HELSINKI METROPOLIA UNIVERSITY OF APPLIED SCIENCES

Information Technology (Degree program)

Multimedia Communications (Specialization)

Master's Thesis

Cross-Platform Application Development on Symbian

Author: John Alvin Mathew
Instructor: Ville Jääskeläinen

Approved: 18.05.2010

Kari Björn
Programme Chair

PREFACE

I would like to thank all the people who have helped and inspired me during my Master of Science studies. Qt is an evolving framework and it was a challenge to keep up with the changes taking place on a daily basis.

I would especially like to thank my boss Kari Kurppa from Elektrobit Wireless Communications for providing me with the resources to be able to become acquainted with the world of Qt. I would also like to thank Mr Jouko Kurki from Metropolia University for his support and guidance through the entire MS course. Also I would like to thank Dr Marjatta Huhta for her guidance and teachings on how to write a good Master's thesis.

Helsinki, May 18, 2010

John Alvin Mathew

ABSTRACT

Name:	**John Mathew**
Title:	**Cross-Platform Application Development on Symbian**
Date:	**04 May 2010** Number of pages: **51**
Degree Programme: **Information Technology** Specialisation: **Multimedia Communications**	
Instructor: **Ville Jääskeläinen**	

This thesis takes an in depth look at how a cross-platform application development framework like Qt was ported on to Symbian. One aim of this work is to study the various underlying layers of Qt, interaction with the various Symbian/ S60 components, build systems integrations, native OS considerations and memory management. Another objective is to examine Qt application development and portability.

The theoretical basis of this study lies with the Qt application framework, the Symbian operating system and the User Interface framework of Symbian which is S60. The study investigates the porting of Qt on top of Symbian, and includes an overview of Open C framework available in Symbian and its usage by Qt. A particular feature like audio is chosen to study the end to end behavior of the cross platform technology. In order to obtain a better understanding of the theory and also to demonstrate the portability and ease of development using the Qt framework, an audio application is developed which is capable of playing mp3 files on a phone. That includes the use of Software Development Kits, Integrated Development Environments, its configurations and how to execute the developed application on a Smartphone. The development of this application also enables the study of the integration of the build systems, platform security and memory management. Through this thesis we understand how a cross-platform application framework can be developed on top of Symbian.

Based on the findings, the current state of Qt and its future is discussed. Qt is widely used in the development of Graphical User Interface programs and also for non-GUI programs such as console tools and servers. QT uses a “write once, compile anywhere” approach. Using a single source tree and a simple recompilation, applications can be written for Windows, Linux, Solaris, MacOS, Windows CE and Symbian. In the near future Qt will be a very interesting option for the mobile software development.

Key words: QT, Symbian,S60

TABLE OF CONTENTS

ABBREVIATIONS / ACRONYMS

API	Application Programming Interface
ASSP	Application Specific Standard Product
AVI	Audio Video Interleave
DLL	Dynamic Link Library
DOM	Document Object Model
FEP	Front End Processor
GCCE	GNU Compiler Collection for Embedded
GUI	Graphical User Interface
IDE	Integrated Development Environment
MOC	Met Object Compiler
Mp3	MPEG-1 Audio Layer 3
MPEG	Moving Picture Experts Group
MMF	Multimedia Framework
OS	Operating System
POSIX	Portable Operating System Interface [for Unix]
P.I.P.S	POSIX on Symbian
SDK	Software Development Kit
SIS	Symbian installation package
SSL	Secure Sockets Layer
SVG	Scalable Vector Graphics
UIQ	User Interface Quartz
XML	Extensible Markup Language

1 INTRODUCTION

A Smartphone OS is the interface between hardware and user application, which is responsible for the management and coordination of the various components of the Smartphone and also acts as a host for the computing applications that run on it. The Symbian platform is an open source software based operating system that has been developed from the ground up for low power, small memory devices such as mobile phones. There are over 350 million Symbian OS based smart-phone devices in the market [2]. As a result, thousands of applications have been developed for Symbian and there is a huge developer community also present. With the growing number of Smartphone vendors the expectations for the Smart-phone operating systems have increased, as well. This has resulted in the devel-opment of new operating systems. This, in turn, has created a need to develop applications for each of these operating systems

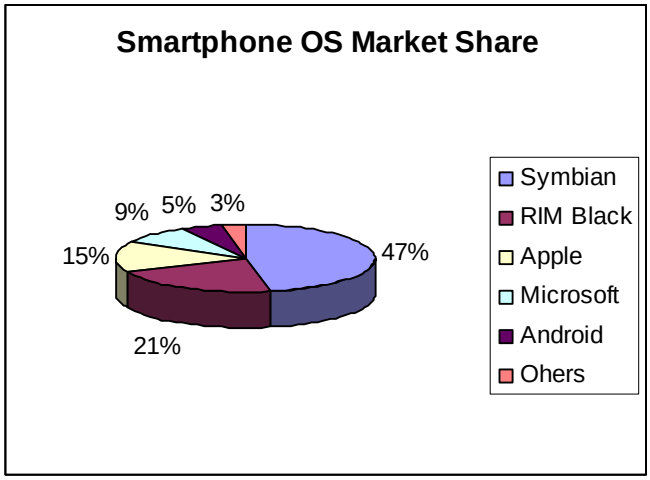


Figure 1. Smartphone OS Market share [1]

The current approach that each of the Smartphone OS vendor has adopted to get applications developed for their OSes is to release free SDKs for their OS. This is why developer communities were formed mainly consisting of amateurs acting out of interest and a few experts. The problem with this approach is mainly time to market and stability. Security holes are another major factor as it is not uncommon to consider security related issues during early development stages. Also the knowledge to write efficient code and applications that utilize the power of the OS

may be missing. So the only solution to all this is to port the existing applications that have become mature, stable and secure with fixes over time to these new OSes.

This brings in the need for a common application development framework. It should be possible to write applications in this framework, and these applications should run on any platform to which the framework is ported. In the mobile software world application portability means power. From a developer's point of view it means more users for the application and less cost of development. From a user's perspective it means the look and feel of the application remains the same on multiple environments. Mobile devices are evolving into increasingly sophisticated, general purpose computers. This means that a user would expect many of the desktop applications to run on the mobile device. In order to meet this goal a developer should be able to write applications once and deploy them across desktop, mobile and embedded operating systems without rewriting the source code.

A popular cross-platform application development framework which satisfies all of the above requirements is QT (pronounced “cute”). It is widely used in development of GUI programs and also non-GUI programs such as console tools and servers. Some popular applications based on QT are Google earth, Adobe Photoshop album and Skype. At the time of writing this thesis the QT framework had already been ported to the following OSes: Linux, MacOS, Windows, Embedded Linux, Windows CE, S60 Platform, Java, Android and iPhone OS

Thus by developing applications in QT the developer is guaranteed that his application can be easily executed on any of the OS that has QT ported on it. Also implementing an application with QT is much faster compared to any other application development framework. With the use of the various QT tools which will be explained in the sections to come it can be seen that application development , portability and maintenance is much easier using QT compared to other application development frameworks. This study focuses on how cross-platform frameworks like QT are ported on to an OS. The main objectives of the study are described in the next section.

1.1 Objectives

This thesis takes an in depth look at how QT was ported on to Symbian, the underlying layers, and interaction with the various Symbian/ S60 components. This is done by developing an experiment application using Qt Integrated Development environment (IDEs) and tools and studying the effectiveness of this application. The objectives of this thesis can be categorized as follows.

- To study the various QT application framework components
- To investigate the interaction between the QT layers and S60.
- To observe the role of Open C
- To find out how to create a QT application using QT tools - SDK's, IDE's and configurations.
- To understand the interaction between the QT audio framework and Symbian audio framework.
- To study how of Symbian and QT build were integrated.
- Study of Qt memory management and platform security

This thesis is written in five sections. The first section deals with the background information of the software components such as S60 Qt and Open C that will be dealt with in this thesis. The second section describes the methods and materials used in the thesis in such a way that the experiment performed will be replicable. The tools and software used are explained in this section. The third section explains how the application developed for this experiment was developed at a component level. This section has 3 subsections including are installation, development and deployment. The fourth section contains the analysis of the experiment and the final section focuses on discussion and conclusion.

2 BACKGROUND

The background section covers the necessary background information about the components involved in the subject area of this thesis. It is mainly divided into four sections. The first section describes the Symbian OS and its various layers. The second section deals with the application framework that sits on top of Symbian OS which is S60. It also describes the basic architecture of an S60 application which will help to understand the interaction with Qt better in the later sections. The third section gives a general overview of the Qt framework and its various components. The final section deals with the Portable Operating System Interface (POSIX) compliant software support provided by Symbian through the Open C libraries which is the basis of the Qt support by Symbian.

2.1 Symbian OS

Symbian OS is an advanced, customizable operating system licensed by the world's leading mobile phone manufacturers. It is designed for the specific requirements of advanced 2.5G and 3G mobile phones and includes a robust multi-tasking kernel, integrated telephony support, communications protocols , data management, advanced graphics support, a low level graphical user interface framework and a variety of application engines. The simplest architectural view of Symbian OS is the layered view given below.

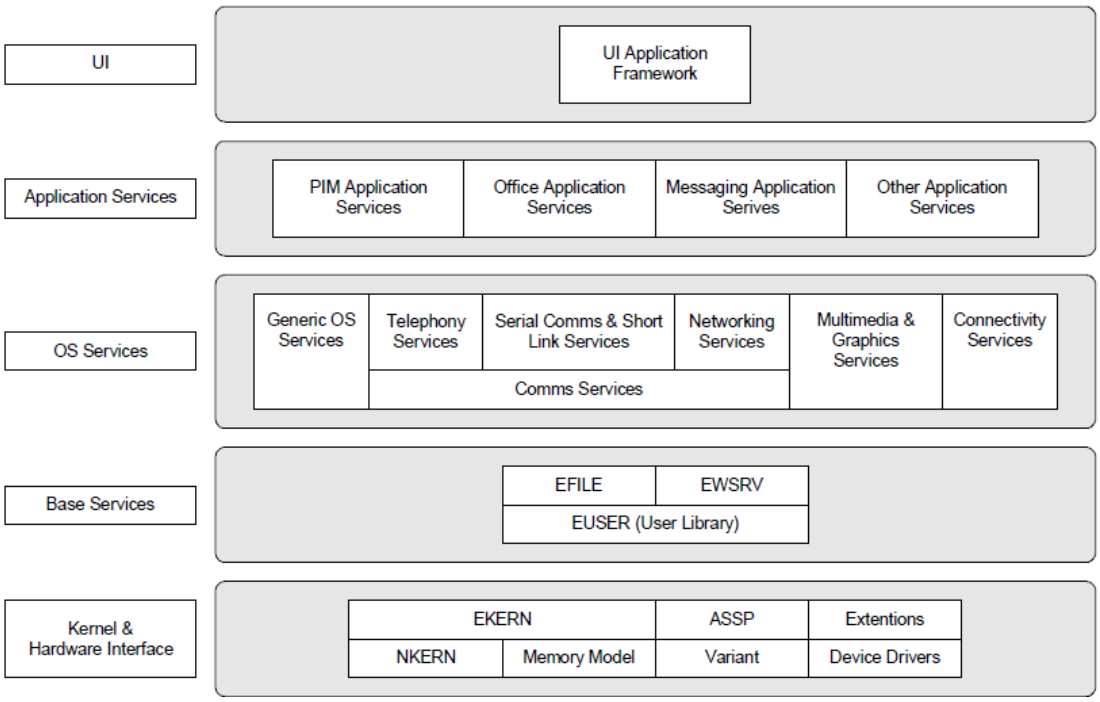


Figure 2. Symbian OS architecture [16]

