Tapio Andersson

PORTABILITY OF QT APPLICATIONS IN THE MOBILE ENVIRONMENT

Information Technology
Software engineering
2009

PORTABILITY OF QT APPLICATIONS IN THE MOBILE ENVIRONMENT

Andersson, Tapio
Satakunta University of Applied Sciences
Degree Programme in Information Technology
Specialization Option in Software Engineering
Commissioned by Digia Plc.
Supervisor: Antti Vainio, M.Sc
May 2009
Tutor: Ismo Trast, Tech.Lic. , Principal Lecturer
UDC: 004.057.5, 004.41
Number of Pages: 36

Keywords: portability, mobile environment, Maemo, S60, Qt

The purpose of this Bachelor's thesis was to get more experience about the portability of Qt applications in the mobile environment. Qt is a cross-platform application framework.

Software development in the mobile environment is diffused to many different platforms. The aim of this thesis was to find answers how portable Qt is. The study was made through analysis of different aspects: platform independence, code maintenance, performance and testability.

The theory part of this thesis covers software development in mobile environments, Qt as a tool and portability between Maemo and S60. The applied part lists findings in those four aspects listed at the end of the previous chapter.

In the near future Qt will be a very interesting option for mobile software development. At its current state it is not ready for commercial products.

QT SOVELLUKSIEN PORTATTAVUUS MOBIILIYMPÄRISTÖSSÄ

Andersson, Tapio
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautumisvaihtoehto
Yritys: Digia Oyj.
Valvoja: DI Antti Vainio
Toukokuu 2009
Ohjaaja: Yliopettaja, TkL Ismo Trast
UDK: 004.057.5, 004.41
Sivumäärä: 36

Asiasanat: portattavuus, mobiiliympäristö, Maemo, S60, Qt

Tämän opinnäytetyön tarkoituksena oli hankkia lisää kokemusta Qt sovellusten portattavuudesta mobiiliympäristöissä. Qt on järjestelmäriippumaton sovelluskehys.

Ohjelmistokehitys mobiiliympäristöissä on hajaantunut monelle eri alustalle. Tämän opinnäytetyön tarkoituksena oli etsiä vastauksia, kuinka portattava Qt on. Tutkimus on tehty analysoimalla aihetta seuraavien näkökulmien kautta: järjestelmäriippumattomuus, ohjelmakoodin ylläpidettävyys, suorituskyky sekä testattavuus.

Teoriaosa opinnäytetyöstä käsittelee ohjelmistokehitystä mobiiliympäristöissä, Qt:ta työkaluna ja portattavuutta Maemo ja S60 alustojen välillä. Käytännönosa listaa löydökset niistä neljästä näkökohdasta jotka mainittiin edellisen kappaleen lopussa.

Lähitulevaisuudessa Qt on hyvin mielenkiintoinen vaihtoehto mobiili-ohjelmistojen kehittämiseen. Se ei ole nykyisessään tilassa valmis kaupallisiin tuotteisiin.

CONTENTS

Android	A new mobile platform provided by Google
API	Application Programming Interface
Cross-platform	Platform independent that works on every supported OS
DEB	Debian installation package
DLL	Dynamic Link Library
GTK	The GIMP Toolkit
GUI	Graphical User Interface
Framework	Reusable abstractions of code wrapped in an API
IDE	Integrated Development Environment
J2ME	Java 2 Micro Edition
KDE	K Desktop Environment
LAN	Local Area Network
LGPL	Lesser GNU Public License
Linux	Linux is UNIX like operating system
Maemo	Lightweight, Linux based operating system
MOC	Meta-Object Compiler
OpenC/C++	Application development environment for S60
OpenGL	Open Graphics Library
OS	Operating System
PDA	Portable Digital Assistant
P.I.P.S	PIPS is POSIX on Symbian
Plc.	Public limited company
Qt	Cross-platform application framework
SIS	Symbian Installation Source, Symbian installation package
SMS	Short Message Service
SSH	Secure Shell
TBA	To Be Announced
WLAN	Wireless LAN
X11	X Window System, Version 11

1 INTRODUCTION

Software development in mobile environment is diffused to many different platforms. Qt is one solution for this multi-platform problem. Qt is a cross-platform C++ application framework. This means that it is possible to implement an application by using Qt and use the same source code on another platform with only some minor platform independent settings or code changes. So it should make software development faster and more cost efficient too.

The purpose of this study is to get more information about portability of the Qt applications between S60 and Maemo platforms. This thesis focuses are finding how platform independent Qt software should be designed, benefits, disadvantages and problems of using Qt in the mobile environment. This study is made through analysis with different aspects: platform independence, code maintenance, performance and testability.

This thesis is prepared for Digia Plc. Digia is a company which is established in Finland but it also has field offices in China, Estonia, Russia and Sweden. Company delivers IT-solutions to it's customers. Digia employs over 1300 professionals globally. Main business areas of the company are information and communication technologies, smartphones and real-time systems.

There is not much written information available about using Qt in mobile environment. The reason for choosing this topic for the thesis is that Digia needs more practical information about Qt. Since Nokia bought Trolltech in 2008 Qt become even more interesting technology in mobile software development. This is also the reason why Qt is very topical subject and Digia is interested in it.

During writing this thesis S60 version of the Qt was in unfinished state. Some of the Qt modules were not ported yet for S60 platform. This caused some problems on S60 platform, but those problems should be fixed in next Qt releases.

2 SOFTWARE DEVELOPMENT IN MOBILE ENVIRONMENTS

2.1 Software development before Qt

In the long run smart phone business is diffused between many different operating systems. The applications need to be separately implemented to different platforms. On the Q4 2008 sales Symbian OS (Operating System) was the leading smart phone operating system that had 47,1% of the smart phone markets, with RIM (Research In Motion) had 19,5%, Windows Mobile (Windows CE) had 12,4% and Apple (iPhone OS) had 10,7%. Leaving 10,3% of the shares goes between Linux, Palm, BREW and Android. [1]

