Fasil Leulseged

Application Development in Symbian C++ and Qt

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

02/11/2012

Author(s) Title Number of Pages Date	Fasil Leulseged Application development in Symbian C++ and Qt 52 pages 02 November 2012
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Ranne Pasi, Project Manager

The smartphone business landscape has shifted considerably ever since the introduction of the iPhone in 2007. The ensuing years have seen Nokia, the world leader in the smartphone industry at the time, loose grip on market share. Consequently, Nokia has set a new roadmap to revamp the Symbian operating system and regain lost market shares. The roadmap is headlined by two important entries: the adoption of the Qt framework and partnership with Microsoft on Windows Phone 7.

This thesis is concerned with the adoption of the Qt framework into the Nokia smartphone family. More precisely, the thesis discusses how the programming paradigms of Qt differ from the archaically well-established Symbian operating system.

The architectural differences of Qt and Symbian OS have been thoroughly examined using code snippets as supplement to demonstrate implementations.

The findings of this thesis show that not only does Qt make the life of a developer much easier than traditional Symbian OS, but also, and more importantly, Qt delivers industry standard user interfaces and fluid user interactions at par with the competition.

Keywords	Symbian, Qt, Programming, C++

Contents

Abbreviations

OS	Operating System
PLC	Private Limited Company
MB	Mega Byte
GB	Giga Byte
RAM	Random Access Memory
JRE	Java Runtime Environment
IDE	Integrated Development Environment
OEM	Original Equipment Manufacturer
API	Application Programming Interface
SDK	Software Development Kit
GUI	Graphical User Interface
UI	User Interface
MVC	Model-View-Controller
UID	Unique ID
FTP	File Transfer Protocol
TCP/IP	Transfer Control Protocol / Internet Protocol
UDP	User Datagram Protocol
SSL/TLS	Secure Socket Layer/Transport Layer Security
SDP	Service Discovery Protocol
HCI	Host Controller Interface

1 Introduction

Mobile phones have become an indispensible tool in day to day activities. Besides the intrinsic design to make calls, users can check e-mail, surf the web, and connect with friends on social networks on the fly.

Nokia has been the market leader in the mobile phone sector. However, fierce competition from Apple’s iPhone and Google’s Android has seen Nokia loose huge market shares. Nokia has set a new direction to regain the lost market share. The new direction includes the introduction of Qt as a major haul to Nokia’s operating system, the Symbian operating system (OS), and partnership with Microsoft on the Windows phone platform.

The Symbian operating system (OS) is developed by Symbian Limited. The company has its roots in Psion private limited company (PLC). In fact, Psion is considered the precursor of Symbian Limited. Founded by Nokia, Psion and Ericsson in July 1998, Symbian limited had its beginnings in the Psion EPOC software. In fact, the early versions of the operating system that shipped with phones in 1999 carried the EPOC name. Symbian OS version 6.0 based mobile phones were ‘open’, meaning that users were able to install their own software. Symbian OS v6.0 appeared in 2001 and shipped on the Nokia 9210 Communicator but the world’s first Symbian OS phone was the Ericcson R380 which went on sale in 2000. Symbian OS continued to evolve in functionality and market-leading features. In 2004 Symbian shareholders purchased Psion stakes. Consequently, the next big change occurred in early 2005 when Symbian OS version 9.0 was announced. This version was designed to be more secure and deliver major cost savings to manufacturers. [1, 24]

Qt was developed by Trolltech and became publicly available in 1995. During inception, Qt was meant to be a C++ GUI framework but has grown popular to become a cross-platform application and GUI framework. Subsequently, Trolltech released newer versions of Qt but Qt 4.0, released in 2005, was a major step forward for the company and was the first Qt edition to be available for both commercial and open source development. [2]

The appeal of Qt to Nokia, with respect to the Symbian OS, is in C++ compatibility and the much needed GUI framework. This allows Qt to be wrapped around Symbian OS. When Qt integration to the Symbian OS is finalized, developers deal directly with Qt instead of Symbian programming paradigms and the end user will get more dynamic applications and a modern user interface among others – a win-win-win situation for Nokia, developers and end users.

The purpose of this thesis is to point out how the shift to Qt programming paradigm differs from Symbian programming paradigms as seen from the developer point of view.

2 Symbian and Qt Development Environments

2.1 Symbian Development Environment

The Symbian development environment is based on the open Symbian (OS) which is the most widely used operating system for mobile phones. Symbian OS is a highly optimized, pre-emptive, multitasking environment. In order to start development of Symbian applications there are prerequisites to be fulfilled.

The following requirements should be fulfilled to start the software development process:

- An operating system.
- At least 512 megabytes (MB) of random access memory (RAM).
- 1 gigabyte (GB) or faster Pentium-class processor.
- At least 1 GB of free disk space.
- 16-bit color display of 1024 * 768 pixels resolution.
- Java Runtime Environment (JRE).
- ActivePerl version 5.6.1 or later.

An Integrated Development Environment (IDE) provides with an organized way of developing applications. Most programmers use an IDE because of the comprehensive facilities that encapsulate a source code editor, a compiler, automation tools and a debugger. There are some IDEs available for developers: CodeWarrior, Visual C++, and Carbide C++. The recommended IDE is Carbide C++ which is based on the Eclipse platform. Carbide C++ has a commercial version and a free version. There are different flavors or editions of the commercial edition of Carbide C++. The free edition is called the Express edition and the commercial ones are Developer edition, Professional edition and Original Equipment Manufacturers (OEM) edition. [1, 1 – 7]

The free edition of Carbide C++ can be downloaded and installed separately or as part of Application Developer’s toolkit (ADT). Either way, one has to make sure the compiler version is compatible with Carbide’s requirements which usually dictate the compiler must be at least version 3.2.5, build 487 to be able to accommodate some functionality such as Qt support .[3]

A Software Development Kit (SDK) is the platform which provides the track on top of which all applications for a specific target are developed using the provided Application

Programming Interface (API).When developing an application for a specific target, like Symbian, the first step is to acquire the SDK for the target. Depending on the need, there are three different SDKs for the Symbian platform: S60, UIQ and MOAP. Each SDK in turn has different versions.S60 versions are referred to as feature packs. S60 is one of the leading smartphone platforms and is the SDK of interest in this thesis.
There are different versions of the S60 SDKs to choose from:

- S60 1st Edition SDK for Symbian OS
- S60 2nd Edition SDK for Symbian OS
- S60 3rd Edition SDK for Symbian OS
- S60 5th Edition SDK for Symbian OS
- Nokia N97 SDK for Symbian OS

Each of these SDKs might have versions of their own usually referred to as feature packs. An example would be S60 3rd edition feature pack 1 and S60 3rd edition feature pack 2. After the SDK installation the binaries and libraries are located in the following directory of the installation folder:

1 - Emulator drives	…/epoc32/winscw
2 - Emulator binaries	…/epoc32/release/winscw/udeb
3 - ARM libraries	…/epoc32/release/gcce/urel

To render the developed application usable, compilers transform the source code into an executable file. Usually, these compilers come bundled with the SDK installer, true at least in the S60 SDK case.

There are two targets in mind when compiling for the Symbian C++ environment. The first target is compiling for an emulator. An emulator mimics the actual device and displays the output of a project as it would appear on device. This allows for an efficient way of debugging code before the software is deployed to device.

The compiler responsible for the Symbian OS emulator for Microsoft Windows is the Nokia x86 compiler which is usually referenced as WINSCW. The second target is compiling for the actual device. This is done using the open source compiler known as GCCE. Both WINSCW and GCCE are bundled with the SDK: they are installed with the SDK and there is no need to download and install them separately. [3]

2.1.1 Carbide C++ IDE

The IDE is organized in such a way that the project files are located on the left screen, file view on the right and error messages and related items on the bottom.

In order to start working on a particular program, the first step is to create a project from the File menu or alternately from the toolbar. After choosing the Symbian OS C++ Project from the menu or toolbar, one has the option to choose from an array of Generic Symbian OS, S60 and tutorial templates, and then proceed to assign a project name and workspace location and selection of the preferred SDK if more than one SDK is present on the system. After these steps are completed, a project with the given name will be created along with all necessary files that can be viewed and modified. These files are located on the left side of the IDE screen.

2.1.2 Carbide C++ Project Files

The project files are inside their respective folders and careful understanding of these files is crucial and in most cases a big time saver in future project endeavors.

- Includes Folder - This is the master ‘includes’ folder with a global scope (SDKs header files …/epoc32/include) and local scope (project header files …/project/inc) as well.
- Data Folder - Contains resource definition files necessary for creation of menus and other UI components. The .rss extension files are resource files and the .rls extension files are localized string files. String localization is not done in the resource file itself.
- Inc Folder - Contains a .hrh extension file which is a common header file for resources and C++ programs which define the commands to be used in the resource files. Also located in this folder are standard C++ header files and a .pan extension file which contains the panic codes for the application.
- Src Folder - Contains the project source files with file extension .cpp.
- Doc Folder - The graphic files used to create icons reside in this folder.
- Gfx Folder - Contains an XML file associated with graphics.
- Group Folder - This folder contains the files necessary to build the project application. The bld.inf file provides the compiler with information for building the project, for instance what project to build, where the project is located and what project specification to include in the build. The projectname.mmp file is the project specification file.

• Sis Folder - Contains the package file used to create an installation file. The package file (projectname.pkg) contains the files and resources necessary to make an installation file with the .sis or .sisx extensions. When the package file is compiled using the IDE or command line (makesis.exe), an installation file (projectname.sis) is created which can be deployed to a device.

2.2 Qt Development Environment

Qt is the de facto C++ user interface development framework which is a cross platform application framework. Applications are written once and can be deployed across desktop and operating systems without rewriting the source code. Qt applications are developed in standard C++ as opposed to Symbian which uses the Symbian C++ dialect. [14]

Qt application development for the Symbian platform is still in its infancy. Until recently, the Qt framework, Qt IDE, Symbian SDK and other required tools were installed separately to start Qt development. In June of 2010, Nokia released the all in one Nokia Qt SDK which contains everything required for development of Qt mobile applications on a compatible Nokia device. So as it stands, there are two ways to start developing Qt mobile applications: using the Nokia Qt SDK and using a standalone SDK.

2.2.1 Qt Mobile Application Development Using Nokia Qt SDK

Starting off with Nokia Qt SDK is the easiest way. Everything that is required to start development is bundled in one installation file. There is also an online installation option which eliminates the need to download the installation file.

The following are required by the system:

• Windows XP service pack 2 or Windows Vista or Windows 7 are the supported Windows operating systems.
• Approximately 4 GB of free disk space.

There is no need to install the Symbian S60 SDK separately as it is bundled with the Nokia Qt SDK installation file.

The native Qt IDE is called Qt Creator. It can be used to develop, build and deploy applications to devices. Applications can also be deployed to the Qt Simulator.