The topmost layer of Symbian OS, the UI Framework layer provides the frameworks and libraries for constructing a user interface, including the basic class hierarchies for user interface controls and other frameworks and utilities used by user interface components. The Application Services layer provides support independent of the user interface for applications on Symbian OS. The OS Services layer is, in effect, the ‘middleware’ layer of Symbian OS, providing the servers, frameworks, and libraries that extend the bare system below it into a complete operating system. The foundational layer of Symbian OS, the Base Services layer provides the lowest level of user-side services. In particular, the Base Services layer includes the File Server and the User Library. The microkernel architecture of Symbian OS places them outside the kernel in user space. The lowest layer of Symbian OS, the Kernel Services and Hardware Interface layer contains the operating system kernel itself, and the supporting components which abstract the interfaces to the underlying hardware, including logical and physical device drivers and ‘variant support’, which implements pre-packaged support for the standard, supported platforms.

The native programming language of Symbian is Symbian C++. It is a variant of C++ developed for resource constrained devices. Though this version of C++ takes care of robustness and memory efficiency, it is relatively difficult to use for programmers. Standard C++ developers find it tough to familiarize with Symbian C++ due to its limited source code availability and small developer community. This issue is hoped to be addressed with Symbian being made open source by the starting of the Symbian Foundation and gradual release of Symbian source codes by this foundation. One big step taken by Symbian in the area of application portability was to provide the support for POSIX on Symbian (PIPS) which will allow an ever increasing number of popular desktop middleware and applications such as web servers and file sharing software as well as applications based on other mobile operating systems to be easily ported to Symbian OS. In fact this is the starting point of porting the Qt framework to Symbian, which will be explained in detail in the coming sections.

2.2 S60 Platform

The S60 platform is the most widely used application framework for the Symbian OS developed by Nokia. It supports a large color screen and an intuitive interface, and it incorporates leading-edge communications and device technologies that interoperate safely and respond quickly. After the Symbian foundation was formed and Symbian was made open source, the S60 application framework was merged with Symbian to be called as Symbian platform [3]. The Symbian Foundation is a non-profit organization that stewards the Symbian platform, an operating system for mobile phones, based on Symbian OS which was previously owned and licensed by Symbian Ltd. So the initial port of Qt to Symbian was done to interface with the UI framework classes with S60. This will be explained in detail in the later sections. The following figure shows the S60 application framework layered architecture.

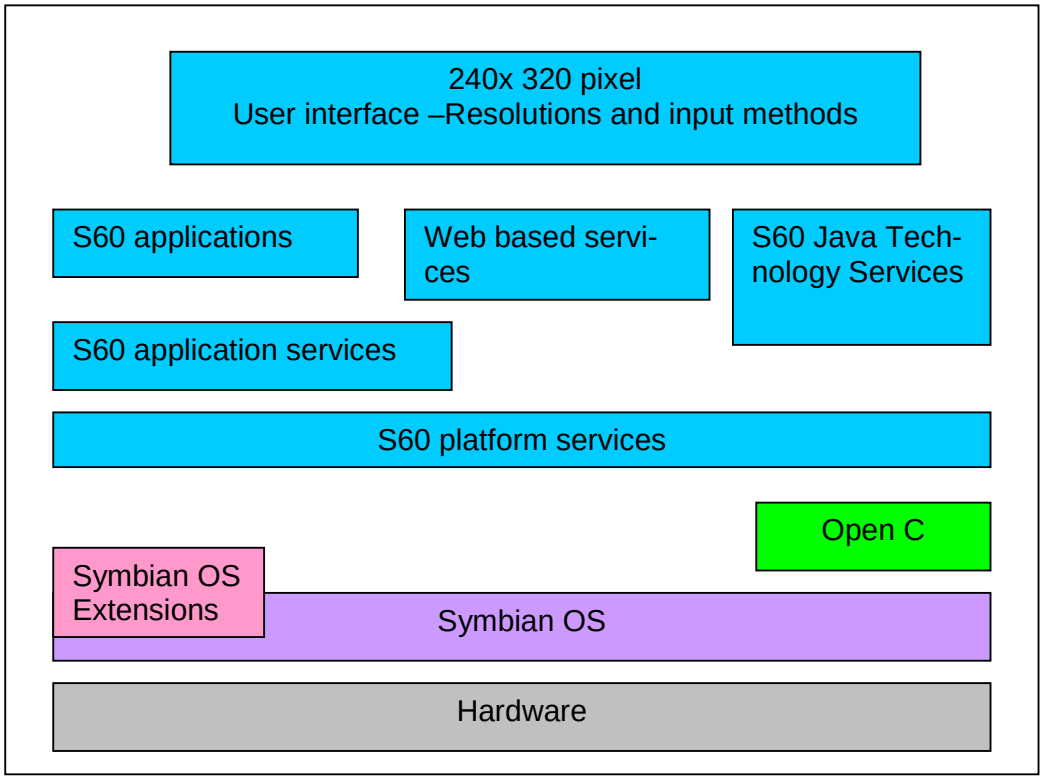


Figure 3. S60 Application framework layered architecture [19]

The various services provided by the S60 platform include

- Application Framework Services - providing the basic capabilities for launching applications and servers, state-persistence management, and UI components.

- UI Framework Services - providing the concrete look and feel for UI components and handling UI events.

- Graphics Services - providing capabilities for the creation of graphics and their drawing to the screen.

- Location Services - allowing the S60 platform to be aware of a device's location.

- Web-Based Services - providing services to establish connections and interact with Web-based functionality, including browsing, file download, and messaging.

- Multimedia Services - providing the capabilities to play audio and video, as well as support for streaming and speech recognition.

- Communication Services — providing support for local and wide area.

In order to understand how the Qt port on Symbian was done we need to understand the S60 application framework of S60 a bit. The following figure shows the objects to be used by an S60 application.

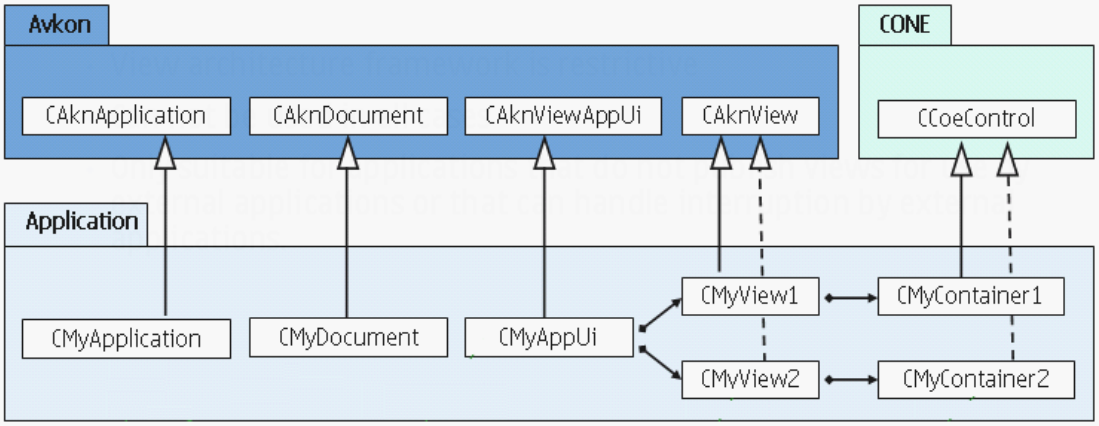


Figure 4. Objects of a typical S60 application [19]

We shall now take a look at the figure shown above in detail. Avkon is UI-Specific application framework layers implemented on top of the common Symbian OS UI framework, which is called Uikon. Cone is the control framework for the UI. The application class stores the properties of the application. The document class manages persistent storage related to the application. The UI class directs the menu and key events to the right controls. The container class contains one or more controls visible to the user. View classes are used to display the application pages. The application will activate and deactivate views based on the inputs from the user. The QT port of Symbian initiates the Avkon classes while starting an application. The details of this are explained in the following sections.

2.3 QT

Qt is a cross-platform application and UI framework that allows developers to write applications that can be deployed to desktop, mobile, and embedded operating systems without the need to rewrite the source code. Qt is a superset of standard C++ so developers can use either Qt's or standard C++ types, or a mixture of both (on the Symbian platform Qt is layered over Symbian's standard C/C++ compatibility layer). Qt is already used by hundreds of thousands of developers, and is notable as the framework used in the popular Linux KDE desktop, Google Earth and Skype. Migrating Qt applications to the Symbian platform is often no more difficult than a recompilation. Qt includes a rich set of widgets ("controls" in Win-

dows terminology) that provide standard GUI functionality. Qt introduces an innovative alternative for inter-object communication, called “signals and slots”, that replaces the old and unsafe callback technique used in many legacy frameworks. Qt also provides a conventional event model for handling mouse clicks, key presses, and other user input. Qt’s cross-platform GUI applications can support all the user interface functionality required by modern applications, such as menus, context menus, drag and drop, and dockable toolbars. [5]

The QT framework consists of modules. An application developer has to link to the binaries of modules whose functionality is required by the application. In this thesis the experiment application was an audio application so it is linked to the audio application which will be explained in detail in section 5.2. The rest of the Qt modules that have been ported to Symbian are shown in table 1 below.

QtCore	QtCore contains the core non-GUI functionality of Qt like Event, text, and time classes are part of this module. All other Qt modules rely on QtCore.
QtGui	QtGui extends QtCore with GUI functionality like Widget, graphics, image, paint, and style-related classes
QtNetwork	QtNetwork offers high-level classes (QHttp , QFtp) that implement specific application-level protocols and lower-level classes like QTcpSocket , QTcpServer , QUpdSocket to provide easier network programming.
QtScript	The QtScript module can be used to make Qt applications scriptable using ECMAScript, the standardized version of JavaScript. With the QtScript module it is possible, to add scripting support to a Qt application and thus allow users to add their own functionality in addition to what the application already provides.
QtSql	Classes for database integration using SQL. The module has SQLite3 implementation for Symbian OS as a backend.
QtSvg	The QtSvg module provides classes for rendering and displaying SVG content. QtSvg supports the static features of SVG 1.2 Tiny

QtTest	The QtTest module provides classes for unit testing Qt applications and libraries.
QtXml	QtXml provides a stream reader and writer for XML documents, and C++ implementations of SAX and DOM.
QtWebKit	QtWebKit provides classes for displaying and editing Web content
Phonon	Phonon module provides multimedia framework classes. With an experimental backend.