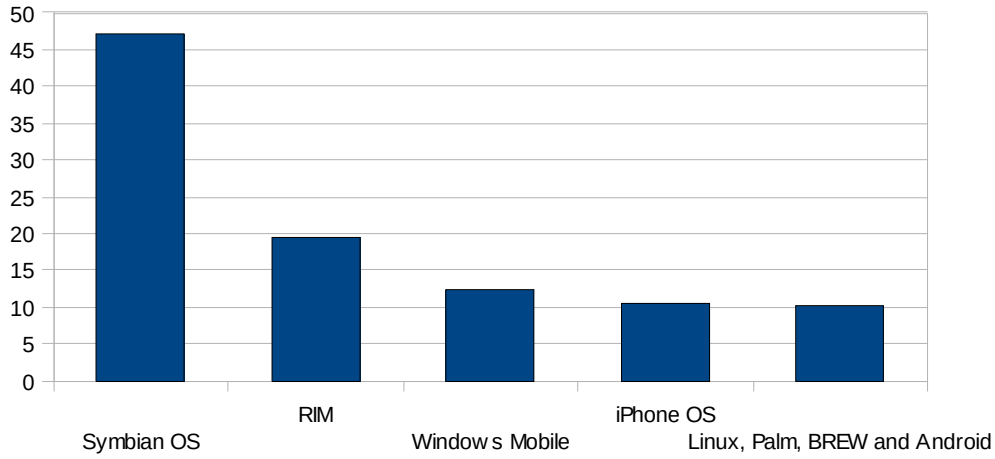

Figure 1. Market shares Q4 2008 [1]

There have been some solutions for the multi-platform problem earlier, but none of those solutions has been very successful. Mobile version of Java, J2ME (Java 2 Micro Edition), had lacks for using it with different types of devices. It was not as adaptive as mobile phone industry wanted it to be.

One big lack in mobile environment is usability. GUI:s (Graphical User Interface) in the phones have looked similar for a long time. S60 platform started to look outdated after Apple released its iPhone OS. Other platforms like RIM and Windows Mobile looked also more modern than S60. So there became a need to

make improvements for S60 GUI and to seriously challenge iPhone's touch UI. Qt may also become a solution for better looking and easy-to-use GUI:s.

2.2 S60 platform before Qt

S60 is software platform that runs on Symbian OS. Symbian OS, originally named as EPOC, was originally designed in the early 90's for PDA:s' (Portable Digital Assistant) operating system. In the year 2000 name EPOC was renamed to Symbian OS. Nokia released the phone model 9210. It was the first Symbian phone, in which it was possible to install custom made 3rd party Symbian applications. [2] [3]

Because Symbian OS has so long history some features of it are made for earlier devices purposes. This causes also some compatibility problems between old and new devices. Writing code on S60 is not always easy or efficient. If we make a simple GUI based dialog application, we have to create four classes: Application, Document, appUI and appView. If we want to show some text in that dialog we have to manually allocate memory for the text. We also have to take care of de-allocating of the memory manually.

In the Qt world all this can be done in a straight-forward manner in one file. First we make a main application and add a QWidget object. We can add a QLabel widget and put the text on it with the setText() function. Same kind of application as formerly mentioned is done much easier and faster with Qt than native S60 code. Allocation and de-allocation of memory is done automatically by Qt. This reduces the number of human errors.

2.3 Maemo platform before Qt

Maemo is a lightweight operating system, which is created and developed by Nokia. It is based on modified Debian GNU/Linux. In the Linux world there has been a problem, that there is too many different programs can be used, multiple programming components and many different versions of them. There has been a

need for something stable, standard and easy to use. Qt is the answer for this kind of problem. After Nokia bought Trolltech they could develop Qt in the way they need and want.

There are no released Maemo devices yet that are using Qt. Nokia N810 Internet tablet is currently using Hildon application framework. Hildon is based on GTK (The GIMP Toolkit). It is possible to install Qt 4.5.0 package from Maemo repository to N810.

When Nokia released Qt for S60 in March, 2009, Qt licensing was changed. It was released under LGPL license (Lesser GNU Public License). Furthermore, it has commercial license. Now it is possible to make commercial software with Qt and use it for free. That makes Qt more interesting for both software companies and open source community. Open source community did not show much interest on Qt before because it was released under GPL license. [2] [13]

2.4 Brief Qt history

The story of Qt began in Norway on 1990. Haavard Nord and Eirik Chambe-Eng started to design an object-oriented C++ application which needed to be able to run on Macintosh, UNIX and Windows. In 1991 they started to design and write classes. In the next year Eirik got an idea for “signal and slot” based system, which was a simple but efficient GUI programming paradigm. That kind of system is now copied to many other toolkits. Haavard and Eirik decided to go into business and to build “the world's best C++ GUI framework”. [4]

In 1994 Qt got its name when its inventors took Q from Emacs font because it looked so “cute” and the letter T from word toolkit. On 4th of March 1994, a company named Quasar Technologies was established. Later it changed its name to Troll Tech, and after that to Trolltech. First years were challenging for the company. The company had no customers and Qt was in unfinished state. [4]

In April, 1995, Trolltech got its first customer and Qt was also released with a commercial and open source license. Next notable step in Qt's history took place in 1996 when Matthias Ettrich decided to build KDE (K Desktop Environment) and chose Qt as a base for the project. That helped Qt to become a standard for C++ GUI development on Linux. Qt got also version number 1.0 in this year. [4]

In August 1999 Qt won LinuxWorld's award in the category of best library/tool. In the next year Trolltech released Qt/Embedded Linux. It had an own lightweight window system as a replacement for the X11 (X Window System, Version 11). Licensing changed also to commonly used GPL for open source usage. By the end of the year Trolltech released Qtopia application platform for mobile phone and PDA usage. Qt/Embedded Linux won the LinuxWorld “Best Embedded Linux Solution” in 2001 and 2002. Trolltech's Qtopia Phone won the same title in 2004. [4]

In the summer 2005 Qt 4.0 version was released. It contained 500 classes and more than 9000 functions. It has been splitted to many libraries so that a user needs to link binaries only against the parts that are really needed. The fourth version of Qt had also many improvements like template containers, advanced model/view functionality, fast and flexible 2D painting framework and Unicode classes.

