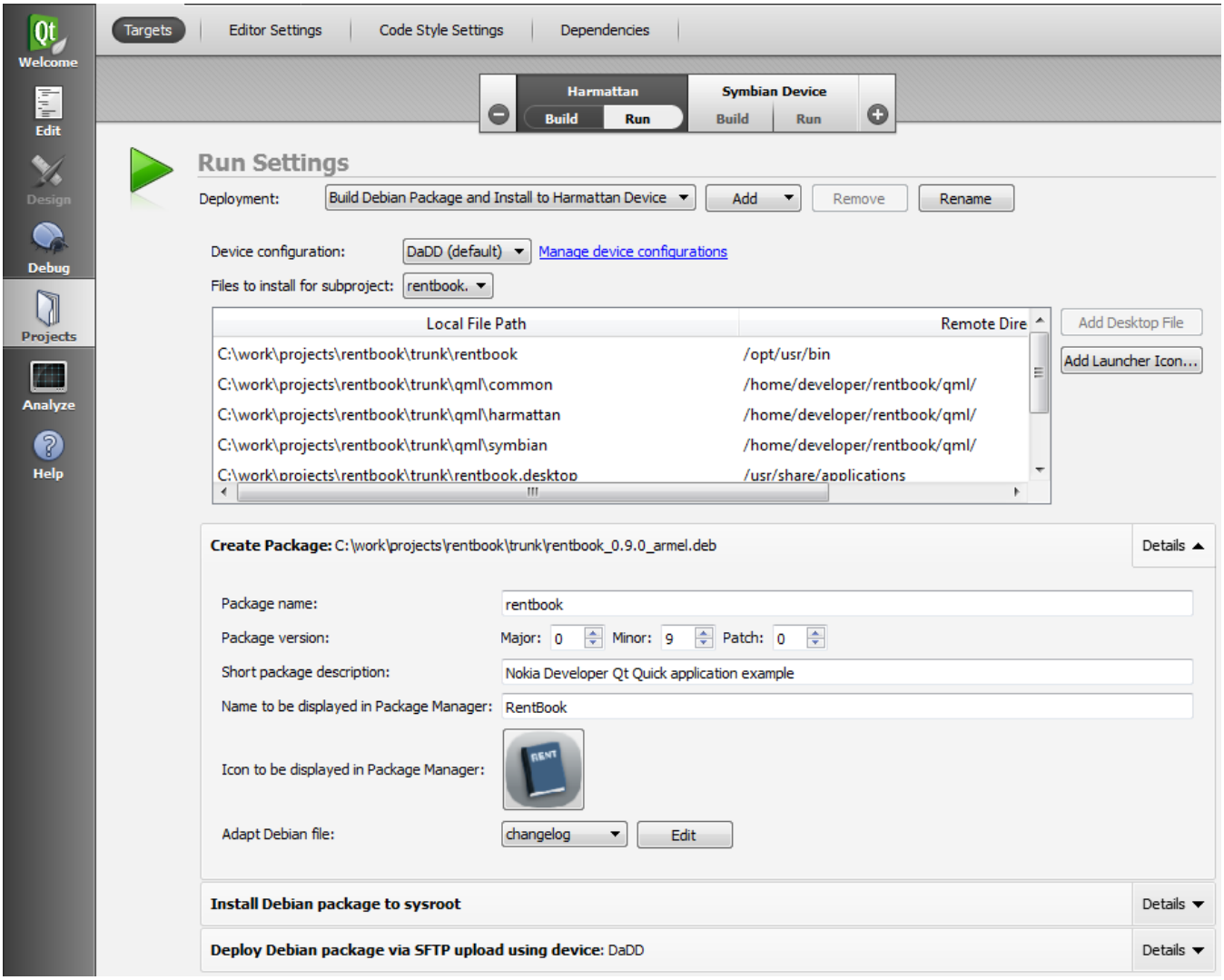
```
QT += core gui sql declarative
CONFIG += qt-components
TARGET = rentbook
TEMPLATE = app

SOURCES += \
    src/main.cpp \
    src/DatabaseManager.cpp \
    src/telephony.cpp

HEADERS += \
    src/DatabaseManager.h \
    src/telephony.h

# Symbian specific
symbian {
    message(Symbian build)
    TARGET = RentBook
    TARGET.UID3 = 0xea6c2793
    TARGET.CAPABILITY += NetworkServices
    TARGET.EPOCSTACKSIZE = 0x14000
    TARGET.EPOCHEAPSIZE = 0x1000 0x1800000 # 24MB
    ICON = icons/rentbook.svg

    RESOURCES += resources.qrc

    HEADERS += src/telephony_symbian.h
    SOURCES += src/telephony_symbian.cpp

    LIBS += -letel3rdparty

    OTHER_FILES += qml/symbian/*.qml

    qmlfiles.sources = qml
    DEPLOYMENT += qmlfiles
}

# Harmattan specific
contains(MEEGO_EDITION, harmattan) {
    message(Harmattan build)

    HEADERS += src/telephony_stub.h
    SOURCES += src/telephony_stub.cpp

    target.path = /opt/usr/bin
    INSTALLS += target

    qmlfiles.path = /home/developer/rentbook/qml/
    qmlfiles.files += qml/*
    INSTALLS += qmlfiles

    desktopfile.files = rentbook.desktop
    desktopfile.path = /usr/share/applications
    icon.files = icons/rentbook.png
    icon.path = /usr/share/icons/hicolor/64x64/apps
    INSTALLS += desktopfile icon

    OTHER_FILES += qml/harmattan/*.qml
}
```