Table 1. Qt Modules

The above Qt modules are available in the Qt 4.6 Beta for the Symbian platform release.

Qt offers an 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 on how to use the 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. Qt assistant was extensively used in the making of this thesis. 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 the devices. Qt Designer generates files that have the 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. Qt Creator is an IDE (Integrated Development Environment) which combines text editor, debugger, Qt Assistant and Qt Designer. Qt creator is the IDE used for this thesis. 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 as well as warning indicators.

2.4 Open C

The need for portable software was realized even before Qt was developed. One such effort is the POSIX - Portable Operating System Interface [for Unix] standard. POSIX assures code portability between systems and is increasingly mandated for commercial applications. To enable such software to be easily ported to Symbian

OS, Symbian introduced frameworks such as P.I.P.S. and Open C that overlay native Symbian OS Application Programming Interfaces (APIs) with standards-compliant wrappers. These enable reuse of existing (standards-compliant) code when developing software for Symbian OS.P.I.P.S. and Open C are primarily porting aids that help you port POSIX-based software to Symbian OS, quickly and efficiently. The Qt port for Symbian utilizes these POSIX APIs provided by Open C which will be explained in detail.

P.I.P.S. (a recursive acronym for P.I.P.S. Is POSIX on Symbian OS) is a set of standard libraries that enable you to write and build POSIX-compliant code on Symbian OS v9.x. Available for both the S60 3rd Edition and UIQ 3 platforms, it comprises the four 'base' libraries - libc, libm, libpthread and libdl .Despite some limitations, P.I.P.S. can substantially reduce the time and effort required to port POSIX-based software to Symbian OS. Open C is a set of additional libraries built on P.I.P.S. and supplied by Nokia for the S60 3rd Edition platform only. The libraries provided are libssl, libz, libcrypto, libcrypt and libglib. These libraries are ported from their open source versions and provide the same interfaces. P.I.P.S. add-ons are libraries and tools that extend P.I.P.S. The first of these is the libz port for UIQ 3 platforms; other libraries that debuted in Open C will follow. Utilities such as telnetd and zsh are also part of this set.

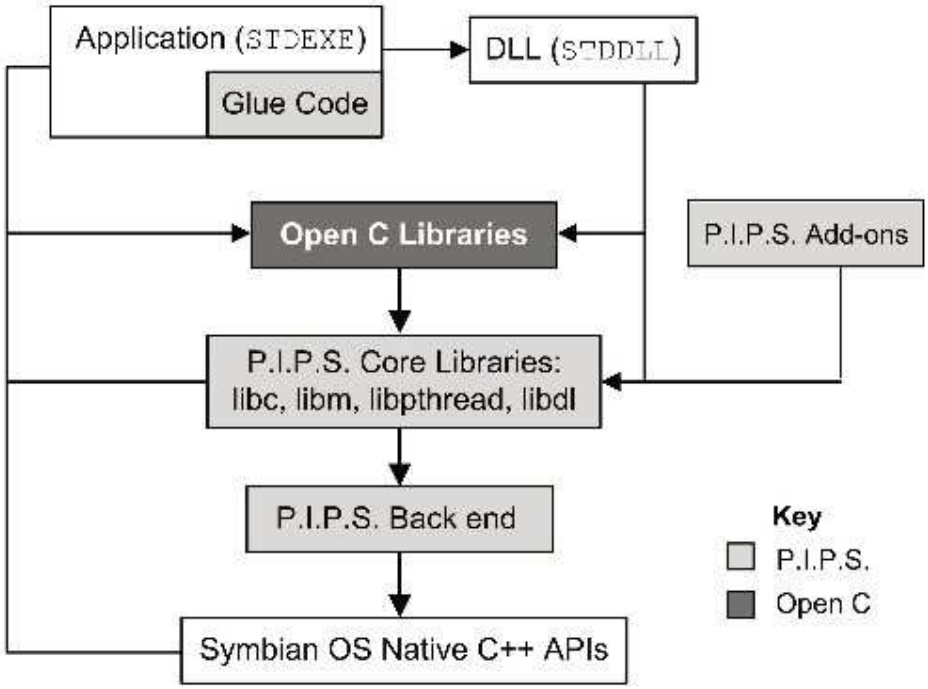


Figure 5. Symbian PIPS architecture [18]

The functionality of each of the P.I.P.S and Open C libraries are described below.

Library	Description
libc (Open C Libs)	Basic C programming routines: basic types, sockets, I/O, process, etc.
libm (Open C Libs)	Mathematical functions.
libdl (Open C Libs)	Dynamic loading and symbol lookup.
libpthread	POSIX standard interface for multiple threads.
libssl	Implements SSL v2/v3 and TLS v1 protocols.
libcrypto	Provides services for OpenSSL
euser (Symbian Libs)	Basic Symbian OS types and services: active scheduler, etc.

A Symbian OS application that uses main() or wmain() as the entry point is a P.I.P.S. application. For these applications, P.I.P.S. provides a static library to serve as glue code between the native E32Main() entry point and the user-specified main() or wmain(). libcrt0 is the glue code that you need to use for appli-

cations that start with main(). This static library has an implementation of E32Main within which it calls the library initialization method followed by calling main() written by the developer. This static library also gets command-line arguments and passes the same to main(). libwcrt0 is the glue code that you need to use for Unicode applications that start with wmain(). Section 5.2 explains how Qt uses this PIPS component of Symbian OS for portability.

3 DESCRIPTION OF METHOD AND MATERIAL.

This study is based on some experiments and desk research. A specific area was chosen to study the end to end functionality between the QT framework and Symbian OS. The audio playback functionality was studied in detail to understand the fine details of the cross-platform framework interaction with the native OS. A test Qt application was developed for this purpose. It is a simple audio player capable of playing mp3 files. This experiment application was developed on a Windows XP laptop. It was developed using the Qt application development IDE called the Qt Creator for Windows. Version 1.3.1 of the Qt creator was used for this thesis. Qt for Symbian version 4.6.2 was used as the Qt library for this thesis. The Qt creator supports testing the application using the S60 emulator. Nokia 3rd Edition FP1 SDK was used as the S60 version. But the S60 emulator had limited audio codec support. It is, therefore, not possible to do any kind of playback of mp3 files or any other audio formats. Thus, the audio functionality was tested directly on the device. The final testing of the application was done using the on target debugging tool called the app TRK on an actual device. A Nokia E63 device was used for this purpose. The TRK tool supports the step by step debugging which helped to set break points in various parts of the code and thus helped in understanding the code flow in detail which gave a clear picture of interaction between the two frameworks which could not be studied from literature available so far. The Qt creator IDE was used for source browsing. Information available from the web was also analyzed and presented in this thesis. In addition to the audio frameworks the following areas were analyzed.

- Qt Symbian Build Integration
- Qt on Symbian memory management
- Qt support for Platform security

Information required for the above was collected from the Symbian and Qt websites and by analyzing the Qt sources.

4 IMPLEMENTATION

In this section we create a Qt for Symbian application to study its operation and interoperability with the Symbian framework.

4.1 Phase 1: Tools Installation

The tools required for creating a Qt for Symbian application are as follows.

- S60 3rd Edition SDK – a combination of Symbian OS binaries and UI framework
- Qt Libraries for Symbian 4.6.2 for Symbian
- Qt Creator IDE

There are some add-ons that need to be installed too like active perl and open C which will be discussed in this section, as well.

4.1.1 Installing the S60 SDK

Before downloading the S60 SDK make sure that that active perl is installed in the system in a folder name that does not contain any space character. This is because perl scripts are used by the SDK. The site for active perl installation is provided in the appendix. Now download the S60 SDK from the forum Nokia website provided in the appendix. At the time of writing this thesis the S60 5th version SDK was available, but S60 3rd edition was used as it was compatible with the phone to be tested. Install the SDK to the same drive as the perl installation. The installation is straight forward. It is advisable to try building an example Symbian application and checking if the build succeeds in order to verify that the environment is functioning properly as it is very important in order for the rest of the steps to succeed.

4.1.2 Installing the Qt Libraries for Symbian 4.6.2 for Symbian

Before Qt can be used on Symbian the Open C/C++ plugin should be installed to

the Symbian environment. This plugin can be downloaded from the Forum Nokia website in the appendix. It should be installed to the S60 SDK installed in the previous step. Now download the Symbian 4.6.2 for Symbian libraries from the Qt website link provided in the appendix. During the installation process it will be asked to select the Symbian SDK that the Qt libraries need to be installed. So select the SDK installed in the previous step.

4.1.3 Installing the Qt Creator IDE

Download and install the Qt Creator 1.3.1 Binary for Windows from the link provided in the appendix. This should be installed on the same drive as your SDK installation. Qt Creator should automatically detect SDKs that have been configured for use with Qt. To ensure that the correct SDK is used as the Qt build target: Start Qt Creator from the windows start button: All Programs -> Qt Creator by Nokia v1.3.0 (open source) -> Qt Creator. Navigate to the Qt Versions settings at menu: Tools -> Options-> Qt4-> Qt Versions. Select the SDK installed in the previous steps in the Default Qt Version field. Press Apply to save the settings.

Now all necessary components to build a Qt application have been installed. This environment can be tested by building a sample application. This can be done by the following steps: Open Qt creation application. Go to File - > Open File or Project. Browse to your qt installation directory and select examples directory. Select an application of your choice and open the .pro file of that application. A .pro file is the project file for Qt. It is similar to the mmp file in Symbian. Now click on the edit tab on the left panel of the Qt creator. The source code of the application selected can be seen. In order to build this example, go to the Projects tab on the left pane of Qt creator. Here the default build configurations can be seen. To build the application for the S60 Emulator, change the tool chain from GCCE to WINSCW. This can be done by clicking on the “show details” tab of the general build settings section. Also to run the example application on the S60 Emulator, add the “Run the application on the Symbian Emulator “option in the Run settings section of the projects window. Now hit on the Green arrow in the left panel .This will compile and run the test application. Any build errors can be seen by selecting the Build issues

tab on the bottom panel. A successfully compiled application will not show any build issues in the build output and the last line would be Exited with code 0. The Qt creator Project settings window for this application is illustrated in Appendix C.