Alternately, Carbide C++ IDE can be used since it supports Qt provided the Carbide compiler version is at least 3.2.5 build 487.

Qt Creator is installed as part of the Nokia Qt SDK along with Qt Designer, Qt Assistant and Qt Linguist.

Qt Designer is a very useful UI creation tool which is integrated with Qt Creator. The Qt user interface can be created either with raw coding in Qt creator or with the simple and easy to use Qt Designer. Alternately, QtQuick can be used for rapid UI implementation.

Qt Assistant provides help with the Qt API documentation. Qt namespaces or classes can be navigated with this tool.

Qt Linguist is the IDE which provides Qt application translation; once an application is developed, it can be translated to the required language and deployed.

In order to deploy Qt applications to a device and make them functional, Qt binaries should be installed on the target device first. The required and optional binaries are listed below.

- Qt binaries - This is the main Qt binary.
- Qt Mobility binaries - Qt Mobility is a recent API release by Nokia which allows developers to use mobile features like messaging, multimedia and sensors to name a few. The term Mobility to Qt is what Capability is to Symbian.
- Qt SQLite - This is an optional binary install which can be deployed to a device if the developer intends to integrate SQLite database into applications.
- TRK - Allows for on-device debugging with Qt creator. TRK needs to be started first before every debug session. TRK is also needed to achieve the same debugging on Carbide C++.

2.2.2 Qt Mobile Application Development Using Standalone SDKs

Setting up a development environment using standalone SDKs is a fairly complicated process when compared to the Nokia Qt SDK alternate. Unless there is a need to use or mix Symbian C++ code, Nokia Qt SDK is the easiest and most natural way to go.

The following prerequisite must be fulfilled to get up and started.

- • Windows XP service pack 2 or Windows Vista or Windows 7 are the supported Windows operating systems.
- • Approximately 4 GB of free disk space.
- • JRE 1.6.0_xx
- • Active Perl 5.6.1 build 635
- • ADT (Application Developer Kit) – this kit includes the Carbide C++ IDE and other required tools.
- • Open C and Open C++ plug in – Qt is dependent on Open C/C++ to work. This compatibility layer should be downloaded and installed. During installation it is crucial to choose the target S60 SDK. If there are multiple SDKs, this plug-in should be installed for each SDK.
- • Compiler compatibility is necessary. A Windows compiler patch should be installed in the x86Build directory of the ADT path.

The target S60 SDK should be downloaded and installed. Qt for Symbian should be downloaded and installed. The latest version is Qt for Symbian by Nokia version 4.6.3. Qt must be installed on the same drive as the target S60 SDK and the Qt install path must not contain any spaces or it will not work.

Qt Creator is installed with Qt for Symbian while Carbide C++ IDE is installed with ADT.

Qt Mobility must be downloaded and installed since it is for some reason not included with the standalone version of Qt. The install text file can be used as a guide to properly compile and configure Qt mobility.

The easy way to install Qt binaries is through Nokia PC Suite or Nokia OVI suite. If either of these applications is installed on the PC, binaries can be installed from Start/Nokia Qt SDK/Symbian menu of the PC. Otherwise, the device should be connected in USB storage mode and the binary installation files copied to the device and launched individually from the device’s file browser. The binaries to be installed are:

- • Qt binary - this is the main Qt binary.

• Qt Mobility binaries - Qt Mobility is a recent API release by Nokia which allows developers to use mobile features like messaging, multimedia and sensors to name a few. The term Mobility to Qt is what Capability to Symbian is.
• Qt SQLite - This is an optional binary install which can be installed to a device if the developer intends to integrate SQLite database into applications.
• TRK - Allows for on-device debugging with Qt Creator. TRK needs to be started first before every debug session. TRK is also needed to achieve the same debugging method on Carbide C++.

2.2.3 Qt Creator IDE

Qt Creator is the preferred IDE of application development. The IDE is organized in such a way that on the far left are switching buttons that make switching from editing mode to design mode and also located here are the debug button which opens the debug window on the right bottom of the screen and also the project and help buttons which allow for viewing the project's build/run configurations and help manuals respectively.

In edit mode, the project files are listed in their respective folders on the left side of the screen immediately next to the switch mode buttons and the individual files can viewed on double click in the viewing area located on the right side of the screen. Most areas on the IDE screen can be resized according to need.

One should check if both Qt for Symbian and Nokia Qt SDK are detected by Qt creator using the menu Tools/Options and then choosing the Qt 4 tab. If all goes well Qt for Symbian and the S60 SDK should be listed under the Auto-detected column of the Qt Versions Tab. Also located in the same tab under the title Manual should be the Nokia Qt SDK.

A project is created from the 'file > new file / Project' menu. This yields an array of templates to choose from. Choosing Qt C++ Project on the left side of the dialogue will give three options: Qt Gui Application, Mobile Qt Application and Qt Console Application. For mobile device development, the Mobile Qt Application option is the right choice. After making the choice, one has to provide the Project name and project location. The next dialogue reflects Qt's motto, "Code once, deploy everywhere". In this dialogue the various available platforms are listed; from Qt simulator to Standalone Qt for Symbian, Nokia Qt SDK and Maemo platform. Of course, all of these targets have to be installed to be accessed. After choosing the targets in need from this dialogue, on clicking next, the class information dialogue appears. The developer should provide a class name, a base class: QMainWindow, QWidget or QDialogue, a header file name, a source file name and a form file name. Qt creator automatically corrects the aforementioned file-names when a class name is provided. Qt creator then creates the project and the files according to the information provided.

2.2.4 Qt Creator Project Files

The following is a list of the Qt project files:

- Sources Folder - This folder contains the source files of the project. Usually, it will contain a classname.cpp and main.cpp files.
- Headers Folder - The project header files reside here. The header and source files are like standard C++ files.
- Forms Folder - This folder contains the classname.ui file. This file is edited only in the design mode. Design mode is the easiest method to create a user interface. Alternately, UI design can be done in code. There are rich UI building tools located on the left side of the screen of the Qt Designer ranging from layouts, spacers, buttons, views, widgets and containers. Designing a UI is as easy as dragging and dropping the required item. UI components developed with Qt Designer can also be referred to in code using a pointer to the UI and subsequently pointing to the desired UI component.
- classname.pro - This is the file used to build the project. It contains items that are included in building the project. The following is a list of items included in a typical .pro file. The sign += indicates the item after the equal sign will be included in building the project.
    - QT+= core gui - This shows QtCore and QtGui components of Qt will be included.
    - TARGET+= project name – This specifies the project name as target.
    - TEMPLATE += app – This shows the project is an application.
    - SOURCES+= main.cpp\ Classname.cpp – This specifies source files.
    - HEADERS+=classname.h – This specifies header file(s).
    - FORMS +=classname.ui – This specifies the form file.
    - CONFIG += mobility – This specifies mobility inclusion.
    - MOBILITY = ______ – This specifies a particular mobility to be added (example: sensors).

The details of setting up Carbide C++ for development are discussed in section 2.1. Nonetheless, it is important to make sure the proper settings are in check. First, one has to make sure that the required S60 SDK is selected. This is done via the ‘window>preferences’ menu, expanding Carbide C++ and clicking on SDK preferences section. Here if the S60 SDK has not already been selected or if there are multiple S60 SDKs, the preferred SDK(s) can be added. Also, the Qt preferences section is located in the same preferences window. The desired Qt version is selected here.

Mobile application development environment set up can be a complex process in general; the developer has to go through hoops to get the environment up and running. Symbian application development environment setup is one such instance; however, recent SDK re-releases incorporate most tools needed for the development environment setup. Nokia Qt SDK eliminates the tedious setup process by providing a single installer that contains everything a developer needs to start application development.

3 Application Development

The main purpose of this project was to show how the programming paradigm of Qt and Symbian differ and in the process show how applications can be developed using the two paradigms. To achieve that, a rich text document application was developed called fWord. fWord works much like any word processing application such as Microsoft word. It is developed both in the Qt framework and the Symbian platform. Since the emphasis of the project was to explain both Qt and Symbian, most of this final year project's time was allocated to explain the two programming paradigms rather than the project application. fWord was used as an illustrative tool. Nevertheless, fWord explains the core concepts sufficiently. It contains sample code components provided by Nokia. The application was designed for Symbian S60 5th edition devices.

3.1 Classes

3.1.1 Symbian Classes

There are several classes with a prefix that define the purpose of a class in Symbian; Symbian classes are prefixed with a letter which stands for a particular purpose: there are four such prefixed classes. [5, 30]

T Classes: T classes are similar to type definitions in standard C++. In fact, the T stands for type. This class does not have a destructor which makes object ownership by T classes a bad idea since ownership implicates a responsibility to delete the object from memory, the result being T classes not owning pointers, references or handles. [5, 32]

Lack of a destructor makes object creation on the heap complicated. The easiest way is to create the object on the stack but if the need arises to create a T class instance on the heap, it has to be pushed to the stack before a potential leave operation. [5, 32]

C Classes: C classes are derived from the CBase class. CBase derived classes have a virtual destructor which ensures object deletion in derived classes in order. Moreover, objects instantiated from the CBase class are zero initialized with an overloaded new operator; this means there is no need to initialize object members individually. Zero initialization works only on the heap because the new operator does not apply for the stack. Hence, C class objects must be initialized on the heap. [5, 34]

R Classes: The R stands for resource. There are no base classes associated with the R class, as it usually refers to an external resource. A constructor must be made to initialize the resource handle to zero and subsequently, resource functions must be developed to allocate and handle opening, closing, resetting or initializing the resource. R classes are usually small in size and contain mostly the resource handle. [5, 35]

M Classes: The M stands for Mix-in. An M class is an abstract interface class; there is usually a main class which serves as the first base class and one or more M class interfaces which extend the functionality. M classes are usually used to define observer classes or callback interfaces. Strict control is required when using M classes due to the complexity arising in the use of multiple classes which have their own peculiar behavior. [5, 39]

Besides T, C, R and M classes, the Symbian OS has several static classes without a prefix used as utility classes. These non-prefixed classes are not directly available but can be invoked with the scope operator (::) using the corresponding static member functions.

3.1.2 Qt Classes

There is a fundamental paradigm shift in class convention and object creation in Qt.
There are no discrete prefixed classes, there is no cleanup stack, there are no two phase constructions and most of the memory management is done by Qt itself.
There is only one base class for all Qt objects, which is the QObject.

The main feature of the Qt object model is the use of signals and slots.
Signals and slots allow for an object to communicate with each other by connecting an object’s signal with another object’s slot [6].

The other feature is object trees; objects are organized in a tree form in order of creation. When child objects are created, they are added to the parent tree and if in turn those children create objects of their own, the object tree extends further down. This makes memory management, which has a paramount place in mobile software engineering, seamless in Qt. When a parent object is deleted, all child objects will be deleted behind the scene; there is no need to delete child objects programmatically. Child objects can also be deleted individually. When this is the case, the parent object tree reflects the deletion by removing the child objects from the tree. [7]

Qt's meta-object system contains detailed information for objects inherited from QObject.