Now Trolltech is known as Qt Software after Nokia bought it in June 2008. Nokia bought also Symbian in June 2008. [4]

2.5 Structure of Qt

The idea in Qt 4 is that it is built from modules. Binaries must be linked against only those modules which are needed. Modules and current working states are listed in following Figure 2:

Module	**Description**	**Works currently with**
QtCore	Non-graphical classes used by other modules	Maemo and S60 (Not fully)
QtGui	GUI programming classes	Maemo and S60 (Not fully)
QtNetwork	Network programming classes	Maemo and S60 (Not fully)

QtOpenGL	QpenGL classes	Maemo (No capable devices)
QtScript	Qt Script classes	Maemo and S60
QtSql	Database classes	Maemo
QtSvg	SVG image file classes	Maemo and S60
QtWebKit	Classes for rendering and editing Web content	Maemo
QtXml	XML classes	Maemo and S60
QtXmlPatterns	XQuery & XPath classes for XML operations	Maemo
Phonon	Multimedia classes (from KDE project)	TBA (for Maemo and S60)
Qt3Support	Compatibility classes for previous Qt3	Maemo

Figure 2. Table which shows different make options with S60

Main modules of the Qt are shown in picture:

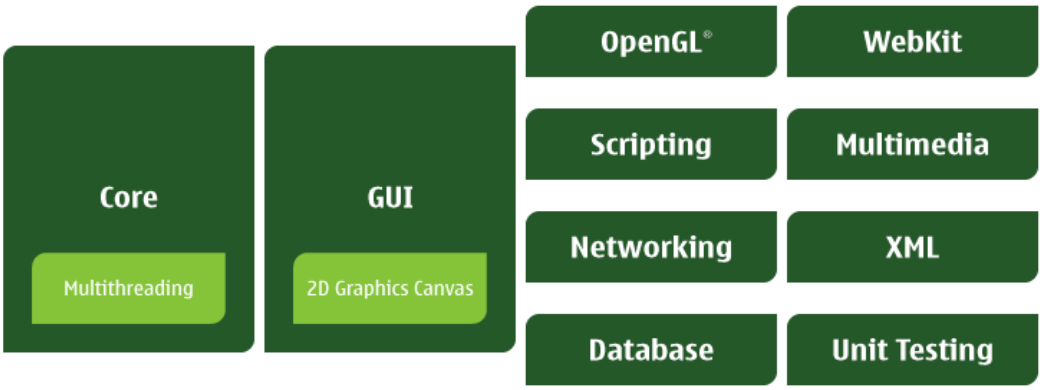

Figure 3. A picture of Qt's main modules. [10]

Of those modules Open GL is implemented for Maemo but not for S60 devices. Phonon module's implementation is still in progress and it’s not fully implemented for Maemo or S60 yet. Phonon is a part of KDE project and will be take care of Qt's Multimedia routing and playback. All Qt modules offer easy-to-use functionalities for different actions. [10]

3 CROSS-PLATFORM SOFTWARE

3.1 Cross-platform software and portability in theory

Cross-platform term means that some application is platform independent. Cross-platform software is usually deployed in a platform specific binary or in a platform independent source code packages. In Qt's case term cross-platform means that Qt framework must be ported natively to every platform so that it works. This operation is done by Qt Software. After some platform has its own port of Qt then applications can be compiled against it. In the ideal case one only needs to run commands: **qmake** and **make**. Native Qt application framework takes care of all platform specific things. Qmake builds platform specific build files like Makefile in Maemo or abld.bat, bld.inf, package and resource files in S60.

Cross-platform software is wise way to reduce costs of developing software. It makes also possible to create software for multiple devices at the same time. From usability perspective platform independent software is also good thing for people who use it. All applications work the same way on every device.

3.2 Open C/C++

Open C/C++ is a development environment for Symbian devices. All Symbian OS versions do not have native support for all of the C- or C++- features. Because of that S60 3rd Edition Feature Pack 1 needs P.I.P.S plug-in (P.I.P.S. Is POSIX on Symbian) for SDK and devices. S60 3rd Edition Feature Pack 2 and S60 5th Edition have Open C/C++ pre-installed.

Open C/C++ works in a layer between native Symbian and Qt libraries. Open C/C++ takes care of initializing the main() function in S60 side. Qt uses private implementation for platform specific classes and that makes porting possible.

Open C/C++ can also be used in middleware components or porting existing desktop applications to S60.

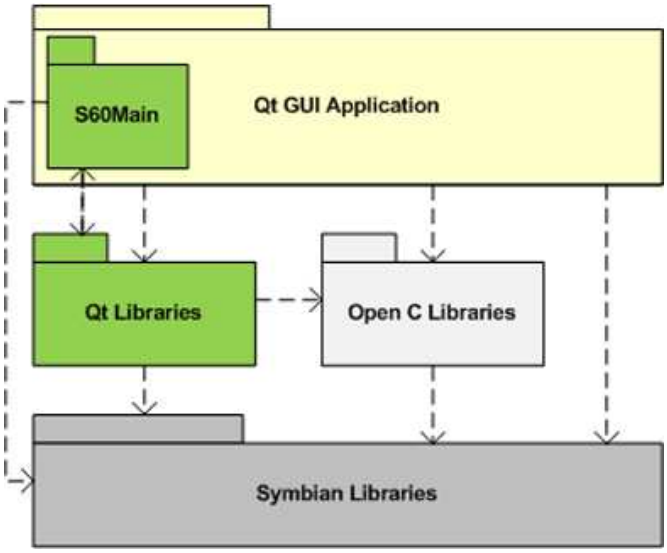

Figure 4. Figure about how Open C works between S60 and Qt. [14]

4 QT AS A TOOL

4.1 Signals and Slots

Qt has a tool named **moc** (Meta-Object Compiler). Meta-Object Compiler reads C++ header files. If it finds a class declaration that contains the Q_OBJECT macro, it produces C++ source code containing the meta-object code for those classes. This Q_OBJECT macro is possible to add in a class which is inherited from the QObject class. [20]

Qt uses signals and slots for communication between objects. Macro SIGNAL defines a signal that is going to be emitted. Macro SLOT defines a function which receives emitted signal. MOC generates automatically loose coupling connection used between signals and slots. Multiple signals could be connected to multiple slots. [19][20]

It is possible to make subclasses from Qt's widgets and add own slots to they. Multiple signals could be connected to a single slot and one signal could be

connected to multiple slots. It is also possible to connect some signal to another signal which triggers some slot function.

According to personal experience, signals and slots have one problem. It is possible to implement application which goes though compiler and still does not work. If signal or slot gets renamed or is named badly then connect() function does not work correctly with Meta-Object Compiler. It may take long time to find this kind of problem because compiler does not give any errors.

The signals and slots is type safe mechanism. Signals and slots have signatures and those need to match between used signals and slots for that implementation works. There is loosely coupling between signals and slots. This means that emitted signal does not need to know about slot which receives it. It is possible to use slots for receiving signals and as normal member functions. See more in Annex 1.

4.2 Qt's tools

Qt offers extensive set of tools for developing software. Qt Assistant is a tool which works as Qt's reference documentation. It contains code snippets and a lot of helpful information how to use framework's classes. The information in Qt Assistant is usually in a simple format and easy to use. Qt Assistant has also good search tools.