4.2 Phase 2: Development.

Our aim is to develop an mp3 player application capable of playing mp3 files. In Qt the multimedia framework is called phonon. It provides functionality to playback most of the common media formats. We will use the built in Qt widgets to provide controls to the user. Phonon provides its own widgets for volume slider, seek slider etc. The application we develop will be able to queue up music files and play them from the queue. More functionality can be added to this application but our purpose is to understand the Qt port on Symbian so only minimal functionality is developed. The S60 emulator can be used in the development phase to test the UI functionality only. Mp3 playback is not supported in emulator because there is no audio codec support in the emulator. Once the application has been developed testing will be done on the device using the App Trk tool.

4.2.1 Creating a New Application in Qt Creator.

Qt creator is a very useful tool to create applications in Qt. Its main features are smart code completion and navigation, error and warning indicators as we type and build in context sensitive help and integrated debugger, which was used heavily to create this application. The process of creating a new application is pretty straight forward. Select file ->new project or file - >Qt4 Gui application. Select a suitable name and location for your project. Then it asks for the modules to be selected. By default Qt core and Qt Gui modules are selected, the functionalities of which have been discussed in section 2.3. As we are developing a phonon based application we include the phonon module. Then an option to select the class names and file names for the project is given which can be left as such if you are ok with the default names. There is an option to generate form with the project. If this option is selected the GUI can be easily made by dragging and dropping controls on to the form using the Qt designer tool of the Qt creator. We do not select this option as we want to create our own widgets so that the signals and slots of

the widgets can be used in the application. We now have a new application created successfully. The main function of the application will be in a separate file typically named main.cpp. The functionality implementation will be done in mainwindow.cpp

4.2.2 Creating the UI.

The Ui design for the application will be as follows

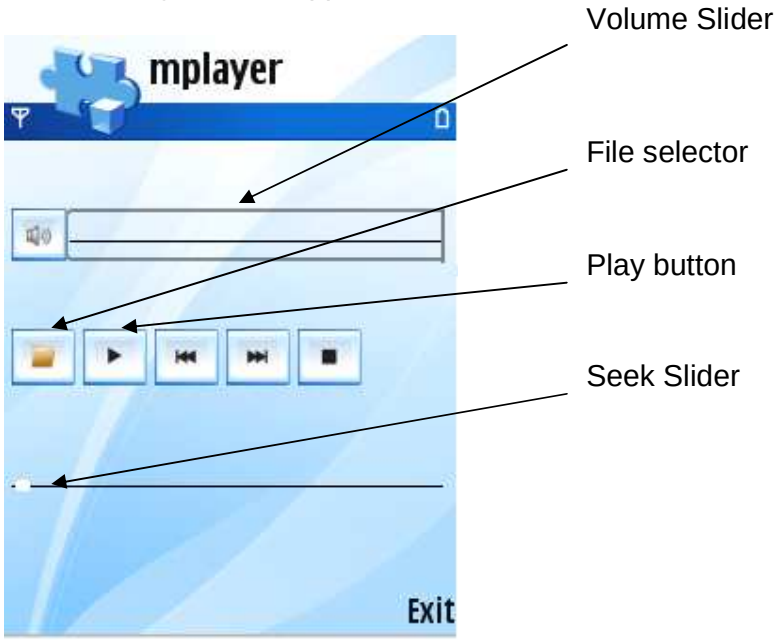


Figure 6. Experiment Application UI in S60 Emulator

Volume Slider

This is built using the Phonon::VolumeSlider widget. This slider also provides an icon to indicate if the audio is muted. It is possible to set the maximum value of the slider. By default, the minimum and maximum values of the slider are 0.0 (no sound) to 1.0 (the maximum volume the audio output can produce).

File selector.

The file selector consists of 2 parts - the icon and the process. The icon is created using the QPushButton widget of Qt. Push buttons display a textual label, and optionally a small icon. These can be set using the constructors and changed later using setText() and setIcon(). In our case we use the

QStyle::SP_DialogOpenButton pixmap.The functionality of this button is provided to the QFileDialog class.The QFileDialog class provides a dialog that allow users to select files or directories. The QFileDialog class enables a user to traverse the file system in order to select one or many files or a directory. The button is connected to the QFileDialog functionality using a Qt mechanism called signals and slots. Signals and slots are used for communication between objects. A signal is emitted when a particular event occurs in our case when the Open button of the file selector is clicked. Qt's widgets have many pre-defined signals so we use the clicked() signal of the Open button widget. A slot is a function that is called in response to a particular signal. Qt's widgets have many pre-defined slots, but it is common practice to add your own slots so that you can handle the signals that you are interested in. In our case we have created our own slot for file selection purposes called the addFiles().

The signals and slots mechanism is type safe: the signature of a signal must match the signature of the receiving slot (in fact a slot may have a shorter signature than the signal it receives because it can ignore extra arguments.) Since the signatures are compatible, the compiler can help us detect type mismatches. Signals and slots are loosely coupled: a class which emits a signal neither knows nor cares which slots receive the signal. Qt's signals and slots mechanism ensure that if you connect a signal to a slot, the slot will be called with the signal's parameters at the right time. Signals and slots can take any number of arguments of any type. 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. MOC generates automatically loose coupling connections used between signals and slots

All classes that inherit from QObject or one of its subclasses (e.g. QWidget) can contain signals and slots. Signals are emitted by objects when they change their state in a way that may be interesting to the outside world. This is all the object does to communicate. It does not know or care whether anything is receiving the signals it emits. This is true information encapsulation, and ensures that the object

can be used as a software component.

Play Button

The play button consists of 2 parts, the icon and the process. The icon is created using the QPushButton widget of Qt. The icon of the button is set using setText() and passing the QStyle::SP_MediaPlay parameter to the function. The button is connected to the Phonon::MediaObject using the signal and slot mechanism. In this case it is connected to the predefined Play() slot of the Phonon::MediaObject that we created.

Seek Slider

The seek slider is built using the Phonon::SeekSlider widget. The SeekSlider class provides a slider for seeking to positions in media streams. The SeekSlider connects to a MediaObject, and controls the seek position in the object's media stream. The slider will connect to the necessary signals to keep track of the slider's maximum, minimum, and current values. It will also disable itself for non-seekable streams, and update the media object when the current value of the slider changes.

Layouts

Qt includes a set of layout management classes that are used to describe how widgets are laid out in an application's user interface. These layouts automatically position and resize widgets when the amount of space available for them changes, ensuring that they are consistently arranged and that the user interface as a whole remains usable. In our case we used 2 of Qs's layout classes: QHBoxLayout to horizontally layout the file selector, play , forward, backward and stop buttons and QVBoxLayout to vertically layout the volume slider, horizontal layout discussed previously and the seek slider. Finally we created a widget consisting of this whole layout and made it to be initialized in the constructor function of the application.

Player setup

The Qt media player consists of a media object class and an audio output class. The state of play (play, pause, stop, seek) of an audio file is controlled by the me-

dia object. It keeps track of the playback position in the media stream, and emits the tick() signal when the current position in the stream changes. The AudioOutput class is used to send data to audio output devices. The AudioOutput class plays sound over a sound device. The audio output needs to be connected to a MediaObject using createPath(). To start playback, you call play() on the media object. This is done by connecting the play() slot to the play button. Similarly the stop button of the player is connected to the stop() slot of the media object. These connections are made using the connect() function during the initialization of the application. The current media source to be played is selected using the Media::Object currentSource () function. In order to enable the player to continue to play the next music file after one file has ended we use the aboutToFinish() signal of the media object. This signal is emitted before the playback of the currently playing audio file ends. We connect this signal to the slot implemented by the application of the same name. In this slot we call the MediaObject::enqueue function to set a new source. One observation of this implementation is that the aboutToFinish() signal will be emitted only if the previously playing file was played fully. If it was advanced using the seek slider, the aboutToFinish() signal is not emitted.

Another debugging technique that was used during the application development was to enable traces on to the Application panel of the Qt creator. This was done by using the qDebug object. The QDebug class provides an output stream for debugging information to the console, which in our case is the Application output window in the Qt creator.

4.3 Phase 3: Deployment

As mentioned in the previous section, it is not possible to test audio play back in the S60 emulator due to missing codec support. So the only option left for testing and debugging is the actual device itself. The Qt creator communicates with the device through a tool called App TRK. The App TRK tool is a small application which you install on your phone. It makes it possible to run and debug applications directly from Qt creator. It communicates with Qt creator over a serial port, either using USB cable or Bluetooth. The App TRK tool supports important debug fea-

tures such as setting breakpoints, monitoring variables etc. To install the TRK to the phone the following steps are performed

To install TRK:

- Install the TRK to your phone from the link given in the appendix. There are TRK's available for each version of S60. So the TRK corresponding to S60 version 3.1 corresponding to the S60 of the test device was installed. The link provided in the appendix also has TRK installation files for older Symbian/S60 platforms.
- Connect your device in PC Suite mode using a USB cable.
- Launch TRK on your phone. If the connection has been successfully established you should see Status: Connected in the TRK application.
- Make sure that the connection mode is USB, by going to options -> settings-> connection and changing it to USB.

Now change the "Run configuration" of the developed project to enable the application to run on the device.

- Select the projects icon in the left panel.
- Select the Projects icon in the Qt Creator sidebar. In the Run Settings section, press Add and choose the Project Name in Symbian Device option. This should create a run configuration for building a Symbian installation file (SIS file) and deploying it to a device. For deploying to a device the application should be compiled using GCCE.
- Set the new run configuration active: you can do this by selecting the blue link to make it active (below the newly created configuration) or by selecting it in the Active Build and Run Configurations are at the top of the project settings.

Now we can deploy the application by pressing the Run button. This will cause the application to be built and launched. The output of the Application out put tab will look as Appendix D. Effectively the following operations are being formed for us by the Qt Creator which are

- Running qmake and make release-gcce tools to create mplayer.exe
- Running make sis tool to create mplayer_gcce_udeb.sis
- Running signsis tool with a self signed certificate to create create mplayer_gcce_udeb.sisx file.
- Copying this mplayer_gcce_udeb.sisx file to the device
- Installing the mplayer_gcce_udeb.sisx file
- Starting the application

An advanced installer from Nokia called the Nokia Smart Installer is now available. When the user now installs his own sis file on the phone, the Smart Installer will go on-line and get all the dependencies that his Qt application requires, typically Qt and QtWebkit + Open C. If these packages are already installed on the phone, the Smart Installer does nothing. So, it is a little bit like an “apt-get for Symbian”. Snapshots of the application on the device can be seen in Appendix B. First select an mp3 file to be played using the file browser of the application. For the first time it would be needed to browse to the location of your choice. But the path is then cached by Qt and next time when you want to select a song the path will be visible in the file browser. It is possible to select more than one song using the shift key of the phone. After selecting press the play button and the song begins to play. It is possible to seek a particular position in the song play back using the seek slider. The Volume Slider can be used to adjust the volume of the file being played and by default it is at maximum.