An object's class name can be retrieved using this system among others. The meta-object system is a prerequisite for the Signal and Slots mechanism. QObjects can receive events that they are interested in and QObjects also provide basic timer support. The Q_OBJECT macro must be used in order to implement any Signal and Slot functionality. By design, QObject has neither a copy constructor nor an assignment operator. Signals and slots are automatically connected by Qt's meta-object system. [6]

Qt has a simplistic and intuitive way of class and object implementation whereas Symbian has a complex set of classes and objects that follow. One point to keep in mind is that Symbian is designed from the ground up with mobile device resource limitations (memory, power) in mind while Qt is designed to be a GUI framework. Of course, Qt has a variety of classes but they are not intrinsic to Qt's core architecture. Regardless, Qt offers an easy class implementation.

3.2 Objects and Memory Management

3.2.1 Symbian Objects

Mobile phones have a limited memory when compared with personal computers. Therefore, application development on a mobile phone is memory critical and Symbian is no exception. The Symbian OS implements a fairly complicated mechanism for object instantiation and memory management; the cleanup stack and the two-phase construction provide such a mechanism, and they are integral parts of the Symbian OS architecture.

The cleanup stack is a data structure which stores pointers to objects for safe destruction later in an event of a leave. If a class owns a pointer, then the pointer must be pushed to the cleanup stack except for class members or sub-objects. In Symbian terminology, a leave is an exception that might arise in code and cause an error. Leaves are analogous to standard C++ exceptions. Standard C++ handles exceptions by nesting code with the try, catch and throw statement. This mechanism was not adopted by the Symbian OS. Rather, the concept of a leave was developed which is also an integral part of the Symbian OS. Potentially leaving functions are denoted with an L suffix. [5, 58]

C class derived objects can describe the role of a cleanup stack very well. C class objects are always created on the heap because they are created using the new operator. If a local variable is a pointer which points to such a C class object, in the event of a leave, the local variable will be destroyed without properly freeing the heap memory the C class object was occupying. This causes a memory leak. An alternate way to avoid such a memory leak is to use trap harnesses which essentially set a trap around the code. Encapsulating every function with such a trap harness, ironically, has an impact on memory; excess usage of traps will increase the size of the end product (application) which is deployed to a mobile phone’s memory. The cleanup stack was introduced to alleviate the problem of memory leaks as described above. The idea behind it is to push objects that are not leave-safe to the cleanup stack before calling any potentially leaving function. If a leave occurs when the potentially leaving function is called, the memory space the object was occupying can be freed by retrieving and destroying the object from the cleanup stack where it was stored. [5, 58-64]

Objects that are not leave safe and which are created on the heap with the new operator can be pushed to the stack. However, what if such an object contains a sub-object in its constructor? The main object can be pushed to the cleanup stack but the sub-object might leave at any point and there is no mechanism that allows for putting the sub-object on the cleanup stack or by design, calling the destructor from within the constructor. Therefore, as a general rule, no code in the constructor should leave. Nevertheless, this rule is not guaranteed to be followed since it ties the hand of the developer. [5, 78]

The two phase construction was introduced as a solution to the above mentioned complex problem. In the first phase, the focus is on creating a constructor that cannot leave. In the second phase, the actual memory allocation for sub objects is done with the ConstructL function which can leave. The following code snippet illustrates the two phase construction. The numbers on the left side are for explanatory purposes. This numbering method will be used throughout this paper.

```
CRTEDocument* CRTEDocument::NewL(CEikApplication& aApp)
 {
  1      CRTEDocument* self = new (ELeave) CRTEDocument( aApp );
  2      CleanupStack::PushL( self );
  3      self->ConstructL();
  4      CleanupStack::Pop();
 5             return self;
 }
```

Listing 1. Symbian two-phase construction

Line 1 is the first phase constructor where the constructor is created with the new (ELeave) operator. The new (ELeave) operator is similar to the new operator except it will call the static function User::Leave() if there is insufficient memory to allocate the object. User::Leave() is used to terminate the current function immediately. The first phase constructor must not leave and the sub-object should not be initialized here. Line 2 shows the object created with the new operator pushed to the stack. At this point, there is a constructor that is designed not to leave and an object in the cleanup stack. Line 3 is the second phase constructor where sub-objects that can leave are initialized within the ConstructL function. Lines 4 and 5 show after ConstructL is called that it is safe to remove the object from the cleanup stack and return it.

To avoid the laborious task of calling ConstructL function every time an object is created, the static methods NewL or NewLC are used.

The cleanup stack and the two phase construction are at the core of object creation, leave compartmentalization, memory leak management and sub-object management. The two concepts are the most common idioms in Symbian C++ programming.

3.2.2 Qt Objects

The Qt object model provides a way for object ownership management in a simplified manner; the owner implies a “responsibility for deletion”, usually when a destructor is called [8]. The Qt object model is based on the object from which most Qt objects are derived, QObject. Objects derived from QObject can have one parent object at most and a container for children which are also of type QObject [9]. Each parent stores pointers to its children in QObjectList; the QObjectList class stores pointers to objects in a list, QPtrList, which is a template class that provides a list [9]. To avoid double deletion, if a child is deleted before the parent, it is automatically removed from the parent's list of children [9].

Qt does not force an object to be part of the object tree. Objects can be declared on the stack as automatic variables or alternately, as heap based objects that are manually owned and to be deleted by the parent in the standard C++ way. [8]

Each QObject child can have a large collection of children: This is one of the reasons why a copy constructor is not allowed in QObject objects [8]. A copy constructor in C++ provides with a deep copy of an object. Allowing copy constructors would have a costly consequence in memory management in Qt objects derived from QObjects.

Unlike the Symbian programming paradigm of using the two phase construction to avoid initializing sub-objects on construction, objects can be added to other objects on construction in Qt.

3.2.3 Signals and Slots

Signals and slots offer an intuitive and easy way of communication between objects. An object emits a signal when a certain event occurs: a click of a button, a menu selection, or when a timer expires. This signal can be connected to the slot of another object; when an object triggers a signal, another object's slot will be executed. Any two connected objects' signal and slot are not mutually exclusive; a signal can also be connected to any number of slots in any kind of objects. All connected slots are executed when a signal is emitted. [8]

An object emitting a signal does not have any prior knowledge of or interest in the slots receiving the signal. Moreover, the signal slot mechanism is type safe: a signal may have more arguments than the connected slot but the signal's signature must be matched allowing the slot to ignore extra arguments. [8]

The following code snippet demonstrates how to derive objects from QObjects, the parent and child paradigm and the concept of signals and slots.

```
1 class MainWindow : public QMainWindow
 {
  2   Q_OBJECT
  public:
 3     MainWindow(QWidget *parent = 0);
 4     ~MainWindow();
 private:

 5   FtpWindow *ftpWin;

 6   private slots:
7    void open();
                  …
};
```

Listing 2. Qt class definition

Line 1 in the header file shows the class is derived from the QMainWindow class which is one of the three GUI classes in Qt. The instance of the class (the object) will be derived ultimately from QObject. Line 2 shows the Q_OBJECT macro which must be declared in order to use signals and slots. Line 3 shows the constructor. QWidget will be the parent of the object, thereby making QWidget responsible for the deletion. Line 4 is the destructor and line 5 is a declaration of an object which will be used to download text files via FTP. This FTP object can be initialized in the constructor. Line 6 shows the slots used in the application. Slots must be declared in the class declaration with the keyword ‘slots’ preceded by either the private or public keyword as in line 6. Slots are declared just like C++ member function. The open slot in line 7 is a slot used for opening files with the appropriate file dialog. This slot is defined in the .cpp file as follows:

```
1 void MainWindow::open()

{

  2  if (maybeSave()) {

     3   QString fileName = QFileDialog::getOpenFileName(this);
      4 if (!fileName.isEmpty())

        5   loadFile(fileName);
```

```
   }

}
```

Listing 3. Qt slot implementation

If any object is interested in the open() slot, then the interested object's signal should be connected to the MainWindow's (which is the object to whom the slot belongs to) open() slot as follows:

```
1 MainWindow::MainWindow(QWidget *parent) :
 2  QMainWindow(parent),
 3   ui(new Ui::MainWindow)
{
4    ui->setupUi(this);
 5  createToolBars();
6connect(openToolButton,SIGNAL(clicked()),MainWindow,SLOT(open()))
```

Listing 4. A typical Qt constructor

Listing 4 shows a typical Qt Constructor. Line 4 sets up the user interface created by Qt Designer. Qt Designer is a fully fledged GUI designing tool.

As shown in listing 4, line 6, the appropriately named connect function is used to connect Signals with Slots. The first argument, openToolButton is a button in the tool bar. When this button object is clicked by the user, it emits the clicked() signal. The clicked signal is passed as an argument to the Signal macro and the Slot macro has the open() method passed to it. The SIGNAL() and SLOT() macros are used to specify the method to use. In Listing 4, the clicked() method and the open() method are used for this particular purpose.

Listing 4, line 6 shows that when the openToolButton is clicked, it emits the clicked() signal and when the clicked signal is emitted, the MainWindow object will execute the open() slot. Then the file open dialog will pop up on screen according to the code in listing 3.

A good example to demonstrate the parent-child paradigm in Qt is the partial code from listing 3, line 3: the getOpenFileName function shows a file dialog object and returns the file chosen by the user. The ‘this’ argument in the getOpenFileName function means the current object, which is MainWindow, is the owner of the file dialog object created by the getOpenFileName function, and as such the MainWindow object is responsible for the deletion of the file dialog object. Therefore, the file dialog object (child object) will be added to the MainWindow (parent object) object tree and will be removed along with other children when the MainWindow’s destructor is called. Alternately, the setParent() function can be used to set a particular object’s parent.

As demonstrated, Qt’s Signals and Slots mechanism is easy to use, intuitive and a good alternate to the conventional callback function. Signals and Slots can also be connected in Qt Designer by dragging a line from the signal object to the slot object provided both objects are GUI elements. This makes application development in Qt faster and more efficient.

3.3 Application Structure

3.3.1 Symbian Application Structure

Symbian uses the model-view-controller (MVC) pattern in application design. MVC is a common design pattern used in user interface applications in Symbian. MVC splits the application into three components: Model, View and Controller. [20]

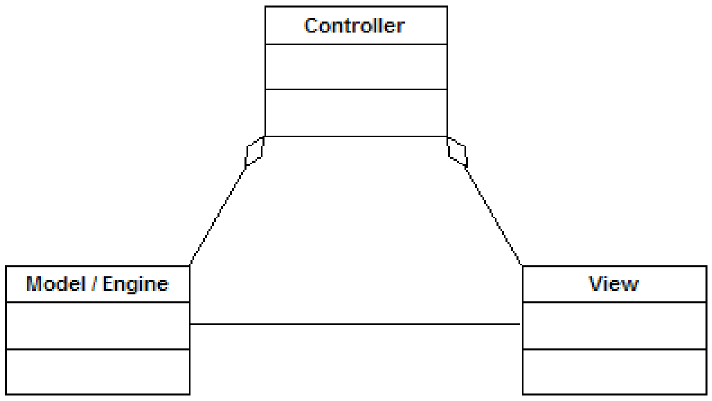


Figure 1. The Model-View-Architecture design pattern in Symbian[20]

Each component has a specific role to play in the overall scheme of the application.