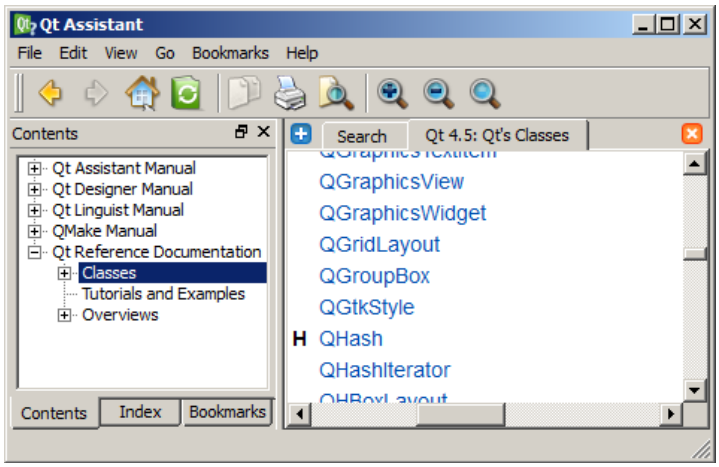

Figure 5. Picture of Qt Assistant

Another good tool is Qt Designer. It makes it possible to design GUI:s fast and easy with layouts. Layouts make it possible that GUI:s look the same in different devices. Qt takes care of scaling GUI components automatically for the resolution used on devices. Qt Designer generates files that have extension .ui. It is an XML-file that is used for creating C++ code automatically. Own Qt widgets can be used with Qt Designer if they are promoted for some Qt Designer's widget.

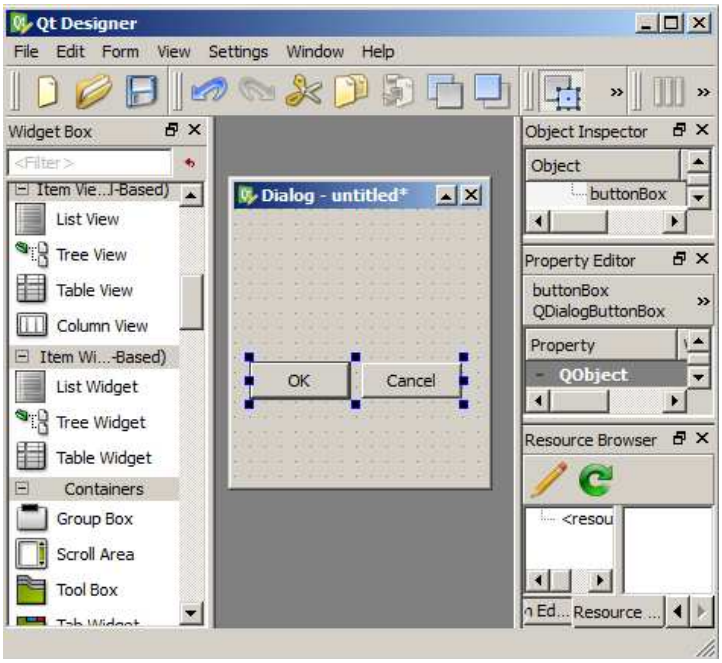

Figure 6. A picture of the Qt Designer.

Qt Creator is an IDE (Integrated Development Environment) which combines text editor, debugger, Qt Assistant and Qt Designer. Compared to other popular IDE used for Qt development, Java-based Carbide C++ IDE, Qt Creator is more lightweight. The features of Qt Creator include text editor with code completion and real time error and warning indicators. There is no support for S60 or Maemo emulators in the Qt Creator. In future there might be emulator support in Qt Creator. [17]

4.3 Using Qt in Mobile Environments

Qt is originally developed for desktop usage. Qt's memory usage has been optimized for performance and not for such small memory usage that is typical for mobile applications. This is the reason why some components have been

refactored especially for mobile environment usage. Some of the Qt's components differ between S60 and Maemo. For example build tool chains, windowing systems and file systems are different. Software binary deliveries differ also very much. S60 uses signed SIS (Symbian Installation Source) packages while Maemo uses DEB (Debian) packages. [2] [13]

Development environments between S60 and Maemo are totally different. S60 software development is done in Windows operating system. It is possible to use Carbide C++ IDE in Windows. Maemo developing is done in Linux operating system. Linux needs a tool named Scratchbox. It is platform in platform. Scratchbox works as a running and cross-compiling environment while developing the Maemo applications. Scratchbox needs device specific applications. Those device specific applications usage could be compared as SDK:s in S60 side. There are no Maemo devices with phone features on the markets yet. [2] [13]

5 PORTABILITY

5.1 Qt differences in S60

Building Qt for S60 application differs from building standard S60 application. The Symbian side tool chain is the same but the standard Qt's build tools **qmake** and **make** are used as a wrapper around original Symbian build tools. Command **qmake -project** can be used for creating a project file. This is needed if there is a folder which contains only .cpp, .h and .ui files and there is no Qt's project file .pro. Command **qmake** generates same files as S60 tool **bldmake bldfiles**. Those files are bld.inf, .mmp file, .reg, .rss, .mk extension makefiles and Makefile. Makefile is used as a wrapper around normal Symbian build command **abld.bat**. S60 specific make options are listed in Figure 8. [5] [9]

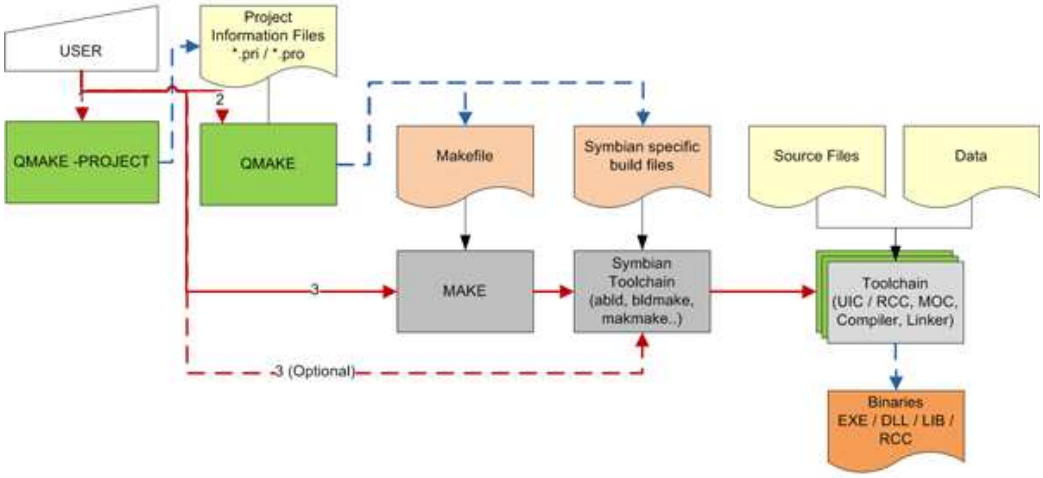

Figure 7. This image shows how Qt for S60's tool chain works. [9]

make	Creates abld.bat and makefiles and builds for emulator
make clean	Removes abld builds and makefiles
make debug	Creates all debug builds (all udeb:s)
make debug-armv5	Creates armv5 debug build
make debug-gcce	Creates gcce debug build
make debug-winscw	Creates winscw debug build for emulator
make release	Creates all release build (all urel:s)
make release-armv5	Creates armv5 release build
make release-gcce	Creates gcce release build
make run	Builds winscw and runs it on emulator