The Debian packaging files for Harmattan are created by the Qt SDK. The desktop file is required to display the application launcher icon in the application menu on the device. The desktop file contains such information as the application title (shown in the application menu), the file name of the icon (without `.png` suffix), and the path of the executable. The application launcher icon for Harmattan must be in PNG format and its size must be 80 x 80 pixels.

```
[Desktop Entry]
Encoding=UTF-8
Version=1.0
Type=Application
Terminal=false
Name=RentBook
Exec=/opt/usr/bin/rentbook
Icon=rentbook
X-Window-Icon=
X-HildonDesk-ShowInToolbar=true
X-Osso-Type=application/x-executable
```

The desktop file content of RentBook example application.

The deployment options for the project can be set in the Projects tab page in the Qt SDK (see the following figure).



Harmattan platform uses a concept called Resource Policy Framework to handle the different roles and modes that a modern smartphone is used in. For example, you can make the hardware volume keys control the audio volume of your game application. For more information, see Selecting the resource application class.

Example:

1. Create a `<your application>.conf` file (the following snippets are from the AirSwype example).

```
[classify gaming]
/opt/usr/bin/airswype/airswype
```

2. Add the following snippet into the Harmattan-specific scope in the project file. This will deploy the `.conf` file with the application.

```
# Classify the application as a game to support volume keys on Harmattan.
gameclassify.files += qtc_packaging/debian_harmattan/airswype.conf
gameclassify.path = /usr/share/policy/etc/syspart.conf.d
INSTALLS += gameclassify
```

Step 3: First build

Before trying to build for the first time, remember to modify `main.cpp` so that the correct QML file is loaded when the application is launched (note that this is only required if the QML files are deployed onto the device instead of putting them into resources):

```
#if defined(Q_OS_SYMBIAN) || defined(Q_WS_SIMULATOR)
    // Symbian and Simulator
    view.setSource(QUrl::fromLocalFile("qml/symbian/main.qml"));
#else
    // Harmattan
    view.setSource(QUrl::fromLocalFile("qml/harmattan/main.qml"));
#endif
```

QML and JavaScript files can be compiled into the binary using the Qt resource system. When using the resource system, you don't have to deploy (copy) the QML files onto the device. However, due to a bug in Qt Quick 1.0 concerning Symbian QML files using the component, icons cannot be placed in resources. The bug is fixed in Qt Quick 1.1 release for Symbian that is available in Qt 4.7.4.