5 ANALYSIS

In this chapter the developed application will be analyzed based on the objectives defined in section 2. The design and implementation analysis includes experiences gathered during application development and real device based testing. We will divide this section into 2 parts – how Qt was ported on to the Symbian framework and how the Qt audio framework, in particular, was ported to the Symbian framework.

5.1 Qt Porting to the Symbian Framework.

The Qt port for Symbian uses the POSIX APIs provided by Open C .Qt for Symbian GUI applications are statically linked against S60Main library as shown in the figure below. S60Main implements the Symbian OS entry point E32Main() for Qt for Symbian applications. This is similar to the Open C libcrt0 in a sense that it initializes the environment and calls the standard main() entry point used by Qt. The difference between the libcrt0 and S60Main is that the S60Main initializes the standard Symbian application UI framework whereas libcrt0 only initializes the services needed by non-GUI applications. In essence it means that whenever the 'main' entry point of Qt GUI application gets called, the standard Symbian application UI framework classes such as CAknApplication, CAknDocument and CAknAppUI are already instantiated. This further means that control environment (CCoeEnv) and active scheduler (CActiveScheduler) for the main thread also exist. The control environment availability makes it possible to integrate the top-level QWidgets to native controls and it enables Qt widgets to utilize the native input methods also known as Front End Processors (FEPs). Note that the non-GUI Qt applications are still linked against the Open C libcrt0 instead of S60Main.

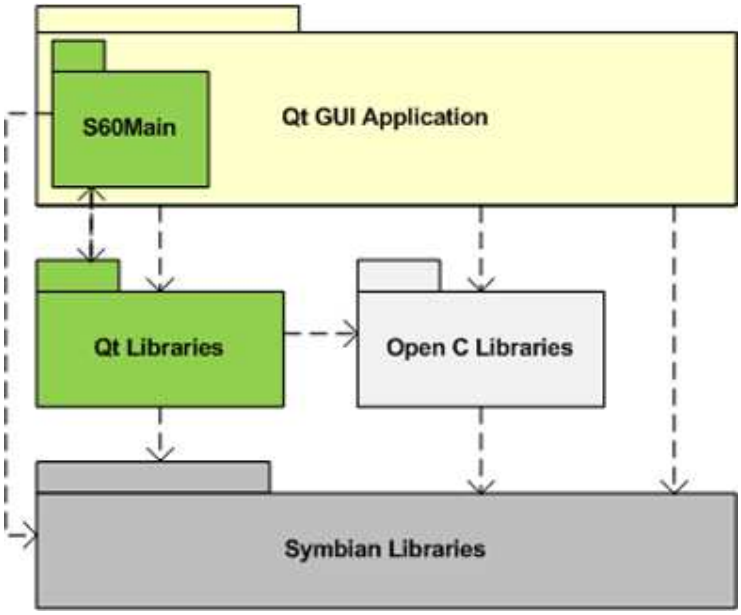


Figure 7. Qt for Symbian architecture [11]

Some modules of Qt had to be rewritten when ported to Symbian due to various reasons. QtCore implements process handling, the file system watcher and the event dispatcher in a Symbian specific way. Due to the limitations of the Open C libraries the process handler had to be written again. Also due to performance issues the file system watcher had to be rewritten. The Symbian active scheduler and POSIX based event dispatchers have been included in QT by rewriting the Qt event dispatcher. Qt core uses the Symbian OS file server library which means that the native Symbian file system is being used by the Qt libraries for Symbian. Qt supports threading by linking against the Open C thread library. Qt has its own network interface implemented for Symbian because of limited support in Open C. Qt uses the insock and esock Symbian OS libraries to implement its network interface adaptation. Libssl and libcrypto libraries from Open C are used to implement the SSL support in Qt for Symbian.

Qt GUI has been implemented differently in the case of Symbian. QtGui module has dependencies to the following Symbian OS libraries: bafl, estor, fntstr, ecom, aknicon, aknskins, aknskinsrv, fontutils, fepbase, directorylocalizer, efsrv, fbscli, bitgdi, hal, gdi, ws32, apgrfx, cone, eikcore, sendui, platformenv, commonui, etext, apmime, avkon, eikcoctl. The module also has dependency to the libpthread

OpenC library.

Qt separates the platform specific implementations from its own using private implementations. As a result of this separation the application developer does not see any difference in the APIS used on different platforms. As mentioned in the beginning of this section the Symbian application UI framework is already instantiated in the "Main" entry point. When the application creates an instance of QApplication from the entry point, the object of S60Data class is initialized. This object contains cached pointers to the native UI related objects created by the S60Main. The cached objects are such as RWsSession, RWindowGroup and CCoeAppUI.

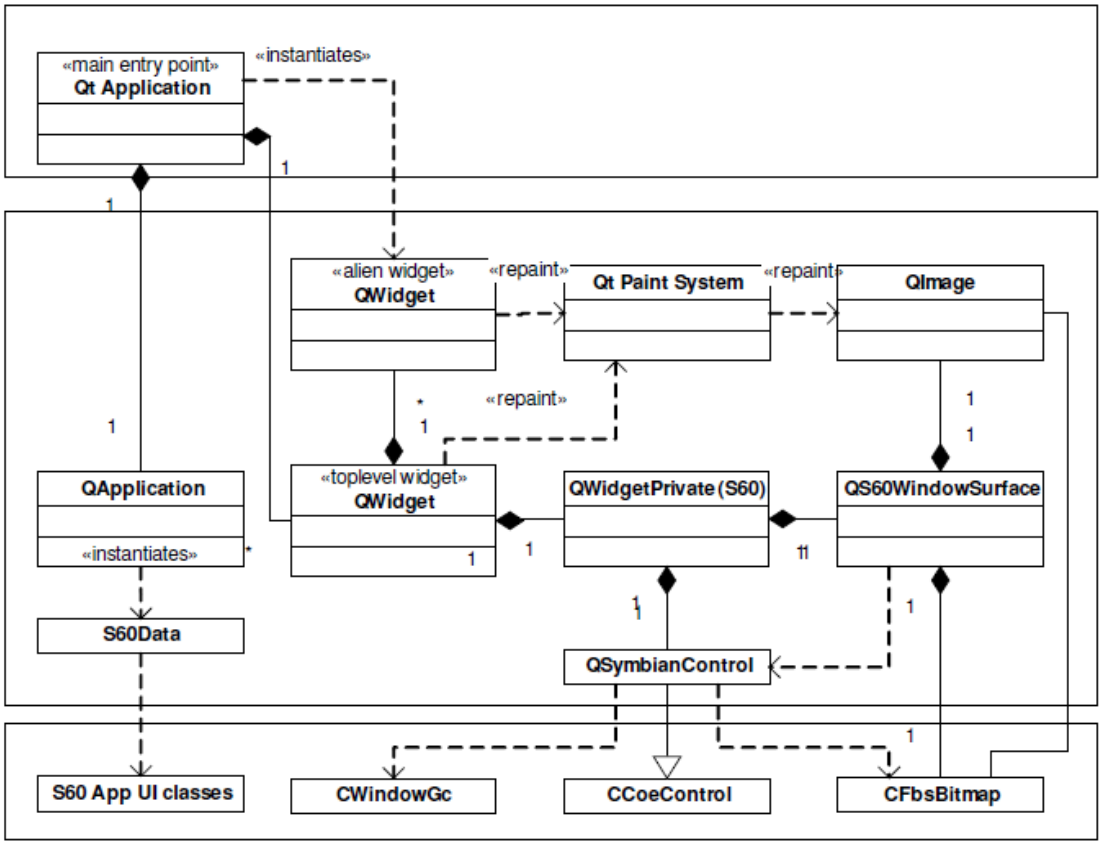


Figure 8. Qt for Symbian GUI Integration [11]

Each top-level widget creates a native control (CCoeControl) by using QSymbianControl. The widget receives different events through this class and forwards them to the correct alien or sub widget. QSymbianControl acts also as a proxy between the native FEP and Qt specific input context. Widgets draw themselves to a Symbian platform-specific widget surface, which essentially is CFbsBitmap but access

to it is provided via QImage interface. In Qt terms the CFbsBitmap represents the paint device of backing store. When necessary, the QS60WindowSurface requests QSymbianControl to paint relevant parts of the backing store to the screen. Because paint device is represented as a CFbsBitmap the drawing is simple bitblit operation to window graphics context (CWindowGC) provided by CCoeControl.

5.1.1 Qt and Symbian Build Integration

Building a Qt for Symbian project is different to building a standard Symbian project. The underlying Symbian tool chain remains the same but the standard Qt build tools are used as a wrapper around the Symbian tools. In other words, a Qt for Symbian application uses the standard Qt project files, i.e. .pro and .prj and is built the same way as a standard Qt application, i.e. using qmake and make.

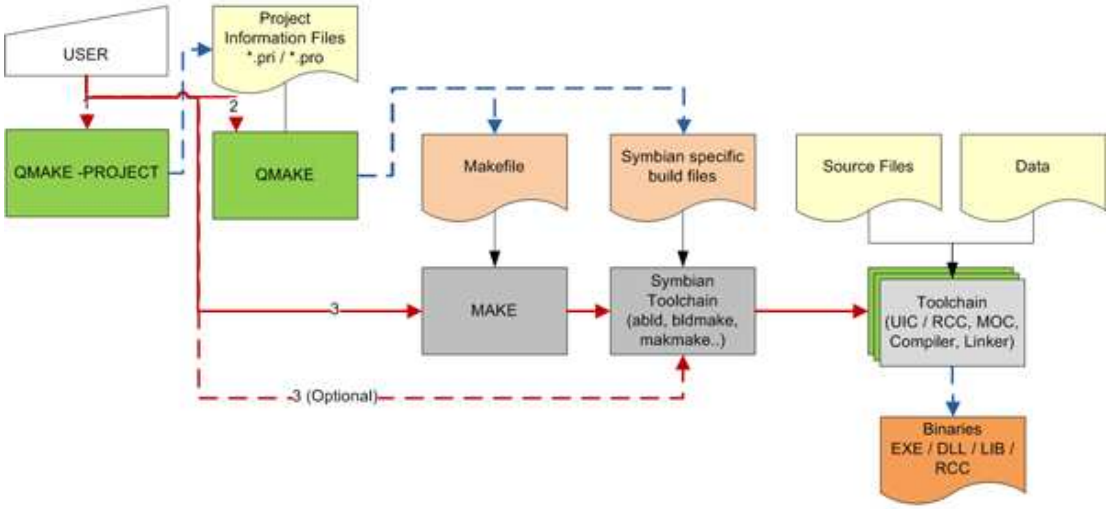


Figure 9. Qt for Symbian Tool Chain [11]

qmake is a tool that helps simplify the build process for development project across different platforms. qmake automates the generation of Makefiles so that only a few lines of information are needed to create each Makefile. qmake can be used for any software project, whether it is written in Qt or not. qmake generates a Makefile based on the information in a project file. Project files are created by the developer, and are usually simple, but more sophisticated project files can be created for complex projects. qmake contains additional features to support development with Qt, automatically including build rules for moc and uic. qmake can also generate projects for Microsoft Visual studio without requiring the developer to

change the project file. Also, a default project file can be created with the qmake –project command. This finds files with known extensions (.h, .cpp, .ui, etc.) from the current directory and lists them in a .pro file.