Figure 8. This table shows that how command **make** could be used with S60 environment. [9]

5.2 Creating SIS-package and Platform Security

Qt itself and libraries which are using it don't need other signing than self signing. S60 has its own platform security architecture. Platform security provides a platform with the ability to defend itself against malware or badly implemented programs. Symbian devices are running many different servers. There are public S60 API:s (Application Programming Interface) to connect those servers. Using of those API:s may need platform security capabilities. For example, if a

programmer wants to use GPS-data from the phone then using of the Location capability is needed. If Qt application needs to use GPS-data then Location capability must be added in the project file like this:
symbian:TARGET.CAPABILITY += CapabilityName. [5] [6] [7] [9]

Because of S60 platform security programmers may need to add two things to the .pro file if some of the capabilities are used. The project file needs line like: **symbian:TARGET.UID3 = 0xE0000001**. This tag UID3 (Unique Identifier) identifies the application so that it is possible to sign SIS package with Symbian Signed service. For Open Signed Online service application's UID must be between ranges: **0xE0000000 - 0xEFFFFFFF**. The signing service can be found from: https://www.symbiansigned.com/. S60 specific libraries or dynamically linked libraries can be set on project file too. Syntax for setting up libraries: **symbian:LIBS += -llibraryname**. S60 specific project file parameters are listed in Figure 9. [5] [6] [7] [9]

```
TARGET = ApplicationEx
symbian:LIBS += -llibraryname -llib2
symbian:TARGET.UID3 = 0xE0000001
symbian:TARGET.CAPABILITY                    +=
UserEnvironment
```

Figure 9. Here is example of self signable S60 specific project file. [5] [6] [7] [9]

Qt applications in S60 environment don't need any capabilities for them but if some S60 specific API:s are used then application signing is needed. User grantable capabilities are listed in the “Basic set”. If “Extended set” capabilities are needed those SIS-packages must be signed with Symbian Signed signing. “Phone manufacturer approved set” capabilities must be signed with manufacturer certificate. If capabilities and UID are not correctly set for a program that is going be installed in the phone then package doesn't work or install correctly.

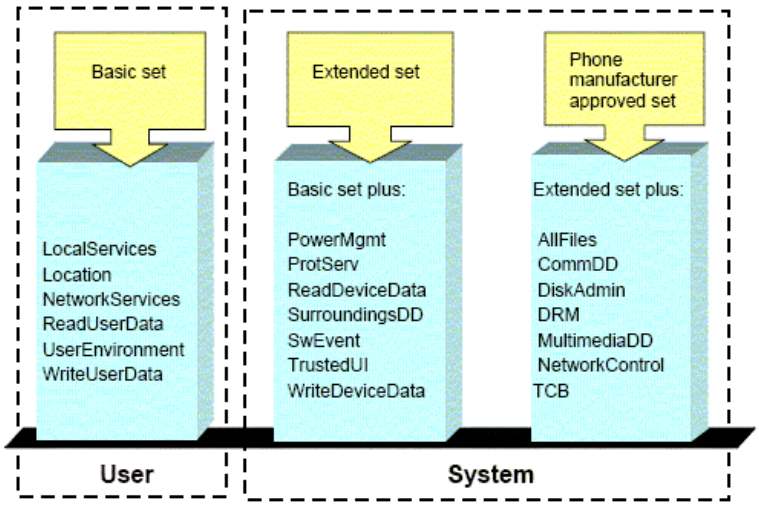

Figure 10. This image shows Capabilities. [6]

After successiful running of the command **make release-gcce** it is possible to create package from file ProjectFileName_gcce_urel.pkg. SIS-package could be created with command **createpackage -i ProjectFileName_gcce_urel.pkg**.

5.3 Maemo specific differences in Qt

The biggest difference between Maemo and S60 is that devices are using different operating systems. All Maemo platforms are based on GNU/Debian Linux. Because of this packaging goes nearly like when creating a standard Debian package. In Maemo environment there are no Servers. Instead there are Daemons. There is not S60 kind platform security. In contrary to S60, the platform security in Maemo devices is taken care of by different user right levels. The highest user level is Root user. Application or daemon can act as a user and it can be granted to do only things that are wanted.

5.4 Creating DEB-package

Packaging in Maemo is quite complicated. Here are the simplified steps that are needed for creating DEB-package: [11] [12]

1. Project folder must be formed like: "**../myapp-0.1**". [11] [12]

2. All files from that directory must be copied to folder: "**../myapp-0.1/src**" and "**../myapp-0.1/src/myapp.pro**" must be renamed to "**../myapp-0.1/src/src.pro**". [11] [12]

3. The next step is creating a new project file: "**../myapp-0.1/myapp.pro**". Content of the project file must be like this: [11] [12]

```
QMAKEVERSION = $$[QMAKE_VERSION]
ISQT4 = $$find(QMAKEVERSION, ^[2-9])
isEmpty( ISQT4 ) {
   error("Use the qmake include with Qt4.4 or greater, on
Debian that is
   qmake-qt4");
}
TEMPLATE = subdirs
SUBDIRS  = src
```

Figure 11. DEB package specific options in .pro file.

4. Following command creates Debian specific files needed for creating the package:

dh_make --createorig --single -e maintainer@email.org -c gpl [11] [12]

5. After executing previous command orig.tar.gz source code archive is created and sub folder debian with some files in it. Created files must be edited manually.

These instructions aren't enough for creating working Debian package. Files in that debian folder named control and rules must be edited manually. Developer must add the needed dependencies in the **control** file. Dependencies are the programming components like libraries which are needed for running building the application binary. Using of the **qmake** must be added for **rules**. See more in Annex 2.

Better and more comprehensive instructions for creating packages can be found from the following Internet resources:
http://wiki.maemo.org/Packaging_a_Qt_application

https://maemo.org/forrest-images/pdf/maemo-policy.pdf.