If the main QML file is loaded from resources and you have created separate resource files for both platforms, no changes are required:

```
view.setSource(QUrl("qrc:/<path>/main.qml"));
```

Build the application using the Harmattan target and fix any errors found. Build errors help you to find the platform-specific code that needs to be rewritten for Harmattan. The `#ifdef` approach usually works.

Step 4: Adapting to Harmattan Qt Quick Components

If your Symbian Qt Quick application uses Qt Quick Components 1.0 and components such as PageStack, StatusBar, or ToolBar, you probably need to modify at least the `main.qml` file of the Harmattan build. The following snippets show the possible differences in `main.qml` between Symbian and Harmattan:

```
import QtQuick 1.0
import com.nokia.symbian 1.0 // Symbian Qt Quick Components

Window {
    id: root

    property int orientation: PageOrientation.Automatic

    SensorListPage {
        id: sensorListPage
        orientationLock: root.orientation
    }

    // Common application statusbar
    StatusBar {
        id: statusBar
        anchors.top: root.top
        z: 1
    }

    // Page stack for all pages
    PageStack {
        id: pageStack

        toolBar: toolBar

        anchors {
            left: parent.left
            right: parent.right
            top: statusBar.bottom
            bottom: toolBar.top
        }

        Component.onCompleted: {
            // Push the first page on the stack.
            pageStack.push(sensorListPage);
        }
    }

    // Common toolbar for the pages
    ToolBar {
         id: toolBar
         anchors.bottom: parent.bottom
    }
}
```

main.qml for Symbian build.

```
import QtQuick 1.1
import com.nokia.meego 1.0 // Harmattan Qt Quick components

PageStackWindow {
    id: root

    property int orientation: PageOrientation.Automatic

    showStatusBar: true
    showToolBar: true

    initialPage: SensorListPage {}

    Component.onCompleted: {
        // Use the dark theme.
        theme.inverted = true;
    }
}
```

Note that with Qt Quick 1.1 and Qt Quick Components 1.1 for Symbian, Symbian components will also be equipped with the PageStackWindow element.

Next, run/debug your application with an emulator or on a device and locate the possible errors in the debug log. Also fix possible scaling issues since the resolution of Nokia N9 is significantly higher than in Symbian devices (854 x 480 vs. 640 x 360 [except 640 x 480 in Nokia E6-00]).

Step 5: Testing and fixing

No special tips or tricks exist for this step. Just keep testing and improving your new Harmattan version until your application looks and behaves the way you want it to. Don't forget to test that your original Symbian version still works!

If your application uses Qt Quick Components, note that the component set does not match perfectly. For example, Qt Quick Components 1.0 for Symbian lack the PageStackWindow element and Harmattan components version 1.0 lack the ListItem element which you need to implement yourself when porting an application (see the following snippets). Moving to Qt Quick Components 1.1 for Symbian usage will ease the porting to Harmattan as there's less differences in fundamental elements like PageStackWindow. Generally, applications will get other benefits too like the split-view inputs which is similar how text input happens in Harmattan.

```
Component {
    id: listDelegate

    ListItem {
        id: listItem

        ListItemText {
            x: platformStyle.paddingLarge
            anchors.verticalCenter: listItem.verticalCenter
            mode: listItem.mode
            role: "Title"
            text: name
        }

        subItemIndicator: true

        onClicked: {
            showRendDetails();
        }

        onPressAndHold : {
            contextMenu.open();
        }
    }
}
```

A list item delegate implemented with the ListItem element on Symbian.