Model:

- Encapsulates the application state.
- Exposes the application functionality.

- Notifies the view of changes.
- Stores program data in the model.

View:

- Renders the model.
- Receives updates from the model.
- Sends user input and commands to the controller.

Controller:

- Defines the application and reacts to received commands and requests.
- Maps user actions to Model updates.
- Selects views for response. [20]

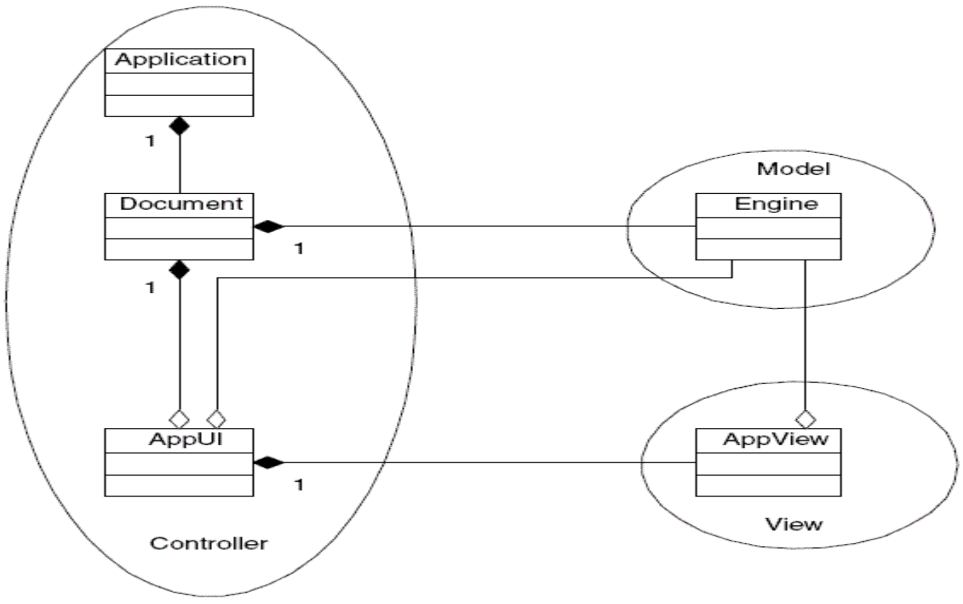
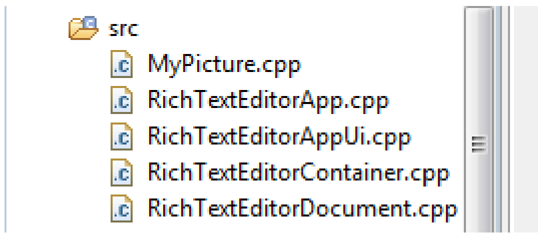


Figure 2. Symbian implementation of MVC. [20]

Figure 2 shows the MVC implementation in Symbian. The Application, Document and AppUi classes are used as the controller while the AppView class is used as the view and a user implemented engine as the model.

The Application class sets up and executes the application by supplying a globally unique 32-bit identifier (UID). The application properties are also defined here and the document class is also created in the application class. The Document class which is created in the Application class has application data ownership and responsibility for persisting data. This class creates the AppUi class. The AppUi class is used to handle application wide events such as menu commands, key presses, opening and closing files. These events can be passed to the views and container classes of an application. The AppView class along with other container classes displays data to the screen and collects data from the user. A view can be activated internally or externally by supplying the UID of the application and the ID of the view. [4, 79 - 80]



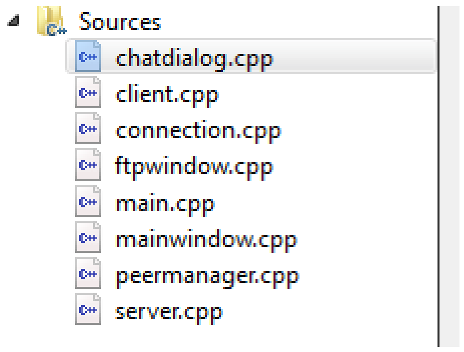
src
MyPicture.cpp
RichTextEditorApp.cpp
RichTextEditorAppUi.cpp
RichTextEditorContainer.cpp
RichTextEditorDocument.cpp

Figure 3. Project source file in Carbide C++ IDE

Figure 3 shows the source files for fWord. The source file names contain App, AppUi, Container and Document suffixes. The files are implementations of the MVC classes so to speak. MyPicture.cpp file is the Model\Engine implementation of the Application. RichTextEditorApp.cpp file is the application class implementation. The RichTextEditorAppUi.cpp file is the AppUi class implementation. The RichTextEditorDocument.cpp file is the Document class implementation and the RichTextEditorContainer.cpp file is a substitute implementation for the AppView class. A container class will suffice when developing small applications.

3.3.2 Qt Application Structure

The structure of a typical Qt application is mostly like a standard C++ application with header files, source files, resource files and forms.



Sources
chatdialog.cpp
client.cpp
connection.cpp
ftpwindow.cpp
main.cpp
mainwindow.cpp
peermanager.cpp
server.cpp

Figure 4. Project source folder

Figure 4 shows the project source folder. It can be observed that no peculiar naming conventions are used on file names to indicate underlying principles. They are just conventional C++ source files.

A slightly different version of the MVC architecture is employed in Qt. The Controller class is merged with the model and called Model\View (MV) architecture. Like the MVC architecture, the model\view architecture separates data storage from data presentation. The concept of the delegate is also introduced in MV architecture. A delegate allows the way items of data are rendered and edited to be customized.[10]

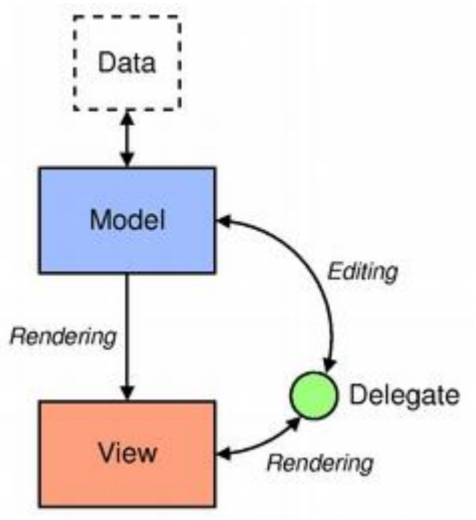


Figure 5. Qt’s Model\View architecture. [10]

Figure 5 illustrates the MV architecture in Qt. The model provides an interface for the other components by communicating with the source of data. The way the model is implemented and the type of data source determines the nature of communication. Model indexes, which are references of items of data, are obtained by the view from the model. The view uses these model indexes to retrieve data from the data source indirectly by supplying the model with the model index(es) of interest. A delegate communicates with the model directly using model indexes when an item is edited. A delegate renders the items of data if in standard views. Models, views and delegates communicate with each other using Signals and Slots. [10]

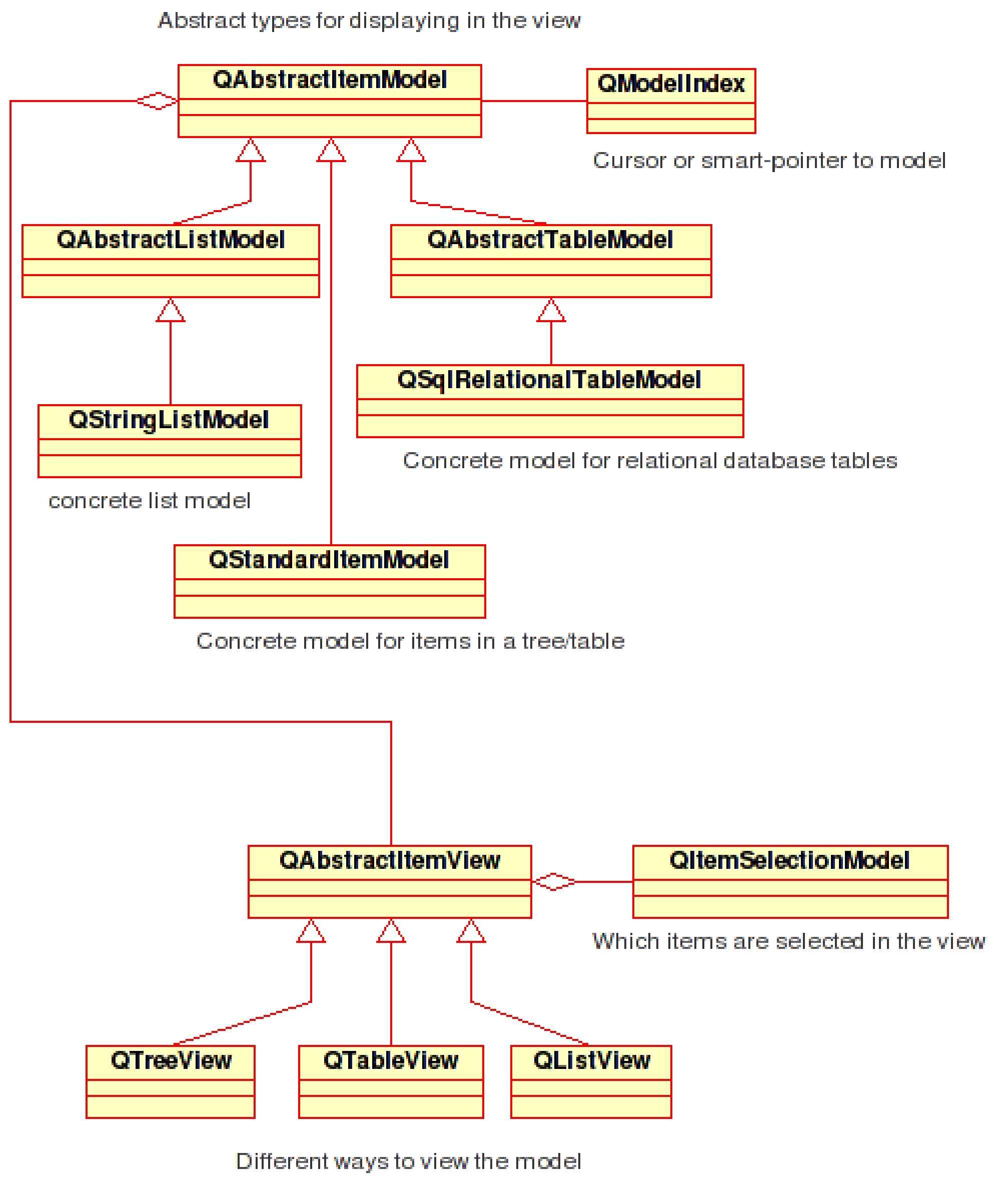

Abstract types for displaying in the view
QAbstractItemModel
QModelIndex
Cursor or smart-pointer to model
QAbstractListModel
QAbstractTableModel
QSqlRelationalTableModel
QStringListModel
Concrete model for relational database tables
concrete list model
QStandardItemModel
Concrete model for items in a tree/table
QAbstractItemView
QItemSelectionModel
Which items are selected in the view
QTreeView
QTableView
QListView
Different ways to view the model

Figure 6. The four main types of classes in Qt Model-View framework

Figure 6 shows the types of classes in Qt's Model-View framework. Unlike Symbian's implementation of MVC, which applies for any application other than a console application, Qt's MV framework is there if a developer has the need to use it. Developers oblivious to Qt's MV framework can still develop the desired application; in other words, it is optional not a necessity as in Symbian. Nevertheless, Symbian is a platform not a framework.