5.5 Installation without creating DEB-package

It is also possible to build Maemo binaries with Scratchbox's ARM tool chain or on the device with GCC-compiler. Built binary can be copied itself to the device with needed libraries. File transfer can be made with SSH connection through data cable or with WLAN connection (Wireless LAN).

Qt applications can be installed to the devices or to the Scratchbox emulator with command **make install** if the following lines are added to the project file: [9]

```
TARGET = myapp
unix:target.path = /usr/bin
unix:INSTALLS += target
```

Figure 12. Maemo make install specific options in .pro file.

5.6 Physical differences in the devices

Even now there are two kinds of devices on the markets. In the S60 side there are plenty of older 3rd Edition Feature Pack 1 and also newer 3rd Edition Feature Pack 2 devices. Both of those device types did not have a support for the touch screen. All of the Qt's GUI widgets are not working as they should in those S60 3rd Edition's devices. There are also S60 5th Edition devices. Those devices are the newest and have support for using touch screen. In those devices Qt's GUI widgets work better than those older S60 3.X devices.

At this moment it is complicated to design software that is usable on 3.X devices with the current release of the Qt for S60. It is possible to make GUI:s but it may need some hacking and customizing for the original Qt GUI widgets. At this moment creation of the GUI:s is not such easy and slick that it will be in future. In some applications the GUI may need to be separately designed depending on the availability of touch screen. With Qt Designer it is fast and easy to create

adaptable GUI:s for different type devices. Using of the layouts makes it possible that GUI components scale automatically for different type devices.

In the Maemo side there are only internet tablet devices on the markets. Those devices are technically older and don't have typical phone features. Existing Maemo devices are still good for testing and developing purposes. Those devices give quite a good view about what should be expected from the devices of the future.

5.7 Qt's Expandability

Qt itself does not have any features to support platform or device specific features. There are no built-in support for phone features telephony or SMS messaging. Forum Nokia has released Mobile Extensions for Qt for S60 package as technology preview. That package has wrapper classes for using already implemented, native S60 classes. Those extensions provide easy to use methods to S60 specific API:s. This Mobile Extensions technology preview can be found from: http://wiki.forum.nokia.com/index.php/Mobile_Extensions. [16]

One lack in those extensions is that all of the features don’t work exactly like in S60. Good thing in using of those extensions is that they make software development easier and also reduce number of possible buffer overflow cases. S60 uses generally manual allocation of memory. Allocation is usually done with pushing and popping data in cleanup stack. Qt's way is do this allocation automatically and that is the reason why number of human errors is smaller.

Another lack in using Mobile Extensions is that there are no Maemo devices released yet on the markets which support those features like phone and receiving SMS messages. In the ideal case same the software should work in both platforms S60 and Maemo. There are not any proofs of concepts about it released. In the future there might be Mobile Extensions which work in both S60 and Maemo.

5.8 Qt Unit testability

Unit tests could be used to check for some kind of errors which are possible to test some way. Unit testing is very important with functions which do operations to complex data structures. The idea of tests is that some function gets or sets some value or changes some variables. The returned values are compared automatically against correct value that is the expected outcome for the test case. If a test passes without errors then value can be written to some log file.

Qt has its own module QTest for writing unit tests. In Qt the unit tests are made with QTest functions. Those tests are portable if code under test is portable. If the code which is going to be tested is made with portable Qt code and native platform specific code is used only through wrapper classes then code under test is portable. If a test is made for a piece of code that combines for example S60 and Qt code then testable functions are limited only those parts that are made with Qt. It is also possible to use QTest for platform specific unit tests.

6 CODE MAINTENANCE

There is not yet much Qt code on S60 or Maemo platforms that needs maintenance or could be maintained. In future there might be same kind of Mobile Extensions package to Maemo. Biggest difference between maintaining those platforms is that Maemo needs a computer where Linux is installed whereas S60 needs Windows. It is possible to install Linux to virtual machine in Windows or vice versa but it is not as efficient as using native computer OS.

In the ideal case only maintenance is done for the Qt application code. In that way development is faster and easier. If some new features are made available to devices then platform specific implementation is needed. This kind of code is not usually directly portable and benefits of Qt don’t come true. There might also be

errors that are caused by Qt. Currently there are some listed bugs in S60 Qt pre-release version 4.5.0-garden. For example QListBox widget does not work correctly in the older 3.X devices. That kind of listed Qt related bugs will be fixed in future.

The code maintenance may be done different in a way on the future. There might be Qt framework related bugs. This kind of bugs may exist in some certain device model or device platform. Because Qt is quite a new tool in the mobile environment there might be lots of bugs at the start. Data security also might cause also problems. Mobile devices are nowadays even more and more like computers. It is very important to take care that data security is always up to date.

Maintenance with Qt might mean also maintenance of the wrappers or the extensions. That kind of maintenance can be platform specific if the fixes are made to the platform native implementations.

One difference in code maintenance between S60 and Maemo applications is that S60 use Mercurial based version controlling whereas Maemo side uses GIT as version controlling. That brings a need for both version controlling program experts because commands and functionalities are different.

7 PERFORMANCE

7.1 Device performance

As a part of the research for this thesis Qt's example application named Padnavigator was modified to work as 15-puzzle game. Padnavigator demonstrates using of QGraphicsView. This application was chosen as a part of the thesis because it needed much computation power. That kind of application is good for performance analysis.

Because Qt has long history in desktop usage some of its features are not optimized for mobile usage. In desktop computers there are usually more computation power for graphical output. During the research for there appeared lacks in graphical performance. The application implemented for this thesis was running with too slow frame rate.

In Maemo devices low graphics performance might be the reason because some irrational image data conversion may occur. Maemo's X windowing system is running in 16-bit mode. Images are processed usually in 32-bit mode. When image is drawn to a screen then image data needs 32-to-16 bit conversion. This operation is called blitting. It wastes too much CPU time and makes the application running slower. [21]

In S60 there were same kinds of problems in performance. In S60 case the problem might be the same or it might be caused by too slow memory allocation. Same kind of problems come out with desktop versions of the Qt. Using Open GL acceleration fixed those performance problems in desktop environment. In both Maemo and S60 there is not yet support for using Qt's Open GL features. In future releases of the Qt those issues will be fixed certainly.

7.2 How efficient it is to produce code with Qt

Maemo and S60 devices differ physically quite much. There are only a couple Maemo devices in the markets and none of them is capable to be used as a normal GSM phone. Maemo devices have a touch screen. There are only couple S60 phones in the markets which have a touch screen. This thing must be taken care of when developing GUI:s for different types of devices.