Based on information in the .pro project file, the qmake command produces the following standard Symbian build items: bld.inf file, .mmp file, default registration file (_reg.rss), extension makefiles (.mk), and package files (.pkg). The extension makefiles are used to integrate Qt-specific tools (i.e. moc, uic, and rcc) with the Symbian toolchain, in a manner similar to the way that mifconv integrates the building of svg icons with the Symbian build chain. A Makefile is also generated by qmake. This works as a wrapper around the standard Symbian build commands, i.e. running the make command calls bldmake and abld to build the Qt application. The Qt for Symbian has support for the new Symbian Build System (SBS v2). However, the new Symbian Build System is only available for Symbian OS 9.5 based SDKs which are not yet available.

5.1.2 Qt on Symbian Memory Management

The Qt classes are implemented in such a way that the cleanupstack is not needed for them. If there is not enough memory, when running the Qt application, the application will simply be closed. When implementing an application with Qt in the Symbian environment, the cleanupstack should be used with Symbian code. The QObject class does not have the initialization of members that the CBase class has. At the moment there are no plans to make a Symbian platform-specific implementation of QObject to provide such a feature.

Qt stores objects into an object tree when they are created. The object tree enables automatic deletion of child objects that have a parent. For example, when a widget is created with another object as a parent, it is added to the parent's child list and deleted automatically when the parent is deleted. If an object with a parent is created with new on the heap, the deletion of the object removes it automatically from the parent. If the deleted object has children, they are automatically deleted when the object is deleted. The same behavior applies to objects created on the

stack. The only objects that have to be explicitly deleted are the objects created with new and that have no parent. A good example of taking care that all objects get deleted when the application is closed is using layouts. If a widget gets the parent information when it is created, adding the widget to a layout changes the parent and the widget is deleted when the layout is deleted. In the following example the parent (this) refers to the widget, which shows the layout and label.

```
QVBoxLayout *layout = new QVBoxLayout;
QLabel *timeLabel = new QLabel("Time", this);
layout->addWidget(timeLabel);
```

If the parent is not in the constructor of the widget, the ownership of the widget is not transferred correctly.

5.1.3 Qt on Symbian Platform Security

Qt is not an interpreted language and so does not have its own security model. Instead Qt applications inherit their security from the underlying platform they are installed on. 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 those API's may need platform security capabilities. Symbian security architecture provides a number of different capabilities, such as access to the phone stack or to the complete file system. To access a system resource, a client program must hold the appropriate capability. In Symbian OS, the 'unit' or base level of protection between any two entities is the process. Thus, under platform security, each process is assigned a set of capabilities. When a process makes a request of another process, the servicing process is able to examine the capabilities of the requestors' process and determine whether the request should proceed. 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 the capabilities and UID are not correctly set for a program that is going be installed in the phone then the package does not work or install correctly.

5.2 Qt Audio Framework Porting to the Symbian Framework.

As we saw in section 2.3, playing of a media file is done by using the Phonon:: Media object class. Phonon has three basic concepts: media objects, sinks, and paths.

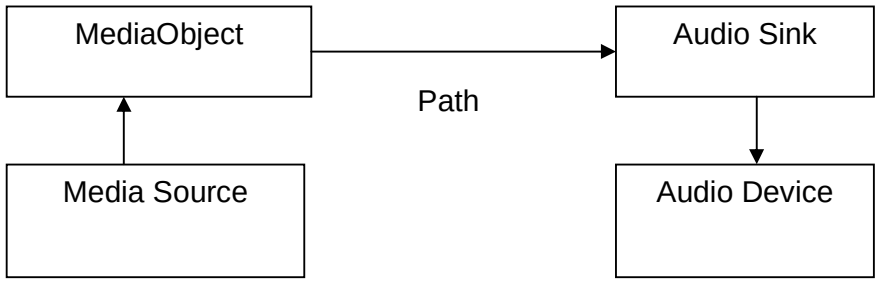


Figure 9. Phonon Architecture [7]

A media object manages a media source, for instance, a music file; it provides simple playback control, such as starting, stopping, and pausing the playback. A sink outputs the media from Phonon, e.g., by rendering video on a widget, or by sending audio to a sound card. Paths are used to connect Phonon objects, i.e., a media object and a sink, in a graph - called a media graph in Phonon. The playback is started and managed by the media object, which sends the media stream to any sinks connected to it by a path. The sink then plays the stream back, usually through a sound card.

Qt provides a backend for Qt's Phonon module, which supports video and sound playback through Symbian's Multimedia Framework, MMF. The audio and video formats that Phonon supports depend on what support the platform provides for MMF. The emulator is known to have limited codec support. The multimedia functionality is not implemented by Phonon itself, but by a back end - often also referred to as an engine. This includes connecting to, managing, and driving the un-

derlying hardware or intermediate technology. For the programmer, this implies that the media nodes, e.g., media objects, processors, and sinks, are produced by the back end. Also, it is responsible for building the graph, i.e., connecting the nodes. The backend's of Qt use the media systems DirectShow (which requires DirectX) on Windows, QuickTime on Mac, and GStreamer on Linux. The functionality provided on the different platforms are dependent on these underlying systems and may vary somewhat, e.g., in the media formats supported.

Backend's expose information about the underlying system. It can tell which media formats are supported, e.g., AVI, mp3, or OGG. The Symbian audio backend is based on the CMdaAudioPlayerUtility and CDrmPlayerUtility API. This API allows audio data to be transferred to the current default audio output device, or from the current default audio input device. For example, these may be the back speaker and on-board microphone respectively. A Symbian device may, however, have more than one audio output or input device available for use at any given time. For example, if a wired headset is plugged in, it may be available for audio output. The platform provides APIs to allow clients (a) to be notified when devices become (un)available and (b) to route audio to / from a given device. At present, the Symbian audio backend does not allow the user to choose audio device; it simply exposes a single QAudioDeviceInfo called "default". The backend should be modified to make use of the Symbian audio routing APIs. The following diagram illustrates the flow of the control for audio playback from Qt to Symbian.

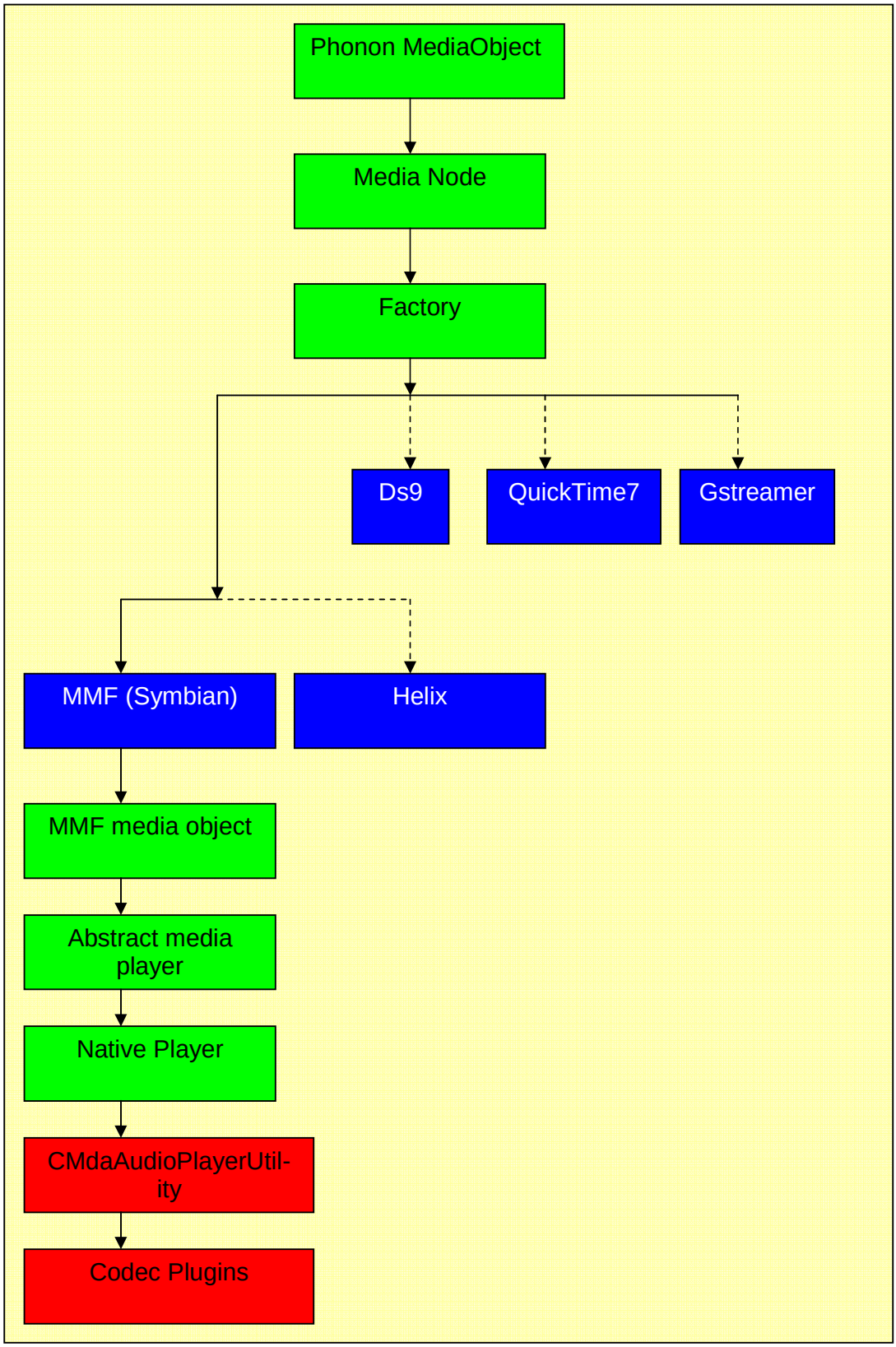


Figure 10. Audio playback on Qt for Symbian

The path of execution of Phonon::Media object class was traced by applying break points using TRK. Qt has a generic implementation of Phonon::Media object class. This implementation calls the corresponding backend object implementation using a helper class to cast the backend object to the correct version of the interface called IFace. Backends are loaded as plugins in to the Qt framework. Plugins are mostly in the form of DLL's which can be loaded at run time. Right now the various backends present are ds9, gstreamer, Symbian mmf and qucktime7. Symbian mmf is the backend used by Qt to play audio on the Symbian device. Plugins are loaded using Qt's QPluginLoader object. On Symbian OS there are two plugins that are supposed to be present, one which uses Symbian MMF framework ("phonon_mmf"), and one which uses Real Networks's Helix("hxphonon"). Helix is preferred because it's more sophisticated. So Helix backend is attempted to be loaded first, and the MMF backend is used for backup. Right now Helix plugin is not being shipped along with QT for Symbian releases.