3.4 UI Implementation

3.4.1 Symbian GUI Implementation

There are three ways to develop a user interface in Symbian: the Traditional Symbian OS architecture, the S60 view architecture and the dialog based architecture.[10]

3.4.1.1 Dialog Based Architecture

In the dialog based architecture, the dominant parts of the views are dialogs. Resource files can be used to change the content and layout of the dialog without rebuilding the entire code.[10]

3.4.1.2 Traditional Symbian OS Architecture

In the traditional Symbian OS architecture, a UI controller based on the CAknAppUi class and a view based on the CCoeControl class should be implemented. Each CCoeControl object acts as a different view.[10]

This option is by far the most flexible way to construct a user interface in Symbian; the only catch is the developer has to do the view management like view switching.[10]

3.4.1.3 S60 View Architecture

The CAknViewAppUi class is used to implement a view management mechanism. The CAknViewAppUi class acts as a UI controller, the CAknView class acts as a view controller, and the CCoeControl acts as a view. Only one view is allowed to be active in each application in this architecture.[10]

The project application, fWord, is developed with the traditional Symbian OS architecture option.

```
class CRTEAppUi : public CAknAppUi
   {
   public:
. . .
```

Listing 5. The fWord controller class

Listing 5 shows the controller class for the project application.

```
class CRTEContainer : public CCoeControl
   {
   public:
. . .
```

Listing 6. The fWord view class

Listing 6 shows the view class for the project application.

The Carbide IDE contains a wizard for creating UI applications. A GUI application with a UI designer can be chosen from the ‘File > New > Symbian OS C++ project’ menu. The Carbide UI designer offers an interactive way to create UI components. Components can be dragged and dropped in the design area. Components can be added, deleted or repositioned until the desired UI layout is achieved. However, each component's functionality must be implemented in code.

3.4.2 Qt GUI Implementation

QWidget is the base class of UI objects in Qt. In Qt terminology, widgets are objects of classes derived from QWidget.

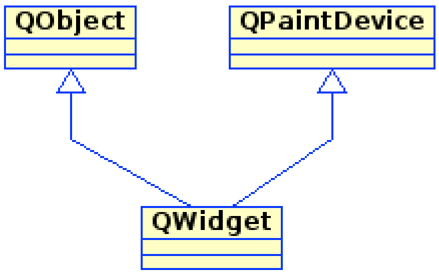


Figure 7. QWidget multiple inheritance

Figure 7 shows QWidget’s multiple inheritance. The inheritance from QObject makes QWidget a QObject. As a consequence, QWidget can implement the Signals and Slots mechanism, manage child objects, and acquire a parent itself. On the other hand, the inheritance from QPaintDevice, which is the base class of objects that can be painted to screen, makes Qwidget a QPaintDevice. [12]

Despite the complexity of the QWidget class and several hundred functions contained in it, QWidget, in its simplest form, is rendered as an empty box. UI objects in Qt like QPushButton and QLineEdit are implementations of the Qwidget class. One can develop one's own UI object by re-implementing the Qwidget class. There is a wide variety of readymade widgets available in Qt. [12]

A widget without a parent is called a window. Usually, applications have at least one such window. QMainWindow and QDialog are the most common window types in Qt. A widget's constructor accepts one or two standard arguments: the first one is Qwidget *parent argument. If this argument is specified as zero, then the widget will be a window. Otherwise, the widget will be the child of the parent. The second argument is used to set the window flags.

```
1class FtpWindow : public QDialog
 {
 2    Q_OBJECT
 public:
  3   FtpWindow(QWidget *parent = 0);
. . .
```

Listing 7. The fWord File Transfer Protocol (FTP) client constructor

As shown in listing 7, line 3, the parent is set to zero making the FtpWindow object a window. The FTP client can be made into a standalone FTP client application, but it is used as one tool amongst many in this project. Therefore, the application's main window has to take ownership of the FTP client object.

```
void MainWindow::startFtp()
{
   1 ftpWin = new FtpWindow(this);
. . .
```

Listing 7. The fWord FTP slot

Listing 7, line 1, shows that the MainWindow object takes ownership of the FTP client object by passing the 'this' pointer in the argument for the constructor. The 'this' pointer in this case refers to the MainWindow object. Thus the FTP client will be deleted when the MainWindow destructor is called.

A user interface can be designed in Qt either by manually typing the code or by using the Qt Designer. The project application, fWord, makes use of both ways ofthe UI design.

```
1 titleLabel = new QLabel;
2 titleLabel->setText("fWord: LANorWIFI Upload");
3 statusLabel = new QLabel;
4 statusLabel->setText("STATUS: ");
5 uploadButton = new QPushButton;
6 uploadButton->setText("Upload");
7quitButton = new QPushButton;
8quitButton->setText("Quit");
9fileTextEdit = new QTextEdit;
10statusLineEdit = new QLineEdit;
11buttonsLayout = new QHBoxLayout;
12buttonBox = new QDialogButtonBox;
13buttonBox->addButton(uploadButton,QDialogButtonBox::ActionRole);
14buttonBox->addButton(quitButton,QDialogButtonBox::RejectRole);
15buttonsLayout->addWidget(buttonBox);
16statusLabel->setBuddy(statusLineEdit);
17statusLabelAndEditLayout = new QHBoxLayout;
18statusLabelAndEditLayout->addWidget(statusLabel);
19statusLabelAndEditLayout->addWidget(statusLineEdit);
20mainLayout = new QVBoxLayout;
21mainLayout->addWidget(titleLabel);
22mainLayout->addWidget(fileTextEdit);
23mainLayout->addLayout(statusLabelAndEditLayout);
24mainLayout->addLayout(buttonsLayout);
25 setLayout(mainLayout);
```

Listing 8. UI design for fWord’s Wi-Fi File upload tool

Line 1 to line 10 in listing 8 instantiates a label object, button objects, a text editor object and a line edit object. These objects are assigned text where applicable. Lines 11,17 and 20 show how to create a layout. Line 11 creates a layout called buttonsLayout which will determine how the buttons are laid out on screen. There are three layouts in Qt: horizontal, vertical and grid. QHBoxLayout, QVBoxLayout and QGridLayout classes are used respectively to achieve the layouts. Line 12 creates a QDialogButtonBox which will be used to contain the upload button and the quit button created in lines 5 and 7 respectively.

Hence, buttonBox becomes the parent widget of the quitButton and the uploadButton when the addButton function is invoked as in line 13 and line 14. Lines 15, 18, 19, 21 and 22 show how a widget can be added to a layout. Lines 23 and 24 show how a layout can be added to another layout. Finally, line 25 sets the layout called mainLayout as the main layout which will be executed when the constructor is called. Lines 21 to 24 shows how widgets and layouts containing widgets can be added to the main layout.
Qt has a very dynamic, interactive and intuitive GUI building tool called QtDesigner. Besides laying out UI components in the desired fashion, QtDesigner also provides with a way to connect signal(s) of a UI component with slot(s) of another UI component by simply dragging an arrow from the former to the latter.

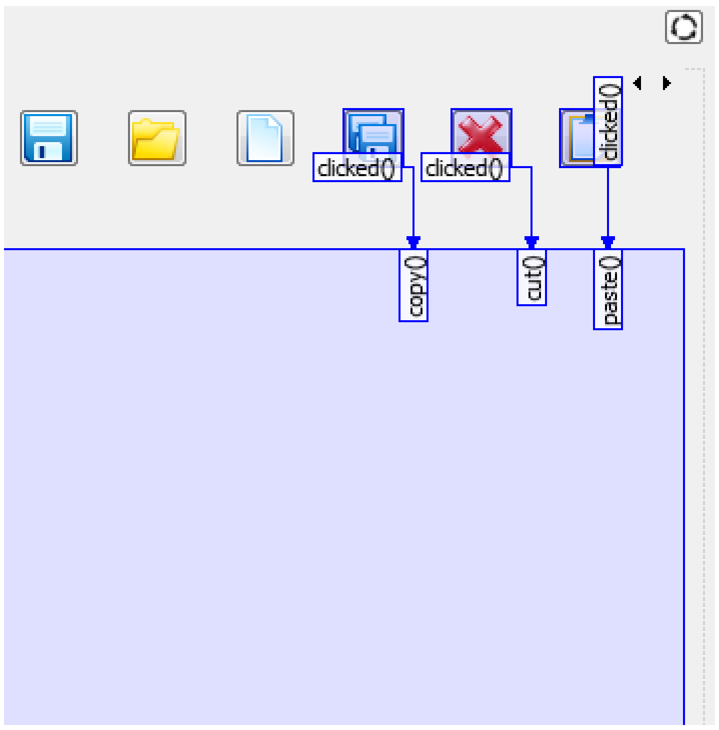


Figure 8. Qt Designer’s Signals and Slots editing feature for fWord application

As shown in figure 8, Qt Designer enables establishing and/or editing signals and slots by dragging a line from one object to another. The copy, cut and paste buttons are separately dragged to the QTextEditor object upon which a dialog will appear with available signals of the button on the left side of the screen and available slots of the text editor on the right side.

3.4.3 Symbian UI Designer vs. Qt Designer

The Symbian UI Designer and the Qt Designer each offer different ways to implement UI design.