Usability in different type devices depends very much on GUI:s. Sometimes it is not possible to make things in GUI:s that work well with touch screen and standard phone buttons. Currently some of Qt's GUI components are not working very well in S60 phones. For example QLineEdit and QSpinBox components are working well only in S60 5th Edition phones. Those issues will be fixed in future releases. [22]

Implementing applications with Qt is faster than with native S60 code. Qt uses automatic memory management and that should reduce the number of code errors. Qt has its own multipurpose data type QVariant. It acts like a union for most common Qt data types. Different Qt data types can be stored to QVariant and read later to other data types. That kind of feature makes also development easier and faster and potentially reduces probability of errors.

7.3 How efficient it is to test with Qt unit tests

Unit tests can be made in the way they work with many different platforms. This reduces the time needed for writing unit tests. Implementation which is going to be tested must be written with Qt. This means that unit test must use public Qt implementation and platform specific code must be wrapped in private implementation. If tested code uses some native code then unit tests might need also some platform specific code.

With QTestLib tool it is possible to make test suites. In that way it is possible to run multiple unit test cases with starting only one test program. This makes unit testing faster. It is possible to write test results to XML-file and add style sheet for that file. Thus the results could be formatted into a report.

8 IMPLEMENTING EXAMPLE APPLICATION WITH QT

8.1 Installing developing environments

Implementation of the example application was started by installing developing environments. Maemo environment needs computer that have Linux. The Linux installation needs Scratchbox cross-compilation toolkit installed. It is possible to install Linux to virtual machine but in practice the usage of virtual machines causes long delays and even occasional jams. Installation time for fully installed development environment for Maemo developing is about 5-6 hours. If developer doesn't has any previous Linux experience it takes much more time. Linux has own command line commands and which are different as Windows. [23] [24]

In the Windows world there must be Carbide C++ 2.0 IDE installed. S60 SDK for 3rd Edition FP2 or 5th Edition is needed too in Windows. After those prerequisites are installed then Qt for S60 can be installed. It is more straight-forward to install Qt for S60 in Windows than installing Linux and Scratchbox for Maemo. It takes about 2-3 hours to install development tools in Windows. Qt libraries must be compiled manually in both Linux (Maemo) and Windows (S60) during development environment installation process. This takes most of the time when installing environments for the developing. [25]

8.2 Implementing example application

Example application is based on Qt's example application, Padnavigator. In Qt's version 4.2 there became a new method, QGraphicsView, to draw graphics. Graphics views are designed to be the future way to produce elegant looking GUI:s and graphics. It seems like the graphics views are still not powerful enough to run smoothly. Qt releases in future and bringing the Open GL support to the Qt in mobile environments will help some of those issues.

Padnavigator example application loads images in QPixmaps objects. Those objects are set to subclass RoundRectItem widget. For this thesis Padnavigator example was modified to 15-puzzle game. There was also an idea that some picture was taken as a starting point and it was cut to 15 pieces. The RoundRectItem pieces were shuffled after they were textured with image pieces. Objects needed also special id:s so that right places in the grid could be checked. Game needed logic that checks if all of the pieces are in correct positions. Also shuffle logic was needed.

Padnavigator example application was designed only to be a good looking demo. There was no logic that identified pieces. It was little complicated to modify the program in a way that it started to work like a 15-puzzle game. Different pieces were identified by setting objectName to every piece.

15-puzzle design was started on S60 platform with much older Qt version 4.4.2-pyramid. That caused problems in developing because that version doesn't have full support for phone keyboard and phone specific soft keys. The developing was started with S60 because its features were not such mature state as Maemo. Developing had to make with the rules of the weakest link. It was in this case S60.

Qt application on S60 needs to be set shown in full screen mode. It can be done with showFullScreen() function. S60 doesn't have support for application windows whereas on Maemo this is possible. Portable Qt applications must be designed the way they usable to use with devices which have only keypads and no

touch screen. In future this thing might get some improvements to solve device control depending issues.

Figure 13. Picture of Qt's Padnavigator based 15-puzzle game

9 RESULTS

9.1 Platform independence

Qt software platform independence depends mainly on two things. All of the Qt modules are not ported to both Maemo and S60. Some of the Qt modules are still in developing state. There will be Phonon multimedia module, but at the moment it does not have all features yet implemented. It is not feasible to port Qt applications between Maemo and S60 if some of those use modules which is not ported yet.

S60 devices have that kind of problem that there are devices with touch screen and those where are only keypad and soft keys. Some of the Qt Gui module components are not working smoothly with those devices with keypad and soft keys. Usability between the applications may be different if there is no keypad or touch screen available. At the moment it is possible to make own subclasses from Qt:s widgets and add needed keypad event handlers. Qt software fixes the detected problems in next Qt releases.

Naturally Qt does not currently bring any solution for that problem that there are also other devices than S60 and Maemo. If Qt get ported to RIM, iPhone and Windows Mobile it increase portability of Qt applications in mobile environments.

9.2 Code maintenance

Code maintenance becomes easier with Qt. It is easier to maintain the code when the applications are portable. Implemented applications will work with other platform devices. At the beginning there might be more Qt related problems like with S60. That kind of problems takes longer time to get solved because new releases of the Qt are needed. After those problems that are found in the early

phase of porting Qt are fixed and Qt is more mature then developing applications between S60 and Maemo is possible and it faster to maintain.

9.3 Performance

Qt is still in development state and has some performance problems. Applications which use QGraphicsView widget seem to perform quite poorly. QGraphicsView seems to performance poorly also in desktop environment if it is not used with Open GL support. In future Qt's graphical performance might be better. This may be possible though if new Qt versions work faster or t Open GL support become available in Qt and devices. It currently seems that it is even hard to make a simple animation with bouncing ball on Maemo or S60 with Qt.

Coding performance is good with Qt. According to personal experience, the implementation speed is much faster compared to native S60. In Qt data types are allocated and de-allocated automatically which reduces the number of human errors.

9.4 Testability

Same unit tests can be used in both Maemo and S60 if test code is made with pure Qt. If test code uses some native code, then test can only be used on the platform it is implemented for. There have not been any portable unit test methods before Qt which can be used in Maemo and S60.

10 SUMMARY

Code less, Create more, Deploy everywhere – says Qt's promotion phrase. Qt is still in developing state. In the near future Qt will be a very interesting option for mobile software development once its “childhood diseases” get solved. It will reduce costs because program components can be reused as portable code. Currently it is only possible to learn how to implement software with Qt in mobile environment. It is not ready for commercial products because all needed Qt

modules are not yet available for Maemo and S60 and some performance issues exists.