The Symbian MMF provides the Multimedia capabilities of Symbian OS. These include audio recording/playback, video recording/playback, still image conversion and camera control, where present. Not all of these capabilities are necessarily present but frameworks exist in each case to support them. Ultimately, in some cases, the functionality provided is at the discretion of the licensee. The following figure gives an overview of the Symbian MMF.

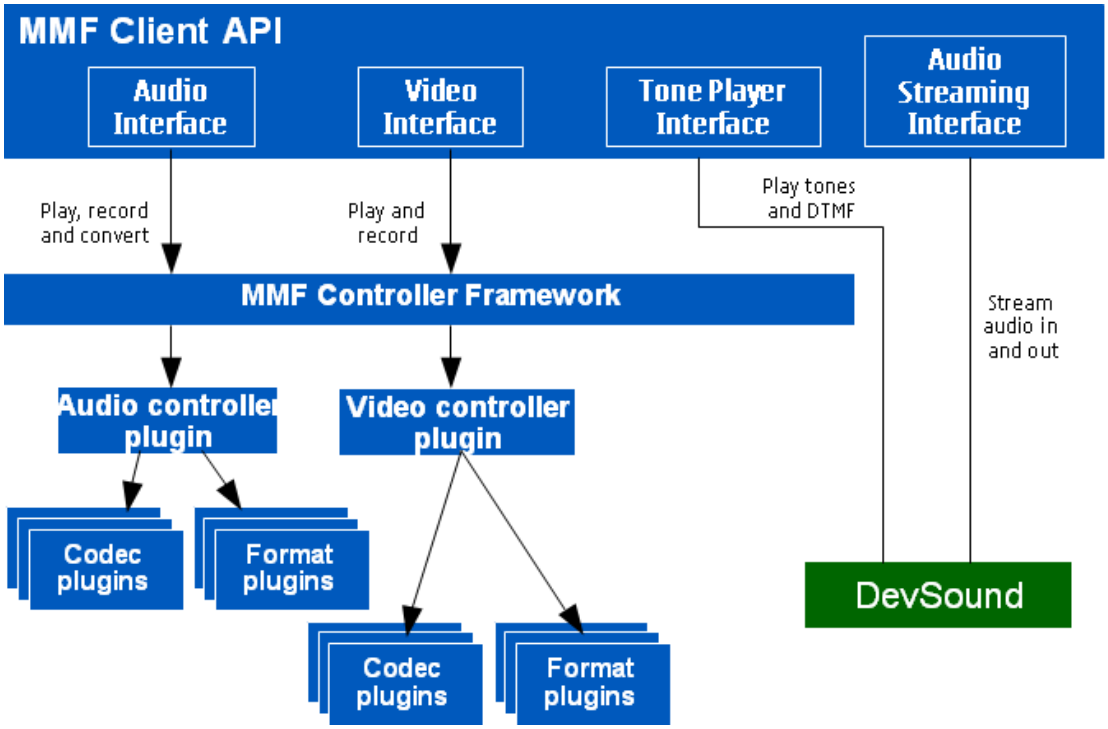


Figure11. Symbian MMF components [17]

The audio interface provides functionality to play back, record and convert audio clips. Supported formats include MP3, WAV and AMR. The MMF controller framework implements some of the client side functionality. It provides the base classes to help licensees implement the plugins. Controller classes implement standard multimedia operations such as play, stop, pause, etc. and can handle multiple formats internally or through format plugins. The source and sink classes act as suppliers and consumers of data. (File, descriptor, and Audio I/O implementations are provided by the framework). Format base classes allow licensees to provide objects that identify and process the format of a multimedia clip and Codec base classes allow licensees to provide objects which perform the conversion of multimedia data in software. The DevSound provides the hardware abstraction layer for digital audio hardware acceleration. It consists of a client API and a plug-in API whereby all requests go through the client API and the client layer takes care of audio policy (managing contention between requests) and data type matching to load the correct plugin. Plugins correspond to a particular accelerated function such as MP3 decoding. The codec plugins are Symbian OS ECom plugins intended for derivation by a specific codec. The derived codec should process source data of a certain datatype and convert it to destination data of a

different data type. An example of a source data type might be say ‘mp3’ and a destination data type ‘pcm16’. This would correspond to a codec that takes mp3 as its source and decodes the data into pcm16.

5.3 Qt Bugs

In the process of the application development a few bugs were discovered in Qt. One bug was in the Qt framework and another was in the Qt Creator

5.3.1 Seek Slider Bug

The Phonon::Media object provides an aboutToFinish() signal which can be used by applications to enque the next file to be played. This signal was not working in my case. After days of debugging the issue was discovered. The issue was that the aboutToFinish() signal will be emitted only if the currently playing file is played completely. During my development the play of the file was being forwarded using the seek slider to the end. This was causing the aboutToFinish() signal to be not emitted. Due to this, it was not able to queue the next file for playback. The issue was reported in the Qt mailing list [14]. It was asked to raise a bug in the Qt bug tracker. Qt bug tracker is the public Qt bug tracking system. There you can track existing bugs in Qt and related products, and can register for an account to report new bugs, discuss existing bugs with Qt developers and make suggestions for Qt feature improvements [15]. The bug Id is QTBUG-9368.

5.3.2 Waiting for App Trk to Start on Port x Bug

APP TRK was used along with Qt to port the experimental application to the device for testing as S60 emulator does not have support for audio codec’s. After the initial setup of the tool it was possible to use the App Trk for many days. But finally one day it showed the following error “Waiting for App Trk to start on Port 20”. Reconnecting the USB, restarting the Trk, restarting the IDE and even restarting the PC didn’t help. In fact the development PC had been used for other development purposes and new devices had been connected to different USB ports in the mean time. After some testing and

looking up in the net the following bug was found - Qt Creator Bug – 568. It says that QtCreator would wait for an AppTRK connection although it already is connected to a higher COM port. So the number of the COM port through which the phone gets connected on my PC was lowered. This can be done by the following steps.

- Go to device Manager of your pc
- Select the COM port on which the phone is connected.
- Go to properties -> port settings -> advanced -> COM port number.
- Assign the COM port to a port below 10 preferably below 4.
- Restart the Qt IDE.

Appendix E provides the screenshot of the port modification window in Windows XP. You will be able to see that all COM ports up to 20 are shown as “In use”. This could be due to devices connected earlier and actually not being used. COM port 4 was assigned which was shown as “In use”. Then restart the Qt creator IDE. This was an important step without which the change would not take effect. When trying to run the Trk again it can be seen that the IDE is using a lower COM port and the IDE is able to copy the sis package to the phone, install it and run it. Appendix B

6 DISCUSSION AND CONCLUSION

As we have seen from this study, portability of Qt on top of Symbian OS was quite feasible due to the fact that Symbian OS had inherent support for Open C. This feature of Symbian will in fact enable it to support much more software on top of it as long as it is Open C compliant. The major modifications that Qt had to make were in the areas of integrating the build systems and GUI modifications.

The phonon port on top of Symbian is still under development. A few bugs were found during the development process. So it will be a while before a mature phonon module will be available. Also as discussed in section 5.2, using Helix as a backend for Symbian will be much more advantageous because of its features. It is expected that the Helix backend will be available in future Qt for Symbian releases.

Implementing applications with Qt is much faster than in s60 due to the various advanced code development tools and inbuilt features in Qt. Some examples would be tools like Qt designer and Qt assistant. It is possible to get the entire documentation of a Qt datatype for example by just pointing the cursor on top the data type and pressing “F1” while using Qt creator which has Qt assistant integrated. Also the use of automatic memory management and multipurpose data types like QVariant reduces a lot of programming efforts. Qt exposes intuitive and high level APIs that address the needs of most developers. The signals and slots mechanism make it easy to connect user actions to application logic, and indeed any objects to arbitrary other objects. Most C++ memory management issues are handled by Qt, leaving application developers to focus on application structure and behavior. Qt applications
follow the native look and feel of the platform. However designers can radically modify the appearance of a user interface, from a single widget to a complete application, using style sheets or custom styles.

Qt itself does not have any features to support platform or device specific features.

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

Since Qt has a long history in desktop usage some of its features are not optimized for mobile usage. In desktop computers there is usually more computation power for graphical output. During the research lack in graphical performance was visible.

The full advantage of platform independence will be available only if all the Qt modules are ported to the Symbian platform. Currently this is a work in progress. In the licensing area Nokia has used the LPL license for the Qt releases which means that companies can build using Qt and release code using an open-source model without having to subscribe to GPL, while also avoiding to become tied into a single company through the terms of Qt's commercial license. This will help to woo more development using Qt. Nokia wants to establish Qt as a framework that spans desktop and mobile application development, while also making it easier to write applications for different mobile devices using a single development framework and code base.

Qt promises to be a very widely accepted platform for application development once its initial bugs are fixed and it gets matured. There is a wide and active developer community which is quite evident from its mailing lists. The success of Qt comes down once again to its easy portability to platforms which we can expect to provide tough competition to the existing GUI frameworks like iPhones and RIM.