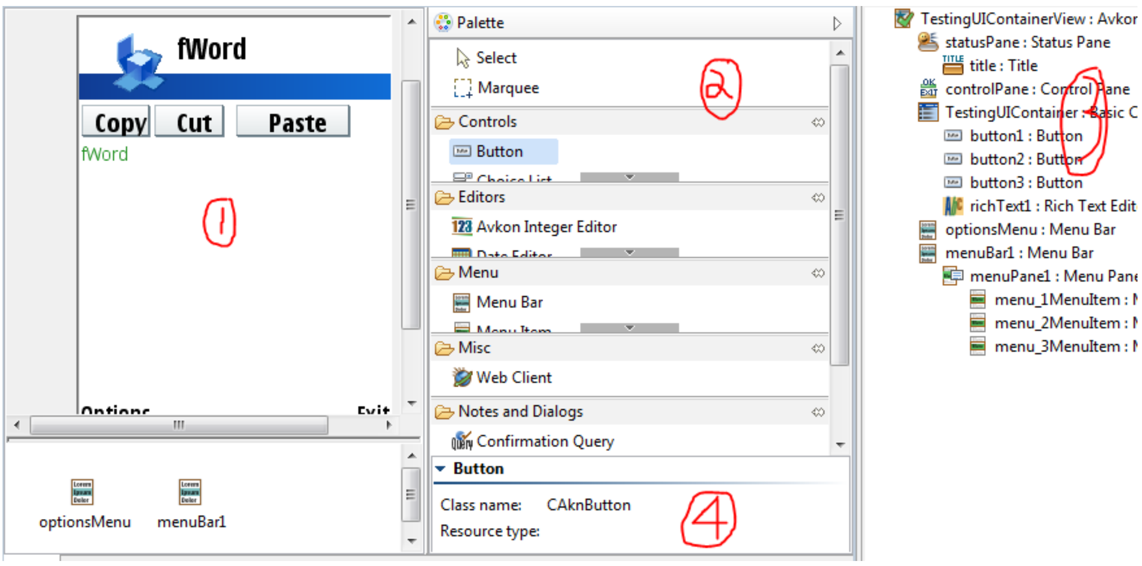
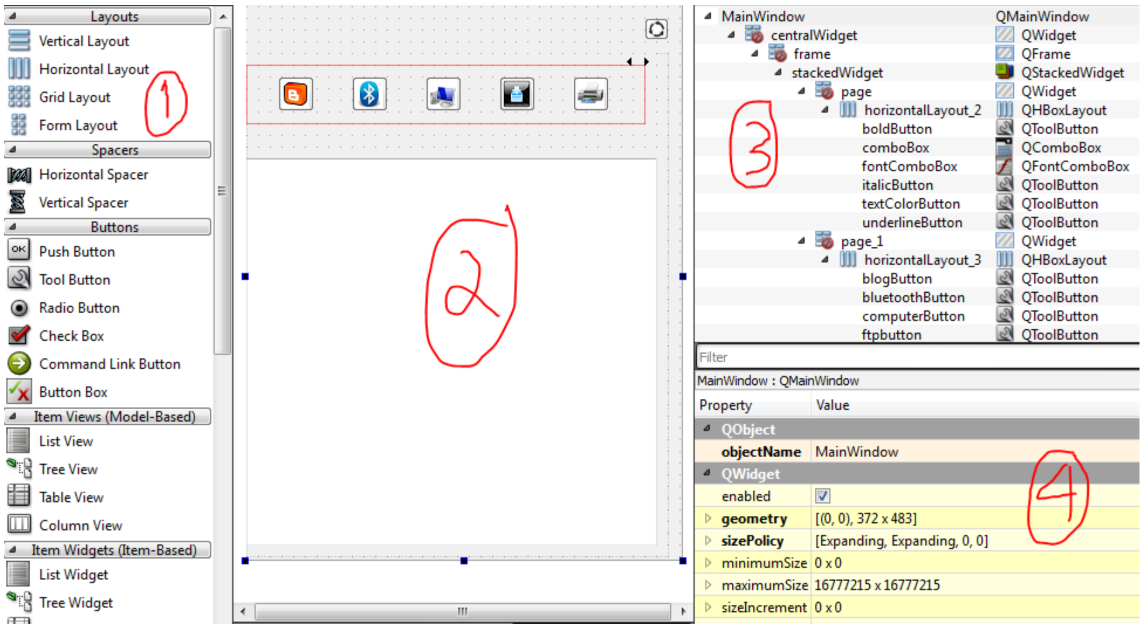


Figure 9. Symbian UI Designer

Figure 9 illustrates Symbian’s UI designer. The areas marked in red show the different working areas. Area 1 is the main working area where individual UI components can be dragged from area 2 and dropped in area 1. Area 2 shows the UI components available in Symbian. Area 4 shows the currently select UI component along with class name. Area 3 shows an outline of area 1’s UI components and their containers.

The UI components to choose from in area 2 are out of view most of the time. Searching for basic components by scrolling down can be tedious; the palette area (area 2) is not efficiently designed.



MainWindow	QMainWindow
centralWidget	QWidget
frame	QFrame
stackedWidget	QStackedWidget
page	QWidget
horizontalLayout_2	QHBoxLayout
boldButton	QToolButton
comboBox	QComboBox
fontComboBox	QFontComboBox
italicButton	QToolButton
textColorButton	QToolButton
underlineButton	QToolButton
page_1	QWidget
horizontalLayout_3	QHBoxLayout
blogButton	QToolButton
bluetoothButton	QToolButton
computerButton	QToolButton
ftpbutton	QToolButton

MainWindow : QMainWindow

Property	Value
QObject	
objectName	MainWindow
QWidget	
enabled	✓
geometry	[(0, 0), 372 x 483]
sizePolicy	[Expanding, Expanding, 0, 0]
minimumSize	0 x 0
maximumSize	16777215 x 16777215
sizeIncrement	0 x 0

Figure 10. Qt Designer used in the fWord Application

Figure 10 illustrates the use of Qt designer in the fWord project application. Area 1 marked in red, contains the UI components. Area 2 is the main working area where UI components from Area 1 can be dragged and dropped to. Area 3 outlines the objects used in area 2 along with their containers and class names. Area 4 is the property section where a UI component's property can be edited.

Qt Designer has advantages over the Symbian UI Designer in that the UI components in Area 1 are clearly and efficiently designed and presented. Signals and Slots can be connected in Qt Designer which would be tantamount to implementing a callback function in Symbian UI Designer had it been implemented. A UI component's properties can be edited directly in Qt Designer. A fully fledged application can be designed, implemented and deployed without writing a single line of code just by using the features of Qt designer. This cannot be achieved in Symbian UI Designer.

In general, Qt Designer has the upper hand in UI design because it has many features which are not available in the Symbian UI Designer but make UI designing seem effortless.

3.5 Files and Streams

3.5.1 Symbian Files and Streams

An RFs class provides a session to a file server.

```
_LIT(KDirectoryPath, "C:\\Private\\e033e32b\\");
RFs& fsSession = CCoeEnv::Static()->FsSession();
User::LeaveIfError(fsSession.Connect());
CleanupClosePushL(fsSession);

RFile fWordfile;
User::LeaveIfError(fWordfile.Write(KWriteData));
CleanupStack::PopAndDestroy(&fsSession);
```

Listing 9. Symbian file server connection and write operation

As shown in listing 9, a session to a file server must be established prior to any operation involving the file system. Subsequently, data can be written to or read from a file. [4, 155]

The traditional way of opening files with the help of a dialog is done by deriving a caknfileselectiondialog object from the caknfileselectiondialog.h header file and linking it against the commondialogs.lib library. The process of creating these dialogs lacks a certain level of abstraction.

A file operation can also be done using streams. Streams provide a way to read from and write to a file with RReadStream and RWriteStream objects.[4,156]

```
_LIT(KDirectoryPath, "C:\\Private\\e033e32b\\");
RFs& fsSession = CCoeEnv::Static()->FsSession();
User::LeaveIfError(fsSession.Connect());
CleanupClosePushL(fsSession);
RFileReadStream readStream;
TInt error = readStream.Open(fsSession, KParentFileName, EFileRead);
```

Listing 10. Symbian streams: read stream

Listing 10 shows how to open a file stream for reading. The other structure available for file manipulation is a store. Persistent objects can be implemented by using a store. A store is a collection of related streams. There are a number of different stores derived from the base store CStreamStore. The most common use of a store is in a relational database.

3.5.2 Qt Files and Streams

Qt provides support for input and output operation through the QIODevice abstraction class. QIODevice subclasses are:

- •QFile - Accesses files in the local file system and in embedded resources.
- •QTemporaryFile - Creates and accesses temporary files in the local file system.
- •QBuffer - Reads data from or writes data to a QByteArray.
- •QProcess - Runs external programs and handles inter-process communication.
- •QTcpSocket - Transfers a stream of data over the network using TCP.
- •QUdpSocket - Sends or receives UDP datagrams over the network.
- •QSslSocket - Transfers an encrypted data stream over the network using SSL/TLS. [14].

QIODevice cannot be instantiated directly since it is an abstract class but one of its sub-classes can be.

Qt also provides with the QFileDialog class for file operation dialogs. The abstraction level provided in QFileDialog makes file operation management very easy.

```
void MainWindow::open()
{
  if (maybeSave()) {
    QString fileName = QFileDialog::getOpenFileName(this);
    if (!fileName.isEmpty())
      loadFile(fileName);
  }
}
```

Listing 11. File open slot

Listing 11, line 3, shows how to launch a file open dialog. The getOpenFileName function returns a string to the file selected by the user. This string is used as the file name to open as shown in line 5. The load file function loads the file.

```
1 void MainWindow::loadFile(const QString &fileName)
{
  2  QFile file(fileName);
   3 if (!file.open(QFile::ReadOnly | QFile::Text)) {
    4    QMessageBox::warning(this, tr("fWord"),
                         tr("Cannot read file %1:\n%2.")
                         .arg(fileName)
                         .arg(file.errorString()));
        return;
    }
    5 QTextStream in(&file);
    6 QApplication::setOverrideCursor(Qt::WaitCursor);
    7 ui->textEdit->setPlainText(in.readAll());
    8 QApplication::restoreOverrideCursor();
    9 setCurrentFile(fileName);
    10 statusBar()->showMessage(tr("File loaded"), 2000);
}
```

Listing 12. The load file function

Listing 12 shows how the load file function loads the file to the rich text editor screen. One of the QIODevice subclasses, QFile, is used to open the file with the name acquired in listing 11, line 3. Lines 3 and 4 of listing 12 show the process of opening the file and in the case where an error occurs, how a message box will notify the user about the current situation and the function exits. Line 5 of listing 12 shows the use of text streams in Qt. QTextStream provides a convenient interface for text operations. The only operation remaining is to show the text file in the rich text editor; this is done as shown in line 7 of listing 12 by passing the entire content of the stream as a string to the text editor’s setPlainText function.