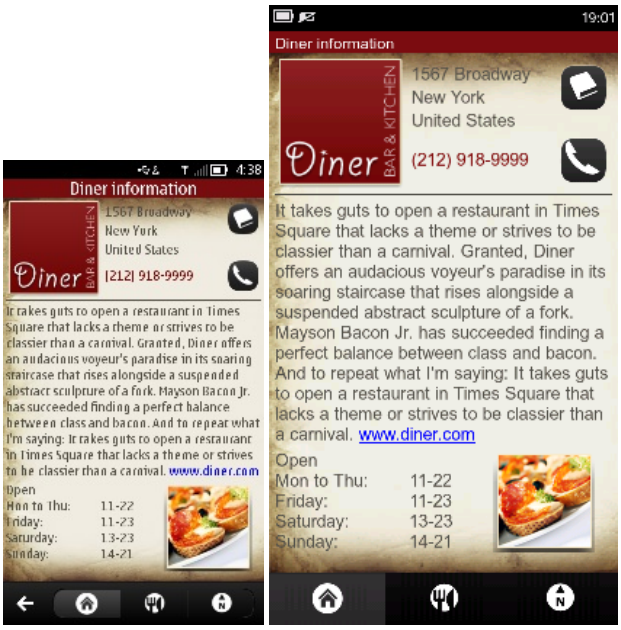
```
Component {
    id: listDelegate

    Item {
        id: listItem
        height: UIConstants.LIST_ITEM_HEIGHT_SMALL
        width: listView.width

        Rectangle {
            radius: 8
            anchors.fill: parent
            opacity: 0.7
            color: "#1874CD"
            visible: itemMouseArea.pressed
        }
        Row {
            x: 10
            width: parent.width - 20
            height: parent.height

            Text {
                width: listItem.width - arrowImage.width - 20
                height: listItem.height
                verticalAlignment: Text.AlignVCenter
                font { family: platformLabelStyle.fontFamily; pixelSize: platformLabelStyle.fontPixelSize }
                color: platformLabelStyle.textColor
                text: name
            }
            Image {
                id: arrowImage
                y: (listItem.height - sourceSize.height) / 2
                source: "image://theme/icon-m-common-drilldown-arrow"
                        + (theme.inverted ? "-inverse" : "");
            }
        }

        MouseArea {
            id: itemMouseArea
            anchors.fill: parent

            onClicked: {
                listView.currentIndex = index;
                showRendDetails();
            }
            onPressAndHold : {
                listView.currentIndex = index;
                contextMenu.open();
            }
        }
    }
}
```

The same list item delegate implemented on Harmattan.

Examples

Nokia Developer Qt examples, that are ported using the approach covered in this section:

- Diner
- RentBook example
- RSS Reader
- Tic-Tac-Toe over Sockets (original ported from Windows Phone to Qt Quick)



Screenshots of the Diner example running on Symbian and Harmattan

OpenGL ES apps

OpenGL ES games and applications usually require few changes. However, the setup around the native OpenGL ES code has to be changed. Fortunately, two set of APIs can help you port your application:

- Qt GameEnabler (also provides an audio framework for both Symbian and Harmattan platforms)
- Games API

Qoat of the Hill example demonstrates the use of both the above APIs. Match'em Poker example is another OpenGL ES game which has been ported from iOS to Qt devices with Qt GameEnabler.

Summary

The approach for porting a Qt application depends on the type of the application and, ultimately, on the differences between the platforms. The work required may be merely changing two or three lines of code, or it could be significantly more. Fortunately, a complete rewrite is rarely required. When porting Qt Quick applications, especially those implemented with Qt Quick components, keep in mind that the component set does not match perfectly on both platforms and some of the components may behave differently. In addition, try to make your QML UI elements scalable since the resolution is higher on the Nokia N9.

Additional information

- Harmattan:Platform Guide
- MeeGo 1.2 Harmattan Developer Documentation: Porting applications to Harmattan
- MeeGo 1.2 Harmattan code examples