7 REFERENCES

[1] Wikipedia (2010) [WWW] Smartphone.

http://en.wikipedia.org/wiki/Smartphone (Accessed Mar 22, 2010)

[2] Wikipedia (2010) [WWW] Symbian OS:

http://en.wikipedia.org/wiki/Symbian_OS (Accessed Mar 22, 2010)

[3] Wikipedia (2010) [WWW] S60(Software Platform):

http://en.wikipedia.org/wiki/S60_(software_platform) (Accessed Mar 22, 2010)

[4] Alan Ezust and Paul Ezust (2009) An Introduction To Design Patterns in C++ With Qt 4 Canada: Prentice Hall

[5] Qt Home (2010) [WWW] Qt on the Symbian Platform

http://qt.nokia.com/products/platform/symbian (Accessed Mar 22, 2010)

[6] Trolltech Documents (2009) [WWW] Qt Reference Documentation

http://doc.trolltech.com/4.6-snapshot/index.html (Accessed Mar 22, 2010)

[7] Trolltech Documents (2009) [WWW] Phonon Overview (2009)

http://doc.trolltech.com/4.6-snapshot/phonon-overview.html (Accessed Mar 22, 2010)

[8] Qt Labs (2009) [WWW] Qt for Symbian and the Nokia Smart Installer

http://labs.trolltech.com/blogs/2010/02/15/qt-for-symbian-and-the-nokia-smart-installer-beta/ (Accessed Mar 22, 2010)

[9] Forum Nokia (2009) [WWW] Qt Tutorial Lesson 1 – Installation

http://wiki.forum.nokia.com/index.php/Qt_Tutorial_Lesson_1:_Installation#Inst

all_Open_C.2FC.2B.2B_Plugin_and_Qt_for_S60 (Accessed Mar 22, 2010)

[10] Youtube (December 2009) Qt for Symbian – developing in Qt Creator

http://www.youtube.com/watch?v=Rb43gnZI1A0 (Accessed Mar 22, 2010)

[11] Symbian Developer (March 2010) [WWW] Qt quick start

http://developer.symbian.org/wiki/index.php/Qt_Quick_Start (Accessed Mar 22, 2010)

[12] Trolltech Documents (2009) Platform Notes - Symbian (2009)

http://doc.trolltech.com/4.6/platform-notes-symbian.html (Accessed Mar 22, 2010)

[13] Jane Sales(2005) Symbian OS Internals West Sussex England: John Wiley & Sons, Ltd

[14] Trolltech Mailing Lists (2008) [WWW] Qt Creator Info page (2008)

http://lists.trolltech.com/mailman/listinfo/qt-creator (Accessed Mar 18, 2010)

[15] System Dashboard [WWW] – Qt Bug Tracker (2009)

http://bugreports.qt.nokia.com/secure/Dashboard.jspa (Accessed Mar 18, 2010)

[16] Jo Stichbury (2005) Symbian OS Explained: Effective C++ programming for Smartphones West Sussex England: John Wiley & Sons, Ltd

[17] Symbian Software Ltd (2006) Symbian OS Library for Device Creators Documentation set version: 9.3/2007-45-00

[18] Symbian Developer [WWW] (2009) A Guide To P.I.P.S (2009) http://developer.symbian.org/wiki/index.php/A_Guide_To_P.I.P.S.

[19] Forum.Nokia.Com [WWW] (2009) The E-learning Curriculum (2010) http://www.forum.nokia.com/Learning_and_Events/E-learning.xhtml

8 APPENDIX A

A.1 Perl Installation Link - http://www.activestate.com/activeperl/

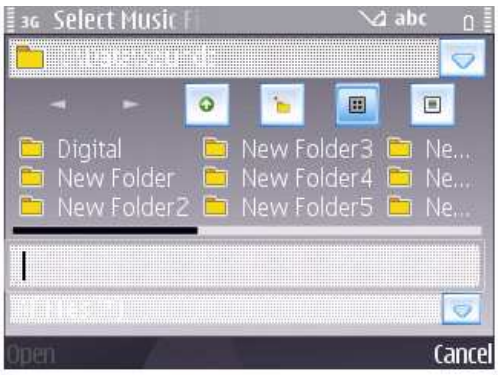
A.2 S60 SDK download link - http://www.forum.nokia.com/Tools_Docs_and_Code/Tools/Platforms/S60_Platform_SDKs/

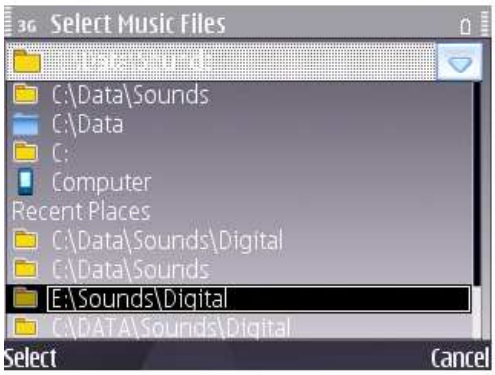
A.3 Open C/C++ download link - http://www.forum.nokia.com/info/sw.nokia.com/id/91d89929-fb8c-4d66-bea0-227e42df9053/Open_C_SDK_Plug-In.html

A.4 Qt for Symbian Windows installer - http://qt.nokia.com/downloads

9 APPENDIX B

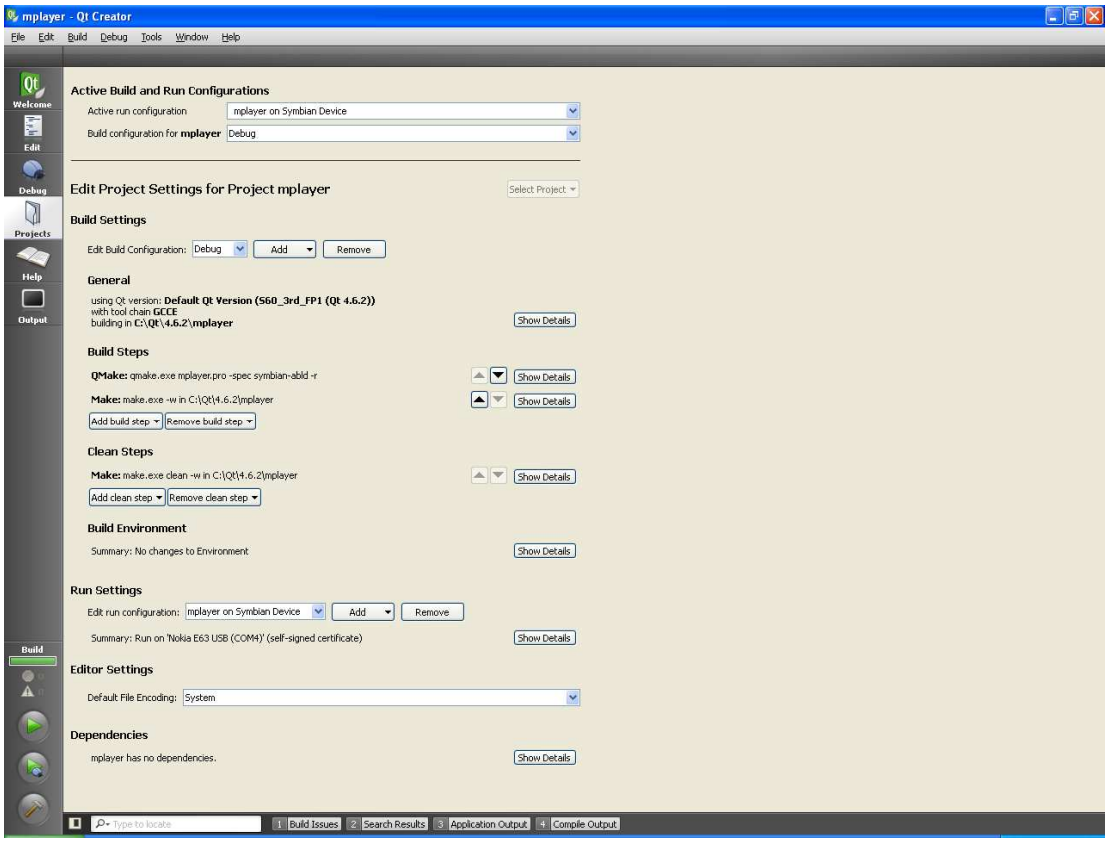

mplayer
Exit


Digital
New Folder
New Folder2
New Folder3
New Folder4
New Folder5
Open
Cancel

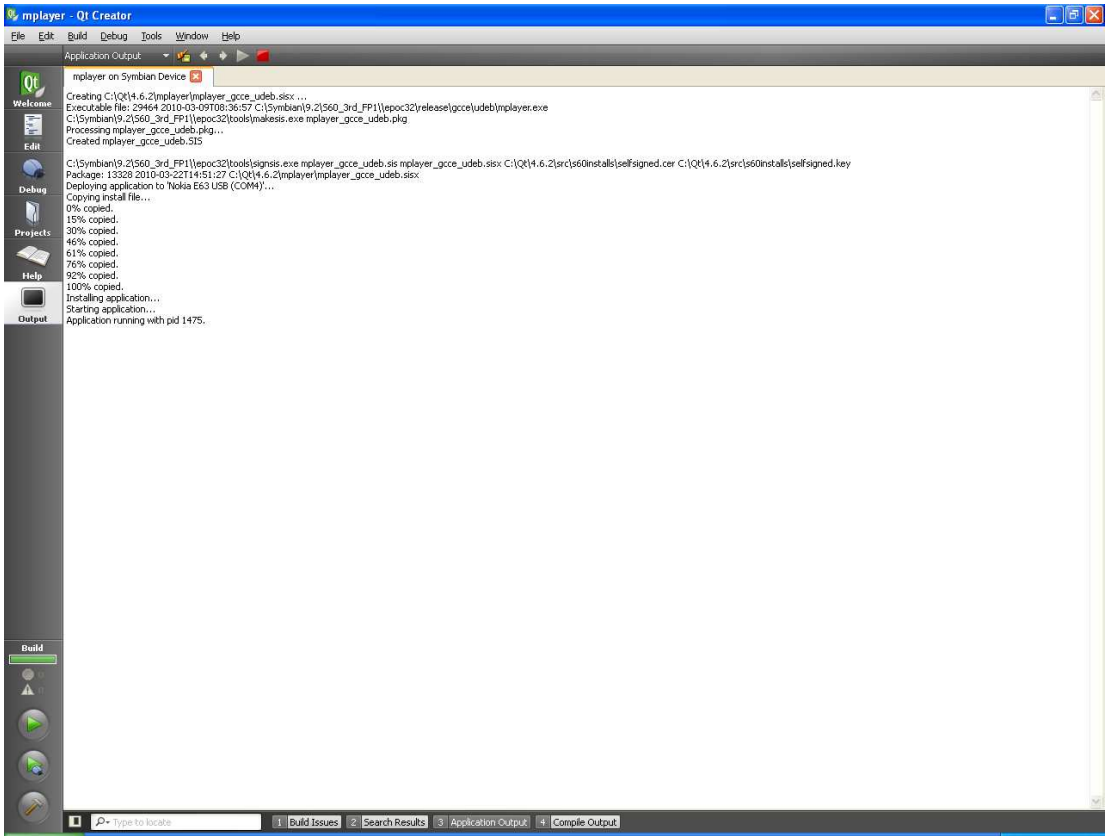

Select Music Files
C:\Data\Sounds
C:\Data
C:
Computer
Recent Places
C:\Data\Sounds\Digital
E:\Sounds\Digital
Select
Cancel


Select Music Files
22 Chris Brown-Forever...
35159_Main Kya Hoon.mp3
35163_Ye Doorian.mp3
22 Chris Brown-Forever.mp3
Done

10 APPENDIX C



11 APPENNDIX D



12 APPENDIX E