3.6 Strings

3.6.1 Symbian Strings

Symbian strings are called descriptors. They are used to manipulate binary data stored in memory in addition to text processing. Descriptors are categorized as modifiable or non-modifiable with the possible location base of stack, heap or pointer. [4, 84]

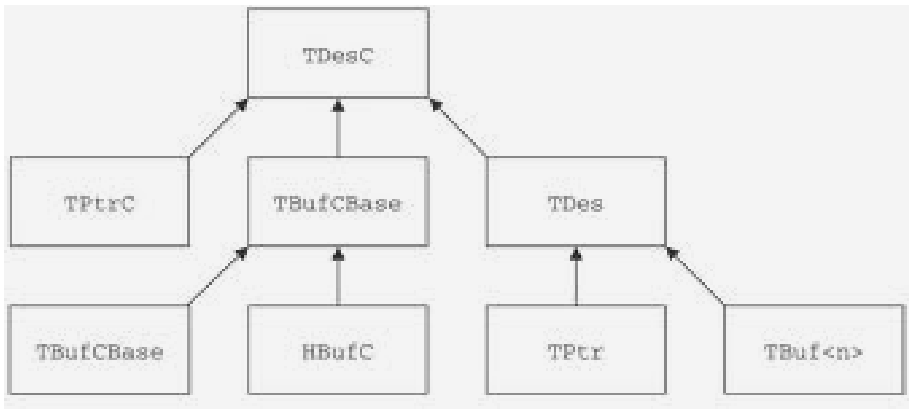


Figure 11. Symbian OS Descriptors

Figure 11 shows the descriptor hierarchy in Symbian OS.
The base class for all descriptors is the non-modifiable abstract class, TDesC. The TDesC class cannot be instantiated or modified directly but it can be used to pass a non modifiable data to functions. [4, 85]

Another abstract class similar to TDesC is TDes. The only difference between the two classes is TDes has a maximum length parameter. As long as the TDes descriptor does not exceed the maximum length, it can be modified. [4, 85]
TPtrC and TPtr are pointer descriptors. The former is non-modifiable while the latter is modifiable. TPtr descriptors contain a maximum length parameter. It also has methods used to modify the string. [4, 85]

HBufC descriptors are heap based non-modifiable descriptors. These descriptors are used for fixed length text and binary data. The size of HBufC descriptors is not known until run time. Even though HBufC descriptors are non-modifiable, a workaround to make modifications is to use the Des() method as shown below in listing 13. [4, 86]

```
_LIT(KMessage, "Hello");
HBufC* myMessage = HBufC8::NewL(KMaxItemLength);
myMessage->Des().Append(KMessage);
```

Listing 13. How to modify an HBufC descriptor

The Symbian OS provides methods to manipulate descriptors of both non-modifiable and modifiable nature. Some of the most common methods are listed in table 1 below:

Table 1. Descriptor methods

Modifiable Descriptor Methods	**Non-Modifiable Descriptor Methods**
Append	Alloc
Capitalize	AllocL
Uppercase	AllocLC
Lowercase	Compare
Copy	CompareC
Delete	ComapareF
Fill	Find
Format	FindC
	FindF

Symbian OS strings can also be defined by using literals.

3.6.2 Qt Strings

QString is Qt's version of a string. QString is a Unicode character string which stores a string of 16-bit QChars. QChars correspond to a character in QString. QChars are Qt's implementation of a character. QString based applications can be translated to other languages since QChars provide Unicode support. [15]

An alternate to QString strings and particularly to 'const char*' strings in Qt is QByteArray. QByteArray is mostly used to store raw binary data, store traditional Null terminated strings or in memory critical applications [16].

```
 void ChatDialog::returnPressed()
{
   1 QString text =fileTextEdit->toPlainText();
   2 if (text.isEmpty())
     3   return;
   4 else {
       5 client.sendMessage(text);
       6 appendMessage(myNickName, text);
   }
}
```

Listing 14. String initialization and manipulation in Qt

Listing 14 shows one of the slots in the Rich Text document application. A QString called text is created and initialized with the content of the rich text editor as shown in line 1. Line 2 checks if the string is empty and if it is empty, the function returns.
Qt's string manipulation methods are similar to standard C++ string manipulation methods.

Table 2. some common Qt string manipulation method

Methods
Append
Begin
Capacity
Length
Compare
Contains
Count
Insert
Remove
Replace

3.7 Communications

The primary purpose of mobile phones is to establish communication between two parties. There are various means of achieving communication besides the conventional voice communication: text communication, video communication.

3.7.1Communications in Symbian

Communication in Symbian is made through the Comms, short for communications, service block. The Comms services block is a major, self-contained block within the OS service layer of Symbian OS. [17, 224]

A wide variety of communications protocols and services are supported by the Comms service:

- •Serial communication
- •Bluetooth communication
- •Network communications
- •Wi-Fi
- •Mobile telephony communications. [17, 226]

The Comms service block is divided into four sub blocks:

- •Comms framework – Provides a generic infrastructure supporting all communication services.
- •Telephony services – Provides support for 2G, 2.5G, 3G , 4G and CDMA mobile phone networks.
- •Short link services – Provides USB, Bluetooth and infrared services.
- •Networking services – Provides packet based network services. Implements TCP/IP, FTP and HTTP. [17, 227 – 228]

3.7.1.1 The Comms Framework

All communication services use the infrastructure implemented by the Comms framework. The primary communications server is the Comms Root Server which is responsible for starting and stopping the communication servers. There are two direct client interfaces for communication services: The first is the C32 serial server which is the data communication server and the second being the ESock Socket server which self descriptively provides a socket server. There are also interfaces that handle socket server client requests namely the Network Interface Manager and the Network Controller. The former is the network interface manager while the latter is the connection manager. Both of these managers find and setup appropriate network connections as requested by the socket server clients. [17, 235 – 236]

Table 3. Components of the data Comms server

Component Name	Development Name
C32 Serial Server	c32
ESock Server	ESOCK
Network Interface Manager	NIFMAN, DIALOG
Network Controller	NETCON

Table 3 shows the components of the data Comms Server

3.7.1.2 Telephony Services

The primary telephony service server is the ETel Telephony server. Access to telephony functions on a Symbian OS device is managed by the ETel telephony server. This server implements the standard Symbian client-server framework and in the process provides a client side API. The CPM interface is also implemented by the ETel server which makes it a communications provider that is managed and run by the Comms Root Server. The telephony server provides basic abstractions as phones, lines and calls. Clients open sub-sessions to phone, line and call after opening a server session with the telephony server. Support for messaging is included in the ETel multimode extension. [17, 245 – 251]

Table 4. Telephony server components

Component Name	Development Name
ETel Server and Core	ETEL
ETel 3rd party API	ETEL3RDPARTY
Fax Client and Server	FAX
ETel Multimode	ETELMM
ETel Packet Data	ETELPCKT
ETel SIM Toolkit	ETELSAT
ETel CDMA	ETELCDMA

Table 4 shows the various components of the Telephony server.

Listing 15 demonstrates how telephony services are used in a messaging application.

```
iMsvSession = CMsvSession::OpenAsyncL(*this);

iClientMtmRegistry = CClientMtmRegistry::NewL(*iMsvSession);

iSmsMtm = static_cast<CSmsClientMtm*>(iClientMtmRegistry->
NewMtmL(KUidMsgTypeSMS));

TMsvEntry msvEntry;
msvEntry.SetInPreparation(ETrue);
msvEntry.iMtm = KUidMsgTypeSMS;
msvEntry.iType = KUidMsvMessageEntry;
msvEntry.iServiceId = iSmsMtmServiceId();

iSmsMtm->Entry()->CreateL(msvEntry);
iMessageId = msvEntry.Id();
iSmsMtm->SwitchCurrentEntryL(iMessageId);

iSmsMtm->Entry().ChangeL(msvEntry);

iSmsMtm->SwitchCurrentEntryL(iMessageId);
iSmsMtmLoadMessageL();
CSmsSettings& smsSettings = iSmsMtm->ServiceSettings();
CSmsNumber * serviceCenter =
&(smsSettings.SCAddress(smsSettings.DefaultSC()));
iSmsMtm->SetServiceCenterAddressL(serviceCenter->Address());
iSmsMtm->SaveMessageL();

msvEntry.SetInPreparation = EFalse;
msvEntry.SetSendingState(KMsvSendStateWaiting);
iSmsMtm->Entry().ChangeL(msvEntry);

iMsvEntrySelection->AppendL(iMessageId);
iMsvOperation = iSmsMtm->InvokeASyncFunctionL( ESmsMtmCom-
mandScheduleCopy,*iMSvEntrySelection, NULL, iStatus);
```

Listing 15. An SMS application demonstrating telephony services

3.7.1.3 Networking Services

The networking services were created primarily to accommodate web and email services. The TCP\IP protocol implementation is the basis for the symbian OS networking services. However, The TCP\IP packets and the stack itself are not directly available. The packets are encapsulated within the stack and the stack is implemented as a socket server PRT protocol plug-in. The socket interface provides access to network services. [17, 255]

Table 5. Network protocol plug-ins.

Component Name	**Development Name**
IP Event Notifier	IPEVENTNOTIFIER
TCP/IPv4/v6 PRT	TCPIP6
IP Hook	INHOOK6
QoS Framework PRT	QOS, QOSLIB, PFQOSLIB,SBLPAPI
Core IPSec PRT	No Unit

Table 5 shows the various network protocol plug-ins.

```
 HBufC8* iMessage;
 RSocketServ iSocketServer;
RSocket iSocket;
TInetAddr iAddress;
User::LeaveIfError(  iSocket.Open(  iSocketServer,KAfInet,KSockStream,
                              KProtocolInetTcp ) );
iSocket.Connect( iAddress, iStatus );
User::LeaveIfError( iResolver.Open( iSocketServer,
      KAfInet,KProtocolInetTcp ) );
    iResolver.GetByName( aAddress, iNameEntry, iStatus );
              iSocket->Write( *iMessage, iStatus );
iSocket->CancelRead();
 iSocket.Close();
iSocketServer.Close();
```

Listing 16. Network services

Listing 16 shows how to connect to a socket using network services.

3.7.1.4 Short-link Services

Short link services provide implementations to serial, USB, infrared and Bluetooth functionalities while infrared communications protocol or IrDA contains a complete set of protocols from application level to link level [17, 272].

Bluetooth communications define a complete protocol stack. Among the Bluetooth components are:

•The Bluetooth Manager – details of local and remote devices are stored in the Bluetooth Manager which acts as the information store. The Bluetooth Manager is implemented over Symbian OS DBMS.
•The Bluetooth Service Discovery protocol (SDP) – Bluetooth devices can find each other and share information through this protocol.
•The Bluetooth Host Controller Interface (HCI) – The Bluetooth stack is interfaced with the on board controller through this interface. [17, 273]

Symbian OS implements a serial over USB (USB CSY) class, mass storage and OBEX over USB classes. The USB manager implements a server interface for USB class implementations.

Table 6: Serial Comms server plug-ins components

Component Name	**Development Name**
Serial Port CSY	ECUART
USB CSY	ECACM
Bluetooth CSY	BTCOMM
IrDA CSY	IRCOMM

Table 6 shows the plug-in components available for the serial comms server.