At its current state Qt keeps its promises for being easier and faster to implement. It reduces also probability of errors. Qt doesn't remove the need for unit tests on every platform but it should make portable unit tests possible at some level. In future it will be possible produce elegant looking GUI:s with Qt for mobile devices. Therefore Qt will also be a tool to seriously challenge and compete with the GUI:s of iPhone and RIM.

REFERENCES

[1] Wikipedia – Smartphone. [WWW]. [referenced 30.03.2009]. Available: http://en.wikipedia.org/wiki/Smartphone

[2] Wikipedia - Symbian OS. [WWW]. [referenced 30.03.2009]. Available: http://en.wikipedia.org/wiki/Symbian_os

[3] Wikipedia – EPOC. [WWW]. [referenced 30.03.2009]. Available: http://en.wikipedia.org/wiki/EPOC_(computing)

[4] Blanchette, J. & Summerfield, M. C++ Qt GUI Programming with Qt4. xix-xxi pp.

[5] qmake Manual. WW]. [referenced 30.03.2009]. Available: http://doc.trolltech.com/4.5/qmake-manual.html

[6] Platform security. [WWW]. [referenced 15.04.2009]. Available: http://developer.symbian.com/main/documentation/sdl/symbian94/sdk/doc_source/guide/platsecsdk/index.html#guide.platsec.index

[7] Platform security [WWW]. [referenced 25.03.2009]. Available: http://www.symbian.com/developer/techlib/v9.2docs/doc_source/guide/platsecsdk/index.html#guide.platsec.index

[8] Open Signed Online [WWW]. [referenced 25.03.2009]. Available: https://www.symbiansigned.com/app/page/public/openSignedOnline.do

[9] Integration of Qt and S60 build systems. [WWW]. [referenced 30.03.2009]. Available: http://library.forum.nokia.com/topic/Qt_for_S60_Developers_Library/GUID-19FFA67B-D2D4-4282-9E95-D382BB4FD594.html

[10] Qt's Modular Class Library. [WWW]. [referenced 30.03.2009]. Available: http://www.qtsoftware.com/products/library/modular-class-library

[11] Packaging a Qt application. [WWW]. [referenced 30.03.2009]. Available: http://wiki.maemo.org/Packaging_a_Qt_application

[12] Maemo Packaging Policy. [WWW]. [referenced 30.03.2009]. Available: https://maemo.org/forrest-images/pdf/maemo-policy.pdf fwerf

[13] Wikipedia – Maemo. [WWW]. [referenced 13.04.2009]. Available: http://en.wikipedia.org/wiki/Maemo_(operating_system)

[14 Qt in the S60 environment. [WWW]. [referenced 30.03.2009]. Available: http://library.forum.nokia.com/index.jsp?topic=/Qt_for_S60_Developers_Library/GUID-A43924F3-1D37-4A84-93B8-917AB01A1629.html

[16] Qt for S60 Mobile Extensions.. [WWW]. [referenced 30.03.2009]. Available (Needs Forum-Nokia account):
http://wiki.forum.nokia.com/index.php/Mobile_Extensions

[18] Youtube - Qt Creator - 01 An Introduction. [WWW]. [referenced 20.04.2009]. Available:
http://www.youtube.com/watch?v=U7yje3D1UM4&feature=PlayList&p=22E601663DAF3A14&index=0&playnext=1

[19] Qt Software - Signals and Slots. [WWW]. [referenced 20.04.2009]. Available: http://doc.trolltech.com/4.5/signalsandslots.html

[20] Qt Software - Using the Meta-Object Compiler (moc). [WWW]. [referenced 20.04.2009]. Available: http://doc.trolltech.com/4.5/moc.html

[21] D.C.T.W.Y.C.D.T - Qt 4.4 and Maemo. [WWW]. [referenced 20.04.2009]. Available: http://ariya.blogspot.com/2008/08/qt-44-and-maemo.html

[22] Qt Labs - QtforS60KnownIssues. [WWW]. [referenced 20.04.2009]. Available: http://labs.trolltech.com/page/QtforS60KnownIssues

[23] maemo.org - SDK Releases. [WWW]. [referenced 04.05.2009]. Available:
http://maemo.org/development/sdks/

[24] Maemo QT4. [WWW]. [referenced 04.05.2009]. Available:
http://qt4.garage.maemo.org/packages.html

[25] Qt 4.5 - Installing Qt on S60. [WWW]. [referenced 04.05.2009]. Available:
http://pepper.troll.no/s60prereleases/doc/install-s60.html

APPENDICES

```
/*
MyClass.h
*/

class MyClass : Qobject
{
   Q_OBJECT
public:
   //Constructor
   MyClass(QObject *parent = 0);

   void testSignal();

// Example slot
public slots:
   void exampleSlot();

// Example signal
signals:
   void exampleSignal();
}
```

```
/*
MyClass.cpp
*/

#include “MyClass.h”

MyClass::MyClass(QWidget* parent)
{
   testSignal();
   connect(this, SIGNAL(exampleSignal()), this,
SLOT(exampleSlot()));
}

void MyClass:: testSignal()
{
   emit exampleSignal();
}
```

Annex 1. Example of using signals and slots.

```
#!/usr/bin/make -f
APPNAME := my_app_name
builddir:
       mkdir -p builddir

builddir/Makefile: builddir
       cd builddir && qmake-qt4
PREFIX=/usr ../$(APPNAME).pro

build: build-stamp

build-stamp: builddir/Makefile
       dh_testdir
       # Add here commands to
compile the package.
```

```
	cd builddir && $(MAKE)
	touch $@

clean:
	dh_testdir
	dh_testroot
	rm -f build-stamp
	# Add here commands to clean
up after the build process.
	rm -rf builddir
	dh_clean
install: build
	dh_testdir
	dh_testroot
	dh_clean -k
	dh_installdirs

	# Add here commands to install
the package into
debian/your_appname
	cd builddir && $(MAKE)
INSTALL_ROOT=$(CURDIR)/debi
an/$(APPNAME) install
# Build architecture-independent files
here.
binary-indep: build install
# We have nothing to do by default.

# Build architecture-dependent files
here.
binary-arch: build install
	dh_testdir
	dh_testroot
	dh_installdocs
	dh_installexamples
	dh_installman
	dh_link
	dh_strip
	dh_compress
	dh_fixperms
	dh_installdeb
	dh_shlibdeps
	dh_gencontrol
	dh_md5sums
	dh_builddeb

binary: binary-indep binary-arch
.PHONY: build clean binary-indep
binary-arch binary install configure
```

Annex 2. Example of debian/rules file.