Listing 17 shows how to discover Bluetooth devices.

```
RSocketServ sockServ;
sockServ.Connect();
TProtocolDesc pd;
_LIT(KL2Cap, "BTLinkManager");
```

```
5 User::LeaveIfError(sockServ.FindProtocol(KL2Cap, pd));
6 RHostResolver hr;
7 User::LeaveIfError(hr.Open(sockServ, pd.iAddrFamily, pd.iProtocol));
8 TInquirySockAddr addr;
9 TNameEntry entry;
10 addr.SetIAC(KGIAC);
11 addr.SetAction(KHostResInquiry);
12 TRequestStatus status;
13 hr.GetByAddress(addr, entry, status);
14 User::WaitForRequest(status);
```

Listing 17: Short link Bluetooth services example

Lines1 to 5 of listing 17 show how to connect to the socket server, RSocketServer and how to choose the protocol to be used for the connection. Lines 6 and 7 create the host name resolution service and initialize it. The rest of the lines of code show how to set up the TInquirySockAddr variable to have a 'general unlimited' inquiry access. Line 13 starts this inquiry process.

3.7.2 Communications in Qt

The term communications encompasses a wide variety of methods from making a call to a Bluetooth file upload. Unfortunately, Qt does not provide direct support for most of the communication protocols provided by Symbian OS. The word communications in Qt is more tuned to the idea of networking. Qt does not provide direct support to USB, Infrared or Bluetooth services. However, Qt provides an impeccable network programming implementation via an assortment of classes as shown in table 7.

Table 7. QtNetwork classes for network programming

Class Name	**Description**
QAbstractSocket	Provides base functionality common to all socket types
QAuthenticator	Authentication object
QFtp	Client side FTP protocol implementation
AHostAddress	IP address
QHostInfo	Host name lookup static functions
QNetworkAccessManager	Sends network requests and receives replies

QNetworkAddressEntry	Stores an IP address
QNetworkInteface	Host’s IP addresses and network interfaces listing
QNetworkProxy	Network layer proxy
QNetworkProxyFactory	Specialized proxy selection
QNetworkReply	Contains the data and headers for a request sent with QNetworkAccessManager
QNetworkRequest	Holds a request to be sent with QNetworkAccessManager
QSocketNotifier	Support for monitoring activity on a file descriptor
QSsl	Declares enumerations common to all SSL classes in QtNetwork
QSslCertification	Convenient API for an X509 certificate
QSslCipher	Represents an SSL cryptographic cipher
QSslConfiguration	Holds the configuration and state of an SSL connection
QSslError	SSL error
QSslKey	Interface for private and public keys
QSslSocket	SSL encrypted socket for clients and servers
QTcpServer	TCP-based server
QTcpSocket	TCP socket
QUdpSocket	UDP Socket
QUrl	Convinient interface for working with URLs
QUrlInfo	Stores information about URLs

As table 7 shows, Qt provides a variety of classes for network programming.

Listing 18 shows how two of Qt's networking classes, QFtp and QUrl, can be used in code.

```
void FtpWindow::connectOrDisconnect()
 {
1 ftp = new QFtp(this);
2 QUrl url(ftpServerLineEdit->text());
 if (!url.isValid() || url.scheme().toLower() !=QLatin1String("ftp"))
{
   3      ftp->connectToHost(ftpServerLineEdit->text(), 21);
     4    ftp->login();
} else {
        ftp->connectToHost(url.host(), url.port(21));
        if (!url.userName().isEmpty())
   ftp->login(QUrl::fromPercentEncoding(url.userName().toLatin1()),
url.password());
        else
            ftp->login();
        if (!url.path().isEmpty())
            ftp->cd(url.path());
    }
```

Listing 18. QFtp and QUrl classes usage

Line 1 of listing 18 shows how to create an FTP object. On line 2 the host address is acquired from the text entered into the provided form by the user. Line 3 and 4 show how to connect to the host address and how to login to the address respectively.
The FTP program shown in listing 18 can also be designed from scratch using Qt's socket classes.

3.8 Threads and Processes

A process is an executing program and one or more processes are contained in an application [19]. Processes are self contained; a process cannot affect the state of another process directly [18]. Moreover, any two processes do not share the same address space and resources [18].

A thread is a unit of execution within a process. All threads of a process share the process's virtual address space and resources. [19]

A single primary thread is initialized within a process when the process is first created. Consequently, other threads may be created as required. [5, 141]

3.8.1Symbian Threads and Processes

There are two types of multitasking in Symbian: pre-emptive and cooperative.
In pre-emptive multitasking, which is based on allocating a time-slice for each process, the kernel, which controls thread scheduling, may suspend or resume a thread when appropriate. This constant switching between threads is called a context switch. A context switch occurs when a higher priority thread is ready to run or when the currently running thread is suspended. [5, 141]

In cooperative multitasking, each process has the responsibility to relinquish processing time to allow other processes to execute. [5, 142]

Active objects and threads (RThread) are Symbian OS implementations of co-operative and pre-emptive multitasking respectively.

3.8.1.1 Active Objects

Active objects are a Symbian OS implementation of cooperative multitasking; "multiple active objects execute in effect within the context of a single thread." [17, 81]

Active objects come into play when an event or events is/are generated by a user or a service provider. The application receives a notification when the request generated by the event is complete. Then, the active object starts handling the request. There is an active object waiting for the request to be completed for each asynchronous service request. Typical to a cooperative multitasking technique, there is a wait loop going through the outstanding task requests and when a completed task is found by the wait loop, it calls for the event handler code of the corresponding handler object. The wait loop is implemented as an active scheduler and the handler objects are implemented as active objects. [20]

```
1class CMyTimerAo : public CActive
{
public:
            CMyTimerAo();
            ~CMyTimerAo();
            void Start();
            void RunL();
private:
            void DoCancel();
private:
  // An integer representing a timing period
            TInt iPediod;
            // A handle to timing services
            RTimer iTimer;
};
```

Listing 19. Active object declaration

Listing 19, line 1, shows how to derive an active object from CActive class.
In general, active objects provide requests to asynchronous services, and once completed they handle the requests. [20]

3.8.1.2 RThread

Threads are scheduled preemptively by the kernel while active objects multitask cooperatively within a thread and cannot be preempted by the active scheduler. [5, 181]

The RThread class is used to manipulate threads, and an RThread object can be used to create or refer to another thread. [5, 183]

```
RHeap* myHeap = RThread().Heap();
```

Listing 20. RThread Object

Listing 20 shows how to use an RThread object to handle to the current thread.

3.8.2 Qt Threads and Processes

There are two classes in Qt in charge of processes and threads: QProcess and QThread. QProcess is used to launch and communicate with external programs while a QThread instance is used to represent and execute a thread.

```
QObject *parent;
QString program = "./QtApps/fWord";
QStringList arguments;
arguments << "-style" << "motif";
QProcess *myProcess = new QProcess(parent);
myProcess->start(program, arguments);
```

Listing 21. Qprocess usage

Listing 21 shows how to start a process by launching an external program, in this case the program fWord.

```
1 class MyThread : public QThread
 {
   2  Q_OBJECT
 protected:
     3 void run();
 };
 4 void MyThread::run()
 {
     ...
 }
```

Listing 22. QThread initialization

Listing 22 shows how to create a thread by subclassing the QThread class. The run function on line 3 should be implemented on line 4 to make the thread usable.

4 Conclusion

This project has shown the introduction of Qt, as a major overhaul to the Symbian based Smartphone and a practically better alternate to Symbian C++ programming, gaining the upper hand over Symbian C++ in almost every category, from fast development environment setup and easy and intuitive class and object implementation to a simple UI designer and uncomplicated string implementation. Moreover, Qt can deliver a fast and relevant user experience demanded by today’s end user.

Touch screen, interactive smart phones offer a fluid and fun experience to users. None of the leading smartphone companies invented the touch screen but rather made innovative use out of it. Such innovative design translates to the difference between a successful brand and a slowly receding brand.

Technically speaking, Qt is being/will be wrapped around the existing Symbian OS platform. Developers retain the choice of developing with Qt or dealing with Symbian OS directly. As examined in this thesis, Qt offers the easiest development route. Qt may not be the messianic framework that saves the Nokia smartphone, but its potential to take Nokia to the next level is high.

References

1 Aubert, Michael. Quick recipes on Symbian OS. West Sussex, England: John Wiley and Sons Ltd; 2008.

2 Qt4 graphical user interface [online].
URL: http://www.civilnet.cn/book/embedded/gui/qt4/pref04.html.
Accessed 10 April 2011.

3 Symbian.org [online].
http://developer.symbian.org/wiki/index.php/Using_Qt_with_Standalone_SDKs.
Accessed 10 August 2010.

4 Coulton Paul, Edwards, Reuben. S60 programming: A tutorial guide.
West Sussex, England: Wiley and sons; 2007.

5 Stichbury Jo. Symbian OS explained. West Sussex, England: Wiley; 2004.

6 QObject [online].
URL: http://doc.qt.nokia.com/4.6/qobject.html#details.
Accessed 3 November 2010.

7 Object trees [online].
URL: http://doc.qt.nokia.com/4.6/objecttrees.html.
Accessed 3 November 2010.

8 Qt for Symbian white paper [online].
URL: http://qt.nokia.com/products/platform/files/pdf/whitepaper-qt-for-the-symbian-platform.
Accessed 3 November 2010.

9 QObject [online].
URL: http://cartan.cas.suffolk.edu/oopdocbook/opensource/qobject.html.
Accessed 5 November 2010.

10 Model-View introduction [online].
URL: http://doc.trolltech.com/4.6/model-view-introduction.html.
Accessed 6 November 2010.

11 S60 view architecture with UI Design [online].
URL: http://wiki.forum.nokia.com/index.php/S60_View_Architecture_with_UI_Design.
Accessed 6 November 2010.

12 QWidget [online].
URL: http://cartan.cas.suffolk.edu/oopdocbook/opensource/widgets.html.
Accessed 8 November 2010.

13 QWidget [online].
URL: http://doc.qt.nokia.com/4.6/qwidget.html#details.
Accessed 8 November 2010

14 Blanchette Jasmin, Summerfield Mark. C++ GUI programming with Qt 4, Second edition. Westford Massachusetts. Prentice Hall; 2008.

15 QString [online].
URL: http://doc.qt.nokia.com/4.6/qstring.html#details.
Accessed 20 November 2010.

16 QByteArray [online].
URL: http://doc.qt.nokia.com/4.6/qbytearray.html#details.
Accessed 20 November 2010.

17 Morris, Ben. The Symbian OS architecture source book. West Sussex, England: Wiley; 2007.

18 Threads and processes [online].
URL: http://williamstallings.com/Extras/OS-Notes/h2.html
Accessed 20 November 2010.

19 Threads and processes [online].
URL: http://msdn.microsoft.com/en-us/library/ms684841(v=vs.85).aspx.
Accessed 20 November 2010.

20 Klemetti, Aarne. Symbian Programming [Lecture Slides].
Espoo, Finland. Metropolia University of Applied Sciences; 